CS 1571 Introduction to AI
Lecture 24

Linear regression

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Supervised learning

Data: $D = \{D_1, D_2, .., D_n\}$ **a set of *n* examples**

$D_i = < \mathbf{x}_i, y_i >$

$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \cdots x_{i,d})$ is an input vector of size d

y_i is the desired output (given by a teacher)

Objective: learn the mapping $f : X \rightarrow Y$

s.t. $y_i \approx f(\mathbf{x}_i)$ for all $i = 1,.., n$

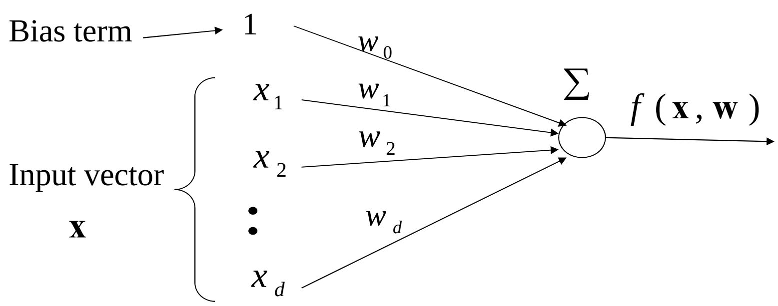
- **Regression:** Y is **continuous**
  Example: earnings, product orders $\rightarrow$ company stock price
- **Classification:** Y is **discrete**
  Example: handwritten digit in binary form $\rightarrow$ digit label

Linear regression

- **Function** $f : X \rightarrow Y$ is a linear combination of input components

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j$$

$w_0, w_1, \ldots w_k$ - **parameters (weights)**



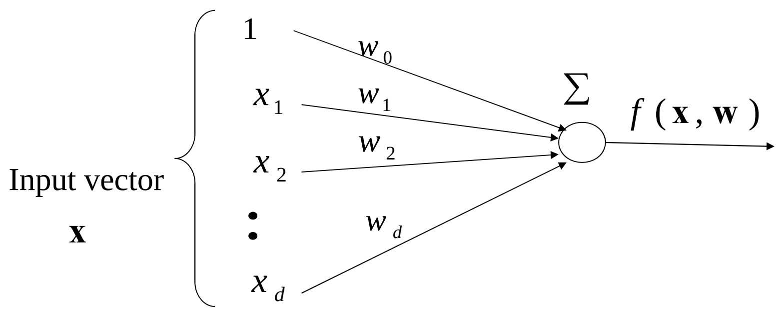
Linear regression

- **Shorter (vector) definition of the model**
  - Include bias constant in the input vector

$\mathbf{x} = (1, x_1, x_2, \cdots x_d)$

$$f(\mathbf{x}) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d = \mathbf{w}^T \mathbf{x}$$

$w_0, w_1, \ldots w_k$ - **parameters (weights)**



Linear regression. Error.

- **Data:** $D_i = < \mathbf{x}_i, y_i >$
- **Function:** $\mathbf{x}_i \to f(\mathbf{x}_i)$
- We would like to have $y_i \approx f(\mathbf{x}_i)$ for all $i = 1,.., n$

- **Error function** measures how much our predictions deviate from the desired answers

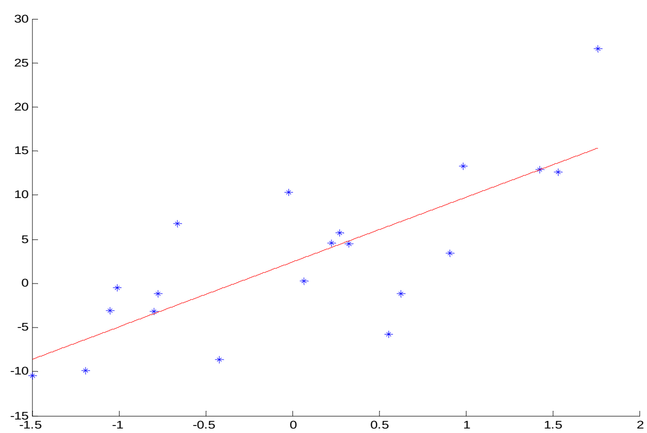
Mean-squared error $J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2$

- **Learning:**

  We want to find the weights minimizing the error !

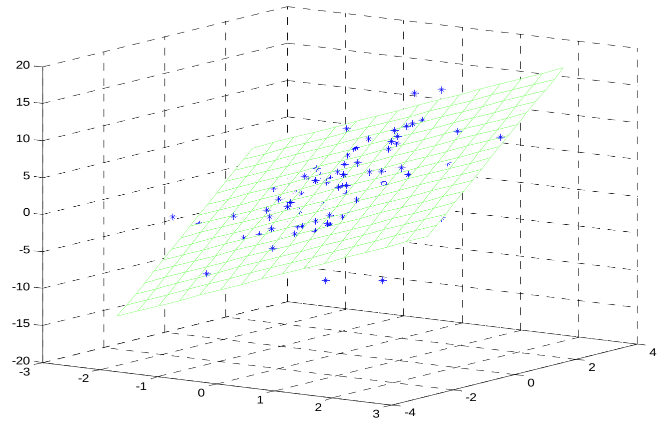
Linear regression. Example

- 1 dimensional input $\mathbf{x} = (x_1)$



Linear regression. Example.

- 2 dimensional input $\mathbf{x} = (x_1, x_2)$



Linear regression. Optimization.

- We want the **weights minimizing the error**

$$J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2 = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0$$

- **Vector of derivatives:**

$$\text{grad}_{\mathbf{w}}(J_n(\mathbf{w})) = \nabla_{\mathbf{w}}(J_n(\mathbf{w})) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i = \bar{\mathbf{0}}$$

Linear regression. Optimization.

- For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0

$$J_n = \frac{1}{n}\sum_{i=1,..n}(y_i - f(\mathbf{x}_i))^2 = \frac{1}{n}\sum_{i=1,..n}(y_i - [w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots w_k x^{(k)}])^2$$

- $\text{grad}_{\mathbf{w}}(J_n(\mathbf{w})) = \overline{\mathbf{0}}$ defines a set of equations in $\mathbf{w}$

$$\frac{\partial}{\partial w_0} J_n(w) = -\frac{2}{n}\sum_{i=1}^{n}[y_i - (w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots w_k x^{(k)})] = 0$$

$$\frac{\partial}{\partial w_1} J_n(w) = -\frac{2}{n}\sum_{i=1}^{n}[y_i - (w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots w_k x^{(k)})]x^{(1)} = 0$$

$$\dots$$

$$\frac{\partial}{\partial w_j} J_n(w) = -\frac{2}{n}\sum_{i=1}^{n}[y_i - (w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots w_k x^{(k)})]x^{(j)} = 0$$

$$\dots$$

Solving linear regression

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 x_{i,0} - w_1 x_{i,1} - \dots - w_d x_{i,d})x_{i,j} = 0$$

By rearranging the terms we get a **system of linear equations** with *d+1* unknowns

$$\mathbf{Aw} = \mathbf{b}$$

$$w_0 \sum_{i=1}^{n} x_{i,0}1 + w_1 \sum_{i=1}^{n} x_{i,1}1 + \dots + w_j \sum_{i=1}^{n} x_{i,j}1 + \dots + w_d \sum_{i=1}^{n} x_{i,d}1 = \sum_{i=1}^{n} y_i 1$$

$$w_0 \sum_{i=1}^{n} x_{i,0}x_{i,1} + w_1 \sum_{i=1}^{n} x_{i,1}x_{i,1} + \dots + w_j \sum_{i=1}^{n} x_{i,j}x_{i,1} + \dots + w_d \sum_{i=1}^{n} x_{i,d}x_{i,1} = \sum_{i=1}^{n} y_i x_{i,1}$$

$$\dots$$

$$w_0 \sum_{i=1}^{n} x_{i,0}x_{i,j} + w_1 \sum_{i=1}^{n} x_{i,1}x_{i,j} + \dots + w_j \sum_{i=1}^{n} x_{i,j}x_{i,j} + \dots + w_d \sum_{i=1}^{n} x_{i,d}x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j}$$

$$\dots$$

Solving linear regression

- The optimal set of weights satisfies:

$$\nabla_{\mathbf{w}}(J_n(\mathbf{w})) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i = \overline{\mathbf{0}}$$

Leads to a **system of linear equations (SLE)** with *d+1* unknowns of the form

$$\mathbf{Aw} = \mathbf{b}$$

$$w_0\sum_{i=1}^{n} x_{i,0}x_{i,j} + w_1\sum_{i=1}^{n} x_{i,1}x_{i,j} + \ldots + w_j\sum_{i=1}^{n} x_{i,j}x_{i,j} + \ldots + w_d\sum_{i=1}^{n} x_{i,d}x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j}$$

Solutions to SLE:

- e.g. matrix inversion (if the matrix is singular)

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$$

Gradient descent solution

- There are other ways to solve the weight optimization problem in the linear regression model

$$J_n = Error(\mathbf{w}) = \frac{1}{n}\sum_{i=1,..n}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- A simple technique:
  - **Gradient descent**

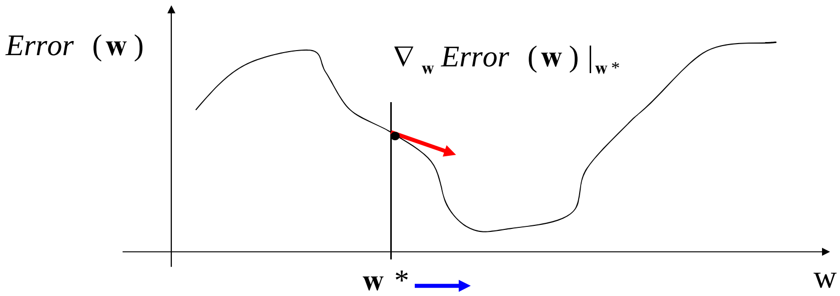
    Idea:
    - Adjust weights in the direction that improves the Error
    - The gradient tells us what is the right direction

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

$\alpha > 0$ - a learning rate (scales the gradient changes)

Gradient descent method

- Descend using the gradient information

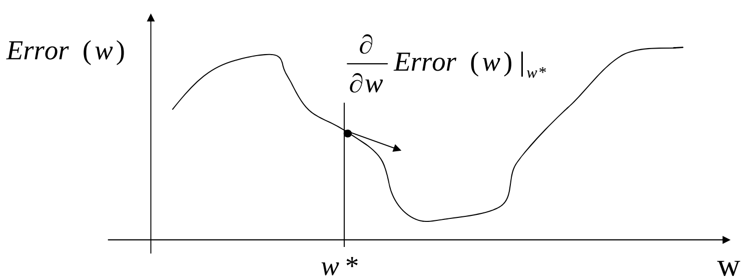


Direction of the descent

- Change the value of **w** according to the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

Gradient descent method



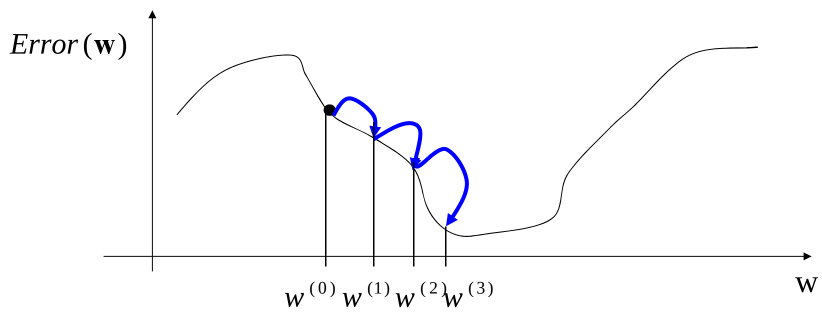
- New value of the parameter

$$w_j \leftarrow w_j^* - \alpha \frac{\partial}{\partial w_j} Error(w)|_{w^*} \qquad \text{For all j}$$

$\alpha > 0$ - a learning rate (scales the gradient changes)

Gradient descent method

- Iteratively converge to the optimum of the Error function



Online regression algorithm

- The error function defined for the whole dataset *D*

$$J_n = Error(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- Instead of the error for all data points we **use error for each examples** $D_i = < \mathbf{x}_i, y_i >$

$$J_{\text{online}} = Error_i(\mathbf{w}) = \frac{1}{2}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- **Change regression weights after every example according to the gradient:**

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} Error_i(\mathbf{w})$$

vector form: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$

$\alpha > 0$ - Learning rate that depends on the number of updates

Gradient for on-line learning

Linear model $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

On-line error $J_{online} = Error_i(\mathbf{w}) = \frac{1}{2}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2$

On-line algorithm: sequence of online updates

(i)-th update for the linear model: $D_i =< \mathbf{x}_i, y_i >$

Vector form:

$$\mathbf{w}^{(i)} \leftarrow \mathbf{w}^{(i-1)} - \alpha(i) \nabla_{\mathbf{w}} Error_i(\mathbf{w})|_{\mathbf{w}^{(i-1)}} = \mathbf{w}^{(i-1)} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}^{(i-1)}))\mathbf{x}_i$$

j-th weight:

$$w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha(i) \frac{\partial Error_i(\mathbf{w})}{\partial w_j}|_{\mathbf{w}^{(i-1)}} = w_j^{(i-1)} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}^{(i-1)}))x_{i,j}$$

Annealed learning rate: $\alpha(i) \approx \frac{1}{i}$

- Gradually rescales changes in weights

Adaptive models

Linear model $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

On-line error $J_{online} = Error_i(\mathbf{w}) = \frac{1}{2}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2$

On-line algorithm:

- Sequence of online updates (one example at the time)
- Useful for continuous data streams

Adaptive models:

- the underlying model is not stationary and can change over time
  - Example: seasonal changes
- On-line algorithm can be made adaptive by keeping the learning at some constant value $\alpha(i) \approx c$

Online regression algorithm

Online-linear-regression (D, *number of iterations*)

Initialize weights $\mathbf{w} = (w_0, w_1, w_2 \ldots w_d)$

for i=1:1: *number of iterations*

do **select** a data point $D_i = (\mathbf{x}_i, y_i)$ from D

set $\alpha = 1/i$

update weight vector

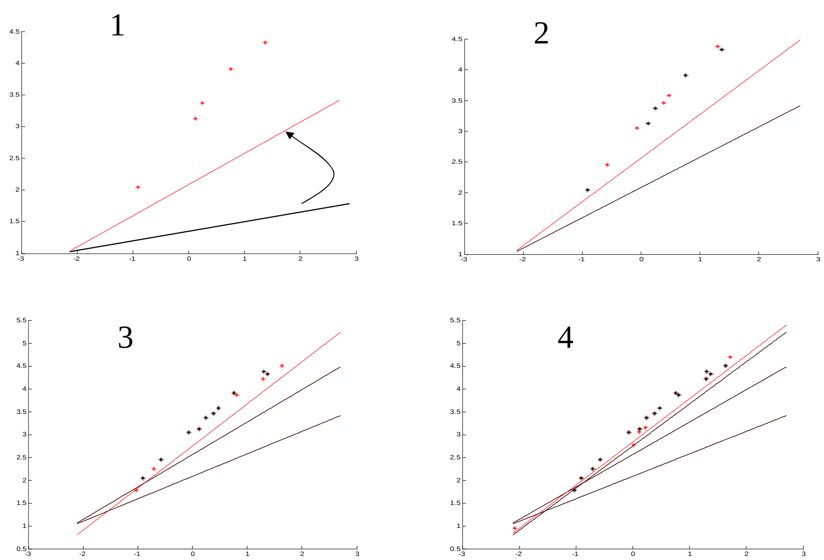
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(y_i - f(\mathbf{x}_i, \mathbf{w}))\mathbf{x}_i$$

end for

return weights $\mathbf{w}$

- **Advantages:** very easy to implement, continuous data streams

On-line learning. Example

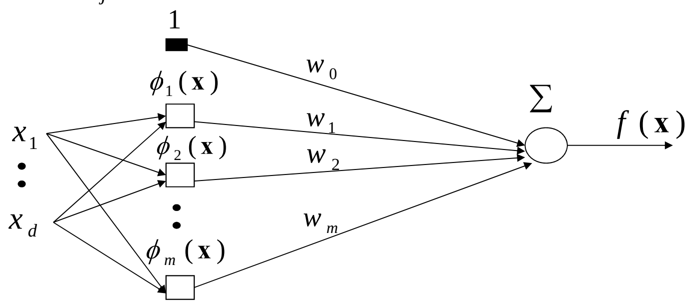


Extensions of simple linear model

Replace inputs to linear units with **feature (basis) functions** to model **nonlinearities**

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

$\phi_j(\mathbf{x})$ - an arbitrary function of $\mathbf{x}$



The same techniques as before to learn the weights

Additive linear models

- **Models linear in the parameters we want to fit**

$$f(\mathbf{x}) = w_0 + \sum_{k=1}^{m} w_k \phi_k(\mathbf{x})$$

$w_0, w_1 ... w_m$ - parameters

$\phi_1(\mathbf{x}), \phi_2(\mathbf{x}) ... \phi_m(\mathbf{x})$ - **feature or basis functions**

- **Basis functions examples:**
  - a higher order polynomial, one-dimensional input $\mathbf{x} = (x_1)$

    $\phi_1(x) = x \quad \phi_2(x) = x^2 \quad \phi_3(x) = x^3$
  - Multidimensional quadratic $\mathbf{x} = (x_1, x_2)$

    $\phi_1(\mathbf{x}) = x_1 \quad \phi_2(\mathbf{x}) = x_1^2 \quad \phi_3(\mathbf{x}) = x_2 \quad \phi_4(\mathbf{x}) = x_2^2 \quad \phi_5(\mathbf{x}) = x_1 x_2$
  - Other types of basis functions

    $\phi_1(x) = \sin x \quad \phi_2(x) = \cos x$

Fitting additive linear models

- **Error function** $J_n(\mathbf{w}) = 1/n \sum_{i=1,..n} (y - f(\mathbf{x}_i))^2$

Assume: $\boldsymbol{\varphi}(\mathbf{x}_i) = (1, \phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), \dots, \phi_m(\mathbf{x}_i))$

$$\nabla_{\mathbf{w}} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i)) \boldsymbol{\varphi}(\mathbf{x}_i) = \overline{\mathbf{0}}$$

- Leads to a **system of *m* linear equations**

$$w_0 \sum_{i=1}^{n} 1 \phi_j(\mathbf{x}_i) + \ldots + w_j \sum_{i=1}^{n} \phi_j(\mathbf{x}_i)\phi_j(\mathbf{x}_i) + \ldots + w_m \sum_{i=1}^{n} \phi_m(\mathbf{x}_i)\phi_j(\mathbf{x}_i) = \sum_{i=1}^{n} y_i \phi_j(\mathbf{x}_i)$$

- Can be solved exactly like the linear case

Example. Regression with polynomials.

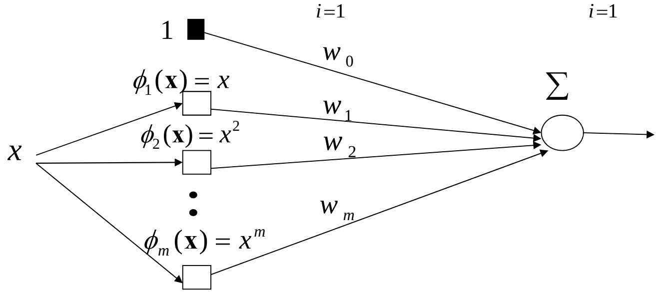
Regression with polynomials of degree m

- **Data points:** **pairs of** $< x, y >$
- **Feature functions:** **m feature functions**

$$\phi_i(x) = x^i \qquad i = 1, 2, \dots, m$$

- **Function to learn:**

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i$$



Learning with feature functions

Function to learn:

$$f(x,\mathbf{w}) = w_0 + \sum_{i=1}^{k} w_i \phi_i(x)$$

On line gradient update for the <*x*,*y*> pair

$$w_0 = w_0 + \alpha(y - f(\mathbf{x},\mathbf{w}))$$

$$\vdots$$

$$w_j = w_j + \alpha(y - f(\mathbf{x},\mathbf{w}))\phi_j(\mathbf{x})$$

Gradient updates are of the same form as in the linear regression models

Example. Regression with polynomials.

Example: Regression with polynomials of degree m

$$f(x,\mathbf{w}) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i$$

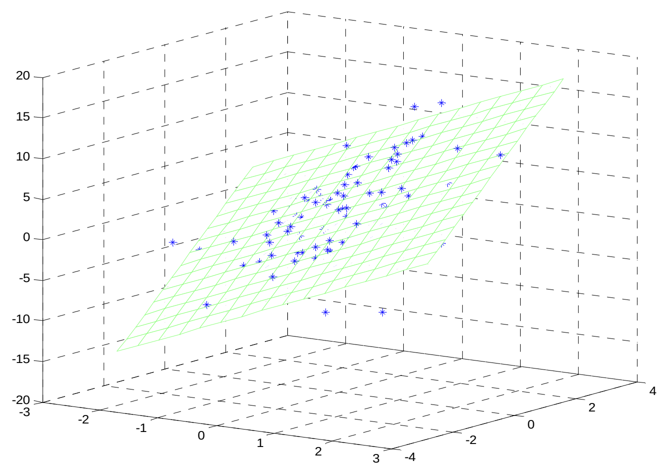
- **On line update** for <*x*,*y*> pair

$$w_0 = w_0 + \alpha(y - f(\mathbf{x},\mathbf{w}))$$

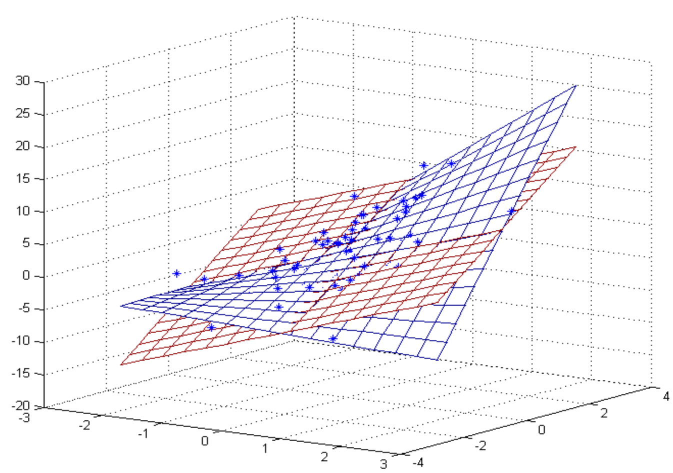
$$\vdots$$

$$w_j = w_j + \alpha(y - f(\mathbf{x},\mathbf{w}))x^j$$

Multidimensional additive model example



Multidimensional additive model example



Regularized linear regression

- If the number of parameters is large relative to the number of data points used to train the model, we face the threat of overfit (generalization error of the model goes up)
- The prediction accuracy can be often improved by setting some coefficients **w** to zero
  - Increases the bias, reduces the variance of estimates
- **Solutions:**
  - **Subset selection**
  - **Ridge regression**
  - **Lasso regression**
  - **Principal component regression**

- Next: **ridge and lasso regression**

Ridge regression

- Error function for the standard least squares estimates:

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- **We seek:** $\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$

- **Ridge regression:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|^2$$

- Where $\|\mathbf{w}\|^2 = \sum_{i=0}^{d} w_i^2$ and $\lambda \geq 0$

- What does the new error function do?

Ridge regression

- **Standard regression:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- **Ridge regression:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|^2$$

- $\|\mathbf{w}\|^2 = \sum_{i=0}^{d} w_i^2$ penalizes non-zero weights with the cost proportional to λ (a **shrinkage coefficient**)
- If an input attribute x_j has a small effect on improving the error function it is "shut down" by the penalty term
- Inclusion of a shrinkage penalty is often referred to as **regularization**

Lasso regression

- **Standard regression:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- **Lasso regression:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_1$$

- $\|\mathbf{w}\|_1 = \sum_{i=0}^{d} | w_i |$ penalizes non-zero weights with the cost proportional to λ
- Lasso regression is more aggressive than the ridge regression in zeroing the weights
- Lasso + ridge regularization combined:
  - **Elastic net regularization**