FSAN/ELEG815: Statistical Learning

Gonzalo R. Arce

Department of Electrical and Computer Engineering
University of Delaware

5a: The Linear Model and Optimization

Linear Regression - Credit Example

Regression $\rightarrow$ Continuous real-valued output

Classification: Credit approval (yes/no)
Regression: Credit line (dollar amount)

Input: $\mathbf{x} =$

age	23 years
gender	male
annual salary	\$30,000
years in residence	1 year
years in job	1 year
current debt	\$15,000
...	...

Linear regression output: $h(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i = \mathbf{w}^\top \mathbf{x}$

Credit Example Again - The data set

Input: $\mathbf{x} =$

age	23 years
gender	male
annual salary	$30,000
years in residence	1 year
years in job	1 year
current debt	$15,000
...	...

Output:

$$h(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i = \mathbf{w}^\top \mathbf{x}$$

Credit officers decide on credit lines:

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$$

$y_n \in \mathbb{R}$ is the credit for customer $\mathbf{x}_n$.

Linear regression wants to automate this task, trying to replicate human experts decisions.

Linear Regression

Linear regression algorithm is based on minimizing the squared error:

$$E_{out}(h) = \mathbb{E}[(h(\mathbf{x}) - y)^2]$$

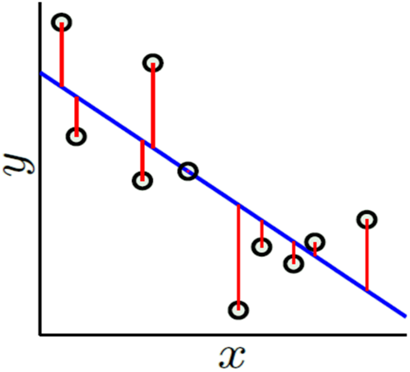
where $\mathbb{E}[\cdot]$ is taken with respect to $P(\mathbf{x}, y)$ that is unknown.
Thus, minimize the in-sample error:

$$E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n) - y_n)^2$$

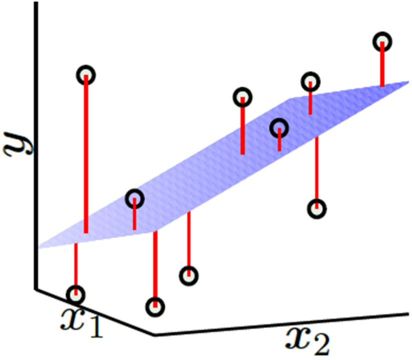
Find a hypothesis ($\mathbf{w}$) that achieves a small E_{in}.

Illustration of Linear Regression

The solution hypothesis (in blue) of the linear regression algorithm in one and two dimensions input. The sum of square error is minimized.



One dimension (line)



Two dimensions (hyperplane)

Linear Regression - The Expression for E_{in}

Linear regression: $\mathbf{y} = w_0\mathbf{1} + w_1\mathbf{x}_1 + w_2\mathbf{x}_2 + \ldots + w_d\mathbf{x}_d + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathbf{N}(\mathbf{0}, \sigma^2\mathbf{I}).$

Estimation:
$$\underbrace{\begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_N \end{bmatrix}}_{\hat{\mathbf{y}} \in \mathbb{R}^N} = \underbrace{\begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nd} \end{bmatrix}}_{\mathbf{X} \in \mathbb{R}^{N \times (d+1)}} \cdot \underbrace{\begin{bmatrix} \hat{w}_0 \\ \hat{w}_1 \\ \vdots \\ \hat{w}_d \end{bmatrix}}_{\hat{\mathbf{w}} \in \mathbb{R}^{(d+1)}}$$

$$\begin{aligned} E_{in} &= \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2 \\ &= \frac{1}{N}||\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}||_2^2 = \frac{1}{N}(\mathbf{X}\hat{\mathbf{w}} - \mathbf{y})^\top(\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}) \\ &= \frac{1}{N}(\hat{\mathbf{w}}^\top\mathbf{X}^\top\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}^\top\mathbf{X}\hat{\mathbf{w}} - \hat{\mathbf{w}}^\top\mathbf{X}^\top\mathbf{y} + \mathbf{y}^\top\mathbf{y}) \\ &= \frac{1}{N}(\hat{\mathbf{w}}^\top\mathbf{X}^\top\mathbf{X}\hat{\mathbf{w}} - 2\hat{\mathbf{w}}^\top\mathbf{X}^\top\mathbf{y} + \mathbf{y}^\top\mathbf{y}) \end{aligned}$$

Learning Algorithm - Minimizing E_{in}

$$\begin{aligned}\hat{\mathbf{w}} &= \arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{N}||\mathbf{X}\mathbf{w}-\mathbf{y}||_2^2 \\ &= \arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{N}(\mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w}-2\mathbf{w}^\top\mathbf{X}^\top\mathbf{y}+\mathbf{y}^\top\mathbf{y})\end{aligned}$$

Observation: The error is a quadratic function of $\mathbf{w}$

Consequences: The error is an $(d+1)$–dimensional bowl–shaped function of $\mathbf{w}$ with a unique minimum

Result: The optimal weight vector, $\mathbf{w}$, is determined by differentiating $E_{in}(\mathbf{w})$ and setting the result to zero

$$\nabla_{\mathbf{w}} E_{in}(\mathbf{w}) = 0$$

- A closed form solution exists

Example

Consider a two dimensional case. Plot the error surface and error contours.

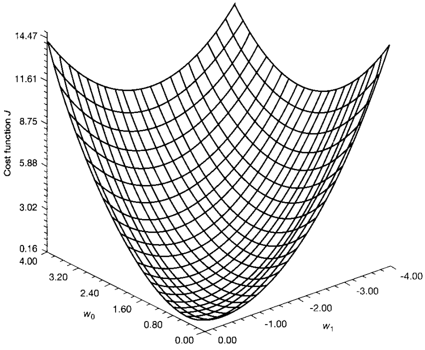


Figure 5.6 Error-performance surface of the two-tap transversal filter described in the numerical example.

Error Surface

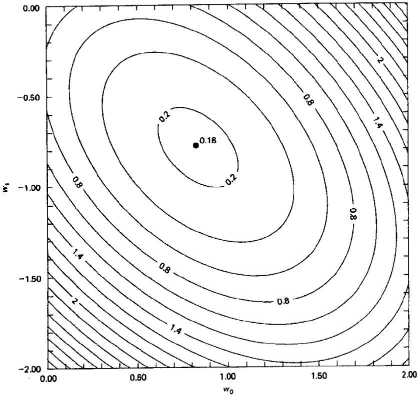


Figure 5.7 Contour plots of the error-performance surface depicted in Fig. 5.6.

Error Contours

Aside (Matrix Differentiation, Real Case):

Let $\mathbf{w} \in \mathbb{R}^{(d+1)}$ and let $f : \mathbb{R}^{(d+1)} \to \mathbb{R}$. The derivative of f (called gradient of f) with respect to $\mathbf{w}$ is:

$$\nabla_{\mathbf{w}}(f) = \frac{\partial f}{\partial \mathbf{w}} = \begin{bmatrix} \nabla_0(f) \\ \nabla_1(f) \\ \vdots \\ \nabla_d(f) \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial w_0} \\ \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_d} \end{bmatrix}$$

Thus,

$$\nabla_k(f) = \frac{\partial f}{\partial w_k}, \qquad k = 0, 1, \cdots, d$$

Example

Now suppose $f = \mathbf{c}^\top \mathbf{w}$. Find $\nabla_{\mathbf{w}}(f)$

In this case,

$$f = \mathbf{c}^\top \mathbf{w} = \sum_{k=0}^{d} w_k c_k$$

and

$$\nabla_k(f) = \frac{\partial f}{\partial w_k} = c_k, \qquad k = 0, 1, \cdots, d$$

Result: For $f = \mathbf{c}^\top \mathbf{w}$

$$\nabla_{\mathbf{w}}(f) = \begin{bmatrix} \nabla_0(f) \\ \nabla_1(f) \\ \vdots \\ \nabla_d(f) \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{bmatrix} = \mathbf{c}$$

Same for $f = \mathbf{w}^\top \mathbf{c}$.

Example

Lastly, suppose $f = \mathbf{w}^\top \mathbf{Q}\mathbf{w}$. Where $\mathbf{Q} \in \mathbb{R}^{(d+1)\times(d+1)}$ and $\mathbf{w} \in \mathbb{R}^{d+1}$. Find $\nabla_{\mathbf{w}}(f)$

In this case, using the product rule:

$$\begin{aligned}\nabla_{\mathbf{w}} f &= \frac{\partial \mathbf{w}^\top (\mathbf{Q}\bar{\mathbf{w}})}{\partial \mathbf{w}} + \frac{\partial (\bar{\mathbf{w}}^\top \mathbf{Q})\mathbf{w}}{\partial \mathbf{w}} \\ &= \frac{\partial \mathbf{w}^\top \mathbf{u}_1}{\partial \mathbf{w}} + \frac{\partial \mathbf{u}_2^\top \mathbf{w}}{\partial \mathbf{w}}\end{aligned}$$

Using previous result, $\frac{\partial \mathbf{c}^\top \mathbf{w}}{\partial \mathbf{w}} = \frac{\partial \mathbf{w}^\top \mathbf{c}}{\partial \mathbf{w}} = \mathbf{c}$,

$$\begin{aligned}\nabla_{\mathbf{w}} f &= \mathbf{u}_1 + \mathbf{u}_2, \\ &= \mathbf{Q}\mathbf{w} + \mathbf{Q}^\top \mathbf{w} = (\mathbf{Q} + \mathbf{Q}^\top)\mathbf{w}, \qquad \text{if } \mathbf{Q} \text{ symmetric, } \mathbf{Q}^\top = \mathbf{Q} \\ &= 2\mathbf{Q}\mathbf{w}\end{aligned}$$

Returning to the MSE performance criteria

$$E_{in}(\hat{\mathbf{w}}) = \left[\frac{1}{N}(\hat{\mathbf{w}}^\top \mathbf{X}^\top \mathbf{X}\hat{\mathbf{w}} - 2\hat{\mathbf{w}}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y})\right]$$

Differentiating with respect to $\hat{\mathbf{w}}$ and setting equal to zero, we obtain,

$$\begin{aligned}\bigtriangledown E_{in}(\hat{\mathbf{w}}) &= \frac{1}{N}(2\mathbf{X}^\top \mathbf{X}\hat{\mathbf{w}} - 2\mathbf{X}^\top \mathbf{y} + 0) \\ &= \frac{2}{N}\mathbf{X}^\top \mathbf{X}\hat{\mathbf{w}} - \frac{2}{N}\mathbf{X}^\top \mathbf{y} = 0\end{aligned}$$

$$\begin{aligned}\Longrightarrow \mathbf{X}^\top \mathbf{X}\hat{\mathbf{w}} &= \mathbf{X}^\top \mathbf{y} \\ \hat{\mathbf{w}} &= (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y} \\ &= \mathbf{X}^\dagger \mathbf{y},\end{aligned}$$

where $\mathbf{X}^\dagger = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top$ is the Moore-Penrose pseudo-inverse of $\mathbf{X}$.

Summarizing

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}}_{\mathbf{y}:\text{ credit line (\$)}} = \underbrace{\begin{bmatrix} 1 & \mathrm{x}_{11} & \mathrm{x}_{12} & \cdots & \mathrm{x}_{1d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \mathrm{x}_{N1} & \mathrm{x}_{N2} & \cdots & \mathrm{x}_{Nd} \end{bmatrix}}_{\mathbf{X}:\text{ age, gender, anual salary...}} \cdot \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}}_{\mathbf{w}:\text{ parameters}} + \underbrace{\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix}}_{\boldsymbol{\epsilon}:\text{randomness}}$$

Estimate w by solving linear system of equations:

$$\underbrace{\mathbf{y}}_{\text{known}} = \underbrace{\mathbf{X}}_{\text{known}} \underbrace{\hat{\mathbf{w}}}_{\text{solve}}$$

Solution:

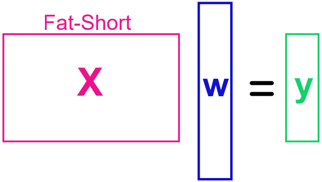
$$\begin{aligned} \hat{\mathbf{w}} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \\ &= \mathbf{X}^\dagger \mathbf{y}, \end{aligned}$$

where $\mathbf{X}^\dagger = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ is the Moore-Penrose pseudo-inverse of $\mathbf{X}$.

$$\mathbf{X}\mathbf{w} = \mathbf{y} \qquad \text{where} \qquad \mathbf{X} \in \mathbb{R}^{N\times(d+1)}$$

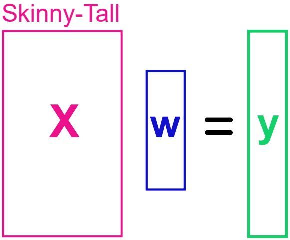
What happens when $\mathbf{X}$ is not square and invertible?

1. **Underdetermined Case** $(N < d+1)$:



- In general, infinite solutions exist.
- Not enough measurements of $\mathbf{y}$ to find a unique solution.
- There are fewer equations than unknowns (degrees of freedom).

2. **Overdetermined Case** ($N > d + 1$):



- In general, no solution exist.
- There are more equations (constraints) than unknowns (degrees of freedom).
- $\mathbf{y}$ cannot be obtained as a linear combination of the vectors in the column space of $\mathbf{X}$ i.e. $col(\mathbf{X})$.

Reminder: $col(\mathbf{X})$: **span** (set of all possible linear combinations) of the column vectors in $\mathbf{X}$.

Exceptions exist for both cases:

- A **solution exist** if $\mathbf{y}$ is on the column space of $\mathbf{X}$ i.e. $col(\mathbf{X})$.
- There is **no solution** if $\mathbf{y}$ is on the space orthogonal complement of $col(\mathbf{X})$ (everything that is not in $col(\mathbf{X})$).
- **Infinite solutions** if $\mathbf{w}$ is a solution and the null space of $\mathbf{X}$ is not empty i.e. $dim(ker(\mathbf{X})) \neq 0$.

Where the null space of $\mathbf{X}$ is all the vectors $\mathbf{w}_{null}$ that solve:

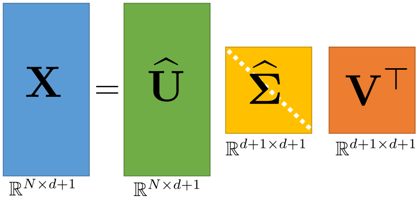
$$\mathbf{X}\mathbf{w}_{null} = \mathbf{0}$$

If and $\mathbf{X}\mathbf{w} = \mathbf{y}$:

$$\mathbf{X}(\mathbf{w} + \beta\mathbf{w}_{null}) = \mathbf{y}$$

Moore-Penrose Pseudo-inverse with SVD (Optional)

SVD allows us to "invert" $\mathbf{X}$. Given $\mathbf{X} = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\mathbf{V}^\top$ and the linear model:



$$\begin{aligned}\mathbf{X}\mathbf{w} &= \mathbf{y}\\ \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\mathbf{V}^\top\mathbf{w} &= \mathbf{y}\end{aligned}$$

Multiplying both sides by $\widehat{\mathbf{U}}^\top$:

$$\begin{aligned}\widehat{\mathbf{U}}^\top\widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\mathbf{V}^\top\mathbf{w} &= \widehat{\mathbf{U}}^\top\mathbf{y} \qquad \text{where} \quad \widehat{\mathbf{U}}^\top\widehat{\mathbf{U}} = \mathbf{I}\\ \widehat{\mathbf{\Sigma}}\mathbf{V}^\top\mathbf{w} &= \widehat{\mathbf{U}}^\top\mathbf{y}\end{aligned}$$

Multiplying both sides by $\widehat{\mathbf{\Sigma}}^{-1}$:

$$\begin{aligned}\widehat{\mathbf{\Sigma}}^{-1}\widehat{\mathbf{\Sigma}}\mathbf{V}^\top\mathbf{w} &= \widehat{\mathbf{\Sigma}}^{-1}\widehat{\mathbf{U}}^\top\mathbf{y} \qquad \text{where} \quad \widehat{\mathbf{\Sigma}}^{-1}\widehat{\mathbf{\Sigma}} = \mathbf{I}\\ \mathbf{V}^\top\mathbf{w} &= \widehat{\mathbf{\Sigma}}^{-1}\widehat{\mathbf{U}}^\top\mathbf{y}\end{aligned}$$

Solving with SVD (Optional)

$$\mathbf{V}^\top \mathbf{w} = \widehat{\mathbf{\Sigma}}^{-1}\widehat{\mathbf{U}}^\top \mathbf{y}$$

Multiplying both sides by $\mathbf{V}$:

$$\begin{aligned} \mathbf{V}\mathbf{V}^\top \mathbf{w} &= \mathbf{V}\widehat{\mathbf{\Sigma}}^{-1}\widehat{\mathbf{U}}^\top \mathbf{y} \qquad \text{where} \quad \mathbf{V}\mathbf{V}^\top = \mathbf{I} \\ \mathbf{w} &= \mathbf{V}\widehat{\mathbf{\Sigma}}^{-1}\widehat{\mathbf{U}}^\top \mathbf{y} \end{aligned}$$

Then

$$\begin{aligned} \mathbf{w} &= \mathbf{V}\widehat{\mathbf{\Sigma}}^{-1}\widehat{\mathbf{U}}^\top \mathbf{y} \\ \mathbf{w} &= \mathbf{X}^\dagger \mathbf{y} \end{aligned}$$

where $\mathbf{X}^\dagger = \mathbf{V}\widehat{\mathbf{\Sigma}}^{-1}\widehat{\mathbf{U}}$ is the Moore-Penrose pseudo-inverse of $\mathbf{X}$.

SVD solutions (Optional)

1. Underdetermined case:

$$\mathbf{w} = \mathbf{V}\widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\mathbf{U}}^{\top}\mathbf{y}$$

Is equivalent to:

$$\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}} \parallel \mathbf{w} \parallel_2 \quad \text{s.t.} \;\; \mathbf{X}\mathbf{w} = \mathbf{y}$$

2. Overdetermined case:

$$\mathbf{w} = \mathbf{V}\widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\mathbf{U}}^{\top}\mathbf{y}$$

Is equivalent to:

$$\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}} \parallel \mathbf{X}\mathbf{w} - \mathbf{y} \parallel_2 \qquad \text{Least Square Solution}$$

A real data set

16x16 pixels gray-scale images of digits from the US Postal Service Zip Code Database. The goal is to recognize the digit in each image.

This is not a trivial task (even for a human). A typical human error E_{out} is about 2.5% due to common confusions between $\{4, 9\}$ and $\{2, 7\}$.

Machine Learning tries to achieve or beat this error.

Input Representation

Since the images are 16×16 pixels:

- 'raw' input
  $\mathbf{x}_r = (x_0, x_1, x_2, \cdots, x_{256})$
- Linear model:
  $(w_0, w_1, w_2, \cdots, w_{256})$

It has too many many parameters.
A better input representation makes it simpler.

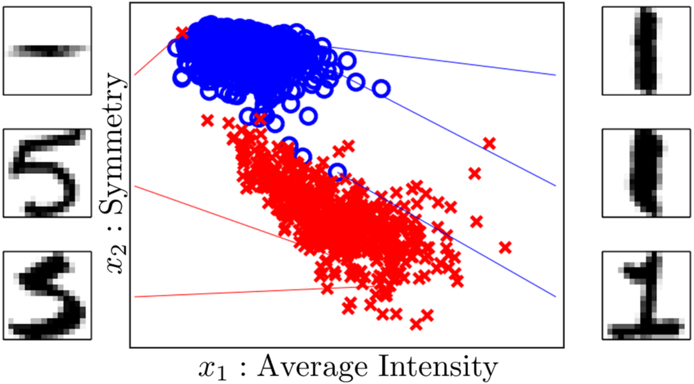
Features: Extract useful information, e.g.,

- Average intensity and symmetry
  $\mathbf{x} = (x_0, x_1, x_2)$
- Linear model: (w_0, w_1, w_2)

The descriptors must be representative of the data.

Illustration of Features

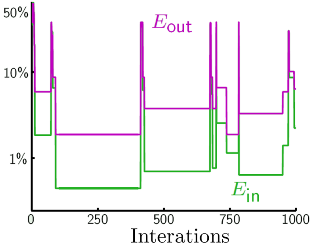
$$\mathbf{x} = (x_0, x_1, x_2) \quad x_0 = 1$$



It's almost linearly separable. However, it is impossible to have them all right.

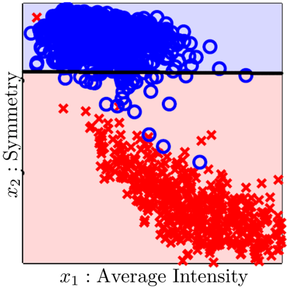
What Perceptron Learning Algorithm does?

Evolution of in-sample error E_{in} and out-of-sample error E_{out} as a function of iterations of PLA



- Assume we know E_{out} .
- E_{in} tracks E_{out}. PLA generalizes well!

- It would never converge (data not linearly separable).
- **Stopping criteria:** Max. number of iterations.



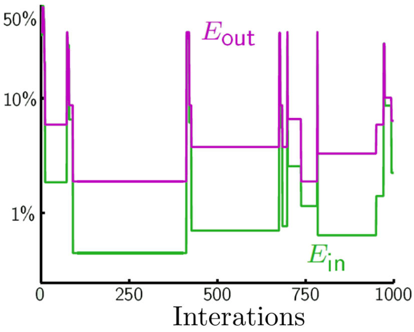
Final perceptron boundary

We can do better...

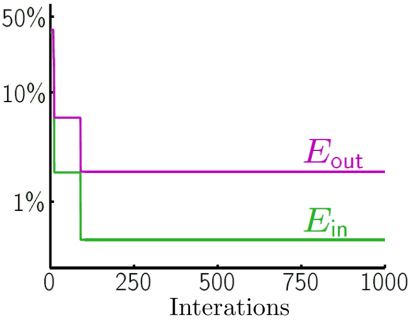
The 'pocket' algorithm

Keeps 'in its pocket' the best weight vector encountered up to the current iteration t in PLA.

PLA

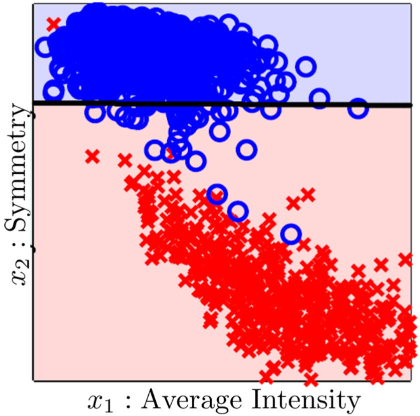


Pocket

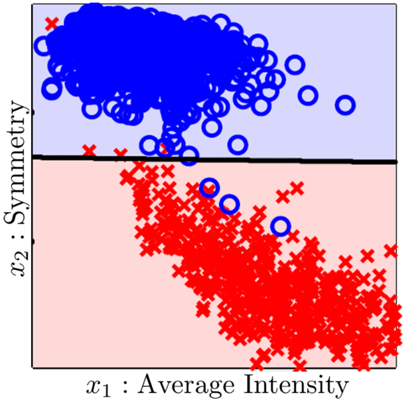


Classification boundary - PLA versus Poket

PLA



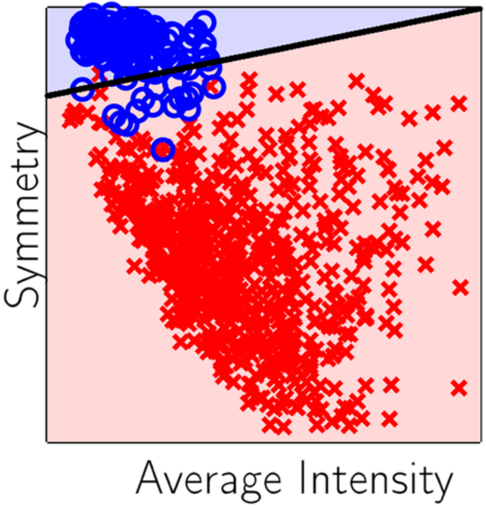
Pocket



Linear Regression for Classification

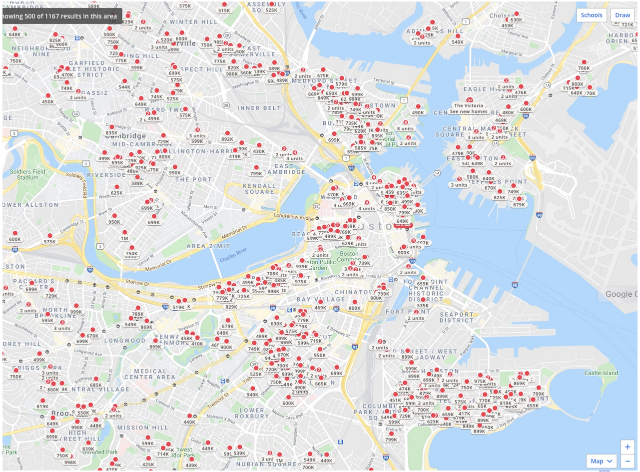
- Linear regression learns a real-valued function $y = f(\mathbf{x}) \in \mathbb{R}$
- Binary-valued functions are also real-valued! $\pm 1 \in \mathbb{R}$
- Use linear regression to get $\hat{\mathbf{w}}$ where $\hat{\mathbf{w}}^\top \mathbf{x}_n \approx y_n = \pm 1$
- In this case, $\text{sign}(\hat{\mathbf{w}}^\top \mathbf{x}_n)$ is likely to agree with y_n
- Good initial weights for classification

Linear regression boundary



Example: Boston Housing Market

- Predict $\mathbf{y}$: Median value of home in thousands.
- $\hat{\mathbf{y}} = \hat{\mathbf{w}}^\top \mathbf{x}$
- $\mathbf{x}$: 13 attributes correlated with house price.



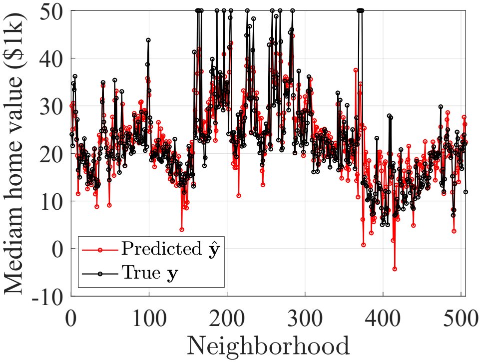
Boston Housing Dataset (1970)

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. The following describes the dataset columns:

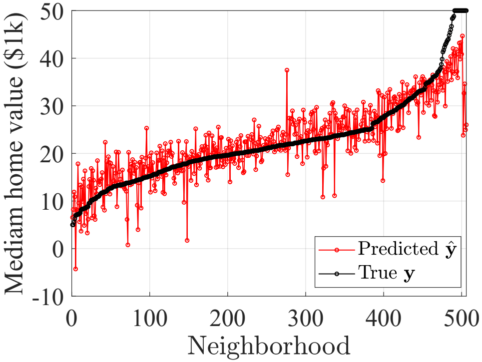
- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's

Example:

Predicted house value ($\hat{\mathbf{y}}$) and the true house value ($\mathbf{y}$):

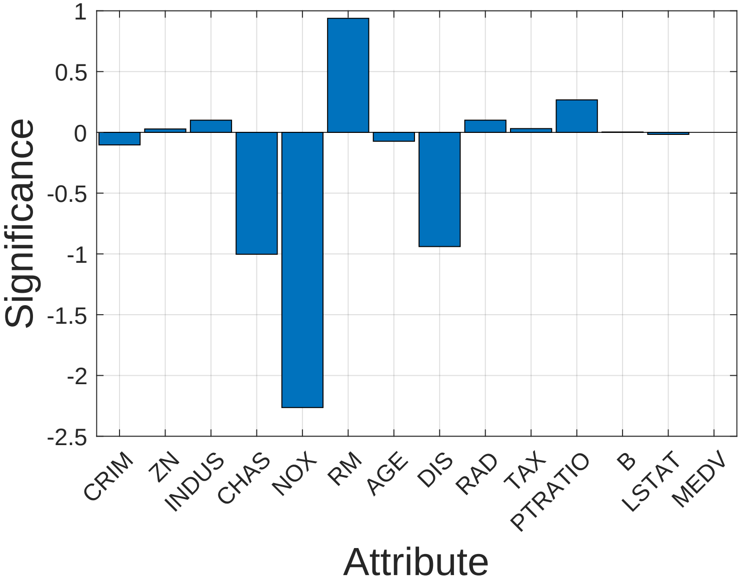


Unsorted data



Sorted Data

The significance of Each Variable



Conclusion so far

- Linear regression aims to find linear relationship between an interested variable $\mathbf{y}$, e.g., credit line and regressors, e.g., age, gender, and etc.
- Expression of E_{in} for linear regression
- Close form solution of $\hat{\mathbf{w}}$ by minimizing E_{in}: $\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$
- Linear regression can solve classification tasks by passing through the sign function, sign$(\mathbf{w}^\top \mathbf{x})$
- Application: The Boston housing price example

Next: an engineering way to solve linear regression: Gradient descent.

Definition (Steepest Descent (SD))

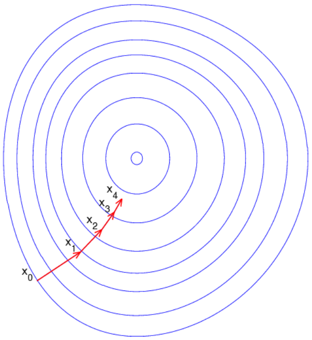
Steepest descent, also known as gradient descent is an iterative technique for finding the local minimum of a function.

Approach: Given an arbitrary starting point, the current location (value) is moved in steps proportional to the negatives of the gradient at the current point.

- SD is an old, deterministic method, that is the basis for stochastic gradient based methods
- SD is a feedback approach to finding local minimum of an error performance surface
- The error surface must be known *a priori*
- In the MSE case, SD converges converges to the optimal solution without inverting a matrix

Example

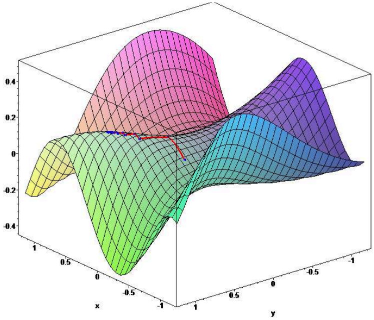
Consider a well structured cost function with a single minimum. The optimization proceeds as follows:



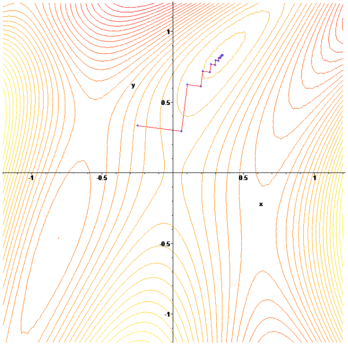
Contour plot showing that evolution of the optimization

Example

Consider a gradient ascent example in which there are multiple minima/maxima



Surface plot showing the multiple minima and maxima



Contour plot illustrating that the final result depends on starting value

To derive the approach, consider:

$$\begin{aligned} E_{in} &= \frac{1}{N}||\mathbf{X}\mathrm{w}-\mathbf{y}||_2^2 \\ &= \frac{1}{N}(\mathbf{y}^\top\mathbf{y}-\mathbf{y}^\top\mathbf{X}\mathrm{w}-\mathrm{w}^\top\mathbf{X}^\top\mathbf{y}+\mathrm{w}^\top\mathbf{X}^\top\mathbf{X}\mathrm{w} \\ &= \sigma_y^2-\mathbf{p}^\top\mathrm{w}-\mathrm{w}^\top\mathbf{p}+\mathrm{w}^\top\mathbf{R}\mathrm{w} \end{aligned}$$

where

$\sigma_y^2=\frac{1}{N}\mathbf{y}^\top\mathbf{y}$ variance estimate of desired signal

$\mathbf{p}=\frac{1}{N}\mathbf{X}^\top\mathbf{y}$ – cross-correlation estimate between $\mathbf{x}$ and y

$\mathbf{R}=\frac{1}{N}\mathbf{X}^\top\mathbf{X}$ – correlation matrix estimate of $\mathbf{x}$

When $\mathbf{w}$ is set to the (optimal) Least Squares solution $\mathbf{w}_0$:

$$\mathbf{w} = \mathbf{w}_0 = \mathbf{R}^{-1}\mathbf{p}$$

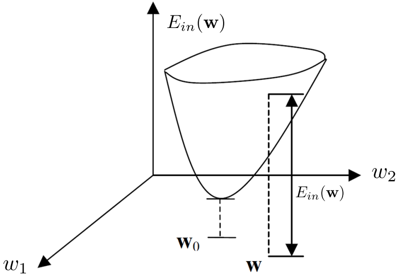
, then

$$\begin{aligned} E_{in} = E_{in_{\min}} &= \sigma_y^2 - 2\mathbf{p}^\top \mathbf{w} + \mathbf{p}^\top (\mathbf{R}^{-1})^\top \mathbf{R}\mathbf{R}^{-1}\mathbf{p} \\ &= \sigma_y^2 - 2\mathbf{p}^\top \mathbf{w} + \mathbf{p}^\top \mathbf{R}^{-1}\mathbf{p} \\ &= \sigma_y^2 - \mathbf{p}^H \mathbf{w}_0 \end{aligned}$$

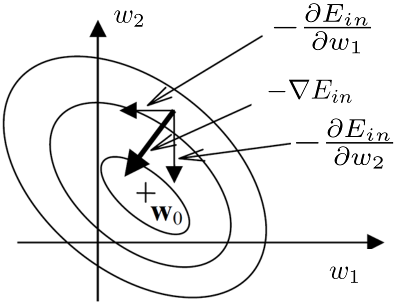
- Use the method of steepest descent to iteratively find $\mathbf{w}_0$.
- The optimal result is achieved since the cost function is a second order polynomial with a single unique minimum

Example

The MSE is a bowl–shaped surface, which is a function of the 2-D space weight vector $\mathbf{w}(n)$



Surface Plot



Contour Plot

Imagine dropping a marble at any point on the bowl-shaped surface.
The ball will reach the minimum point by going through the path of steepest descent.

Observation: Set the direction of filter update as: $-\nabla E_{in}(n)$

Resulting Update:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{1}{2}\mu[-\nabla E_{in}(n)]$$

or, since $\nabla E_{in}(n) = -\frac{2}{N}\mathbf{X}^\top\mathbf{y} + \frac{2}{N}\mathbf{X}^\top\mathbf{X}\mathbf{w} = -2\mathbf{p} + 2\mathbf{R}\mathbf{w}(n)$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu[\mathbf{p} - \mathbf{R}\mathbf{w}(n)] \quad n = 0, 1, 2, \cdots$$

where $\mathbf{w}(0) = \mathbf{0}$ (or other appropriate value) and μ is the step size

Observation: SD uses feedback, which makes it possible for the system to be unstable

- Bounds on the step size guaranteeing stability can be determined with respect to the eigenvalues of $\mathbf{R}$ (Widrow, 1970)

Convergence Analysis

Define the error vector for the tap weights as

$$\mathbf{c}(n) = \mathbf{w}(n) - \mathbf{w}_0$$

Then using $\mathbf{p} = \mathbf{R}\mathbf{w}_0$ in the update,

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu[\mathbf{p} - \mathbf{R}\mathbf{w}(n)] \\ &= \mathbf{w}(n) + \mu[\mathbf{R}\mathbf{w}_0 - \mathbf{R}\mathbf{w}(n)] \\ &= \mathbf{w}(n) - \mu\mathbf{R}\mathbf{c}(n) \end{aligned}$$

and subtracting $\mathbf{w}_0$ from both sides

$$\begin{aligned} \mathbf{w}(n+1) - \mathbf{w}_0 &= \mathbf{w}(n) - \mathbf{w}_0 - \mu\mathbf{R}\mathbf{c}(n) \\ \Rightarrow \mathbf{c}(n+1) &= \mathbf{c}(n) - \mu\mathbf{R}\mathbf{c}(n) \\ &= [\mathbf{I} - \mu\mathbf{R}]\mathbf{c}(n) \end{aligned}$$

Using the Unitary Similarity Transform

$$\mathbf{R} = \mathbf{Q}\mathbf{\Omega}\mathbf{Q}^H$$

we have

$$\begin{aligned}
\mathbf{c}(n+1) &= [\mathbf{I} - \mu\mathbf{R}]\mathbf{c}(n) \\
&= [\mathbf{I} - \mu\mathbf{Q}\mathbf{\Omega}\mathbf{Q}^H]\mathbf{c}(n) \\
\Rightarrow \mathbf{Q}^H\mathbf{c}(n+1) &= [\mathbf{Q}^H - \mu\mathbf{Q}^H\mathbf{Q}\mathbf{\Omega}\mathbf{Q}^H]\mathbf{c}(n) \\
&= [\mathbf{I} - \mu\mathbf{\Omega}]\mathbf{Q}^H\mathbf{c}(n) \qquad (*)
\end{aligned}$$

Define the transformed coefficients as

$$\begin{aligned}
\mathbf{v}(n) &= \mathbf{Q}^H\mathbf{c}(n) \\
&= \mathbf{Q}^H(\mathbf{w}(n) - \mathbf{w}_0)
\end{aligned}$$

Then $(*)$ becomes

$$\mathbf{v}(n+1) = [\mathbf{I} - \mu\mathbf{\Omega}]\mathbf{v}(n)$$

Consider the initial condition of $\mathbf{v}(n)$

$$\begin{aligned}\mathbf{v}(0) &= \mathbf{Q}^H(\mathbf{w}(0) - \mathbf{w}_0) \\ &= -\mathbf{Q}^H\mathbf{w}_0 \qquad\qquad [\text{if } \mathbf{w}(0) = \mathbf{0}]\end{aligned}$$

Consider the k^{th} term (mode) in

$$\mathbf{v}(n+1) = [\mathbf{I} - \mu\mathbf{\Omega}]\mathbf{v}(n)$$

- Note $[\mathbf{I} - \mu\mathbf{\Omega}]$ is diagonal
- Thus all modes are independently updated
- The update for the k^{th} term can be written as

$$v_k(n+1) = (1 - \mu\lambda_k)v_k(n) \quad k = 1, 2, \cdots, M$$

or using recursion

$$v_k(n) = (1 - \mu\lambda_k)^n v_k(0)$$

Observation: Conversion to the optimal solution requires

$$\begin{aligned}
\lim_{n\to\infty} \mathbf{w}(n) &= \mathbf{w}_0 \\
\Rightarrow \lim_{n\to\infty} \mathbf{c}(n) &= \lim_{n\to\infty} \mathbf{w}(n) - \mathbf{w}_0 = \mathbf{0} \\
\Rightarrow \lim_{n\to\infty} \mathbf{v}(n) &= \lim_{n\to\infty} \mathbf{Q}^H \mathbf{c}(n) = \mathbf{0} \\
\Rightarrow \lim_{n\to\infty} v_k(n) &= 0 \quad k = 1, 2, \cdots, M \qquad (*)
\end{aligned}$$

Result: According to the recursion

$$v_k(n) = (1 - \mu\lambda_k)^n v_k(0)$$

the limit in $(*)$ holds if and only if

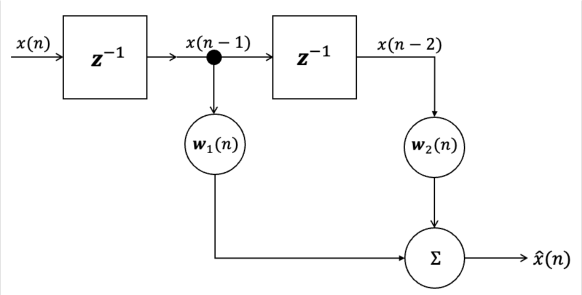
$$|1 - \mu\lambda_k| < 1 \quad \text{for all } k$$

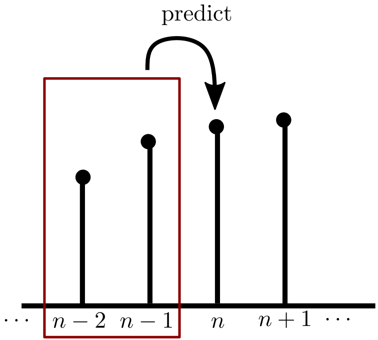
Thus since the eigenvalues are nonnegative, $0 < \mu\lambda_{\max} < 2$, or

$$0 < \mu < \frac{2}{\lambda_{\max}}$$

Example: Predictor

Consider a two–tap predictor for real–valued input





Example: Predictor

Use $\mathbf{x}(n-1) = \begin{bmatrix} x(n-1) \\ x(n-2) \end{bmatrix}$ to predict $x(n)$ such that

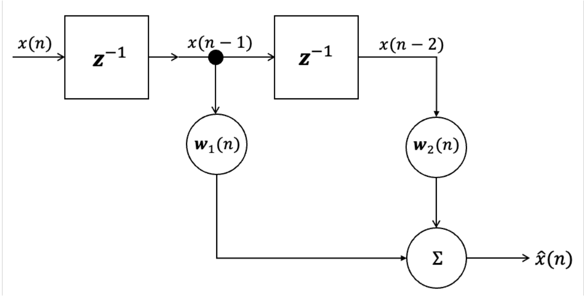
$$y(n) = \hat{x}(n) = \mathbf{x}(n-1)^\top \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} = \mathbf{x}(n-1)^\top \mathbf{w}(n)$$

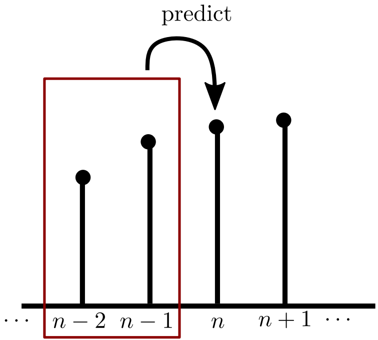
$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

$$\underbrace{\begin{bmatrix} x(2) & x(1) \\ x(3) & x(2) \\ \vdots & \vdots \\ x(n-1) & x(n-2) \end{bmatrix}}_{\mathbf{X} \in \mathbb{R}^{N\times 2}} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \underbrace{\begin{bmatrix} x(3) \\ x(4) \\ \vdots \\ x(n) \end{bmatrix}}_{\mathbf{y} \in \mathbb{R}^{N}}$$

$$\mathbf{R} = \frac{1}{N}\mathbf{X}^T\mathbf{X} \qquad \mathbf{p} = \frac{1}{N}\mathbf{X}^T\mathbf{y}$$

Example: Predictor

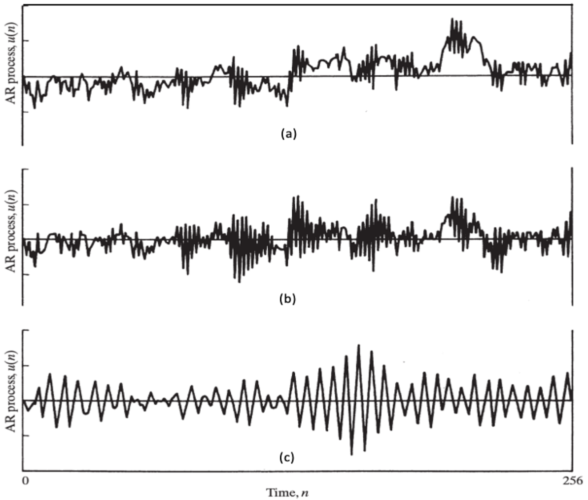




Analyzed the effects of the following cases:

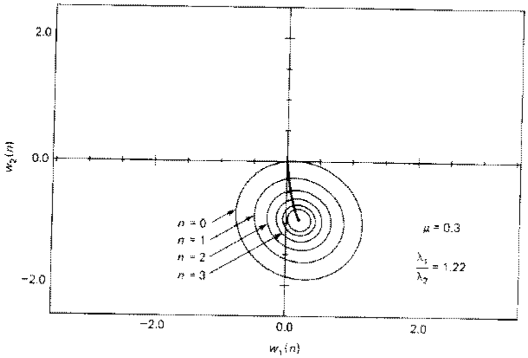
- Varying the eigenvalue spread $\chi(\mathbf{R}) = \frac{\lambda_{\max}}{\lambda_{\min}}$ while keeping μ fixed
- Varying μ and keeping the eigenvalue spread $\chi(\mathbf{R})$ fixed

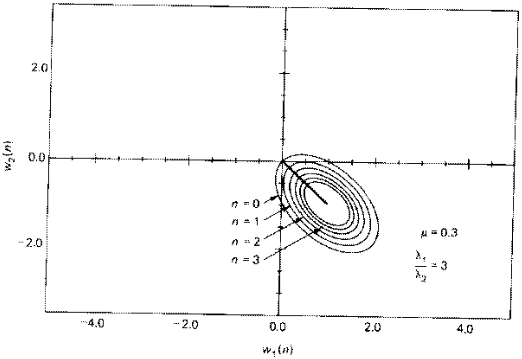
AR model of order 2



Outputs of AR model of order 2 with different parameters w_1 and w_2.

SD loci plots (with shown $E_{in}(n)$ contours) as a function of $[w_1(n), w_2(n)]$ for step-size $\mu = 0.3$

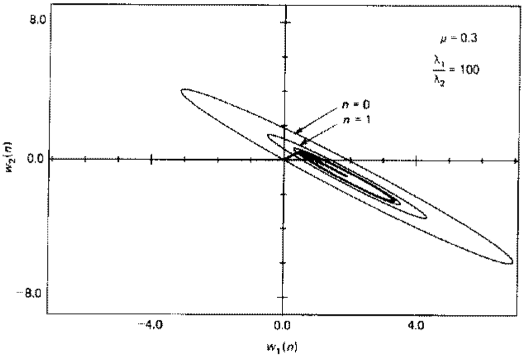




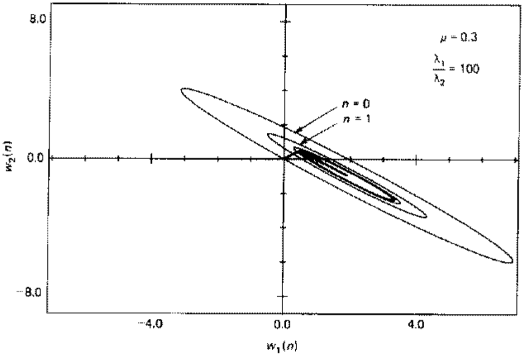
- Eigenvalue spread: $\chi(\mathbf{R}) = 1.22$
- Small eigenvalue spread $\Rightarrow$ modes converge at a similar rate

- Eigenvalue spread: $\chi(\mathbf{R}) = 3$
- Moderate eigenvalue spread $\Rightarrow$ modes converge at moderately similar rates

SD loci plots (with shown $E_{in}(n)$ contours) as a function of $[w_1(n), w_2(n)]$ for step-size $\mu = 0.3$

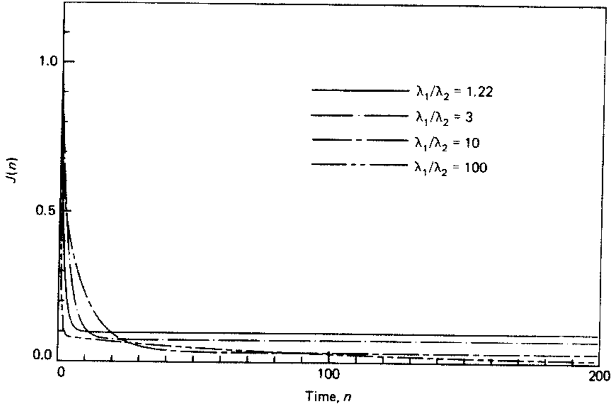


- Eigenvalue spread: $\chi(\mathbf{R}) = 10$
- Large eigenvalue spread $\Rightarrow$ modes converge at different rates

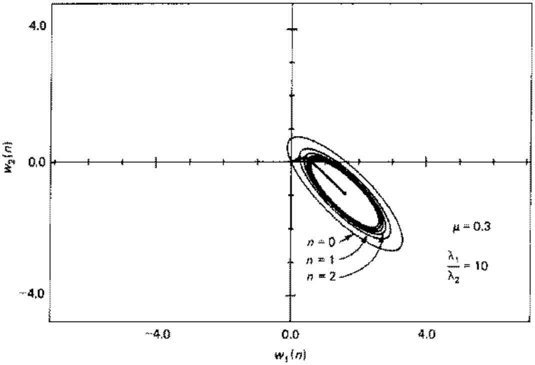


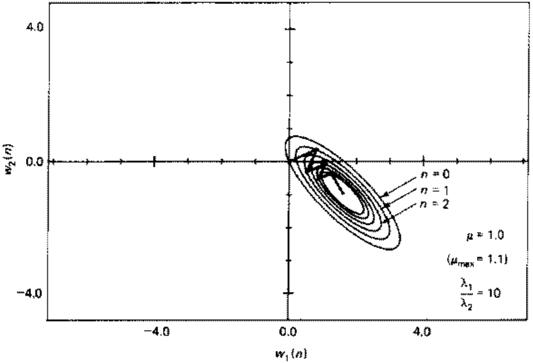
- Eigenvalue spread: $\chi(\mathbf{R}) = 100$
- Very large eigenvalue spread $\Rightarrow$ modes converge at very different rates
- Principle direction convergence is fastest

Learning curves of steepest-descent algorithm with step-size parameter $\mu = 0.3$ and varying eigenvalue spread.



SD loci plots (with shown $E_{in}(n)$ contours) as a function of $[w_1(n), w_2(n)]$ with $\chi(\mathbf{R}) = 10$ and varying step–sizes





- Step–sizes: $\mu = 0.3$
- This is over–damped $\Rightarrow$ slow convergence

- Step–sizes: $\mu = 1$
- This is under–damped $\Rightarrow$ fast (erratic) convergence

Stochastic Gradient Descent (SGD)

Instead of considering the full *batch*, for each iteration, pick one training data point $(\mathbf{X}_n, y_n)$ at random and apply GD update to $\mathrm{e}(h(\mathbf{x}_n, y_n))$

The weight update of SGD is:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla \mathrm{e}_n(\mathbf{w}(t))$$

For $\mathrm{e}(h(\mathbf{x}_n, y_n)) = (\mathbf{w}^\top \mathbf{x}_n - y_n)^2$ i.e. for the mean squared error:

$$\begin{aligned} \nabla \mathrm{e}_n(\mathbf{w}) &= 2\mathbf{x}_n(\mathbf{w}^\top \mathbf{x}_n - y_n) \\ &= 2\mathbf{x}_n(\mathbf{x}_n^\top \mathbf{w} - y_n) \\ &= 2(\mathbf{x}_n\mathbf{x}_n^\top \mathbf{w} - \mathbf{x}_n y_n) \\ &= 2(\widehat{\mathbf{R}}\mathbf{w} - \hat{\mathbf{p}}) \end{aligned} \qquad \mathbf{w}^\top \mathbf{x}_n = \mathbf{x}_n^\top \mathbf{w}$$

where $\widehat{\mathbf{R}} = \mathbf{x}_n\mathbf{x}_n^\top$ is the instantaneous estimate of $\mathbf{R}$ and $\hat{\mathbf{p}} = \mathbf{x}_n y_n$ is the instantaneous estimate of $\mathbf{p}$.

Stochastic Gradient Descent (SGD)

Since n is picked at random, the expected weight change is:

$$\begin{aligned} \mathbb{E}_n\left[-\nabla \mathrm{e}(h(\mathbf{x}_n, y_n))\right] &= \frac{1}{N}\sum_{n=1}^{N} -\nabla \mathrm{e}(h(\mathbf{x}_n, y_n)) \\ &= -\nabla E_{in} \end{aligned}$$

Same as the *batch* gradient descent.

Result: On 'average' the minimization proceeds in the right direction (remember LMS).

Stochastic Gradient Descent (SGD)

Instead of considering the full *batch*, for each iteration, pick one training data point $(\mathbf{x}_n, y_n)$ at random and apply GD update to $\mathrm{e}(h(\mathbf{x}_n, y_n))$

The weight update of SGD is:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla \mathrm{e}_n(\mathbf{w}(t))$$

Since n is picked at random, the expected weight change is:

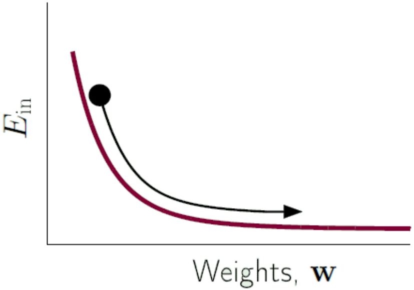
$$\begin{aligned} \mathbb{E}_n\left[-\nabla \mathrm{e}(h(\mathbf{x}_n, y_n))\right] &= \frac{1}{N}\sum_{n=1}^{N} -\nabla \mathrm{e}(h(\mathbf{x}_n, y_n)) \\ &= -\nabla E_{in} \end{aligned}$$

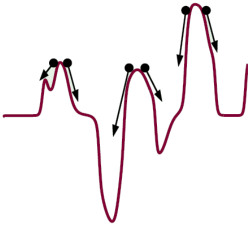
Same as the *batch* gradient descent.

Result: On 'average' the minimization proceeds in the right direction.

Benefits of SGD

1. Cheaper computation (by a factor of N compare to GD)
2. Randomization
3. Simple





Randomization helps to avoid local minima and flat regions.

Rule of thumb:

Start with:

$$\eta = 0.1 \quad \text{works!}$$

SGD is successful in practice!