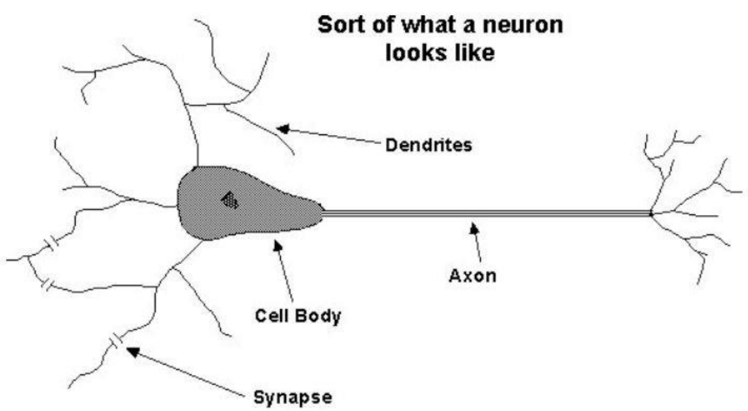
Creative Computing for Engineers

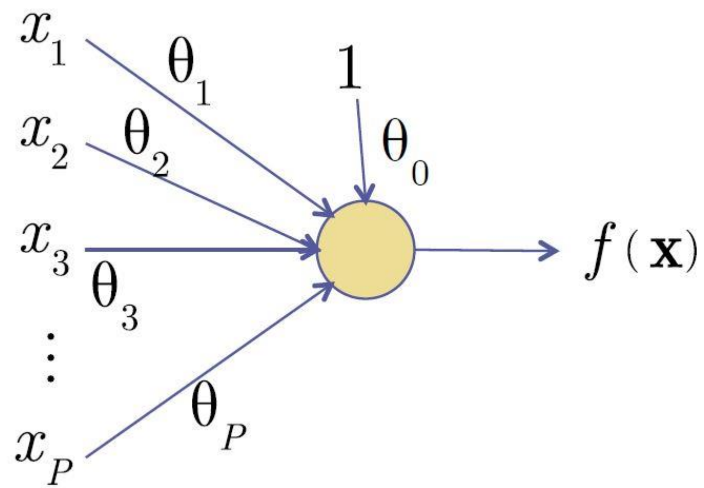
Lecture 11:
Perceptron as Linear Regression

Neural Network

Background

- Developing a computer processing system similar to human brain





- Background
  - 1943: Neural Network initiated by McCulloch and Pitts
  - 1958: Perceptron proposed by Rosenblatt
  - 1986: Multilayer perceptron proposed by Rumelhart, Hinton, and Williams

    … *(hard to train for large network, computing power for big data, etc.)*
  - Today: Advanced techniques (e.g., Deep Learning)

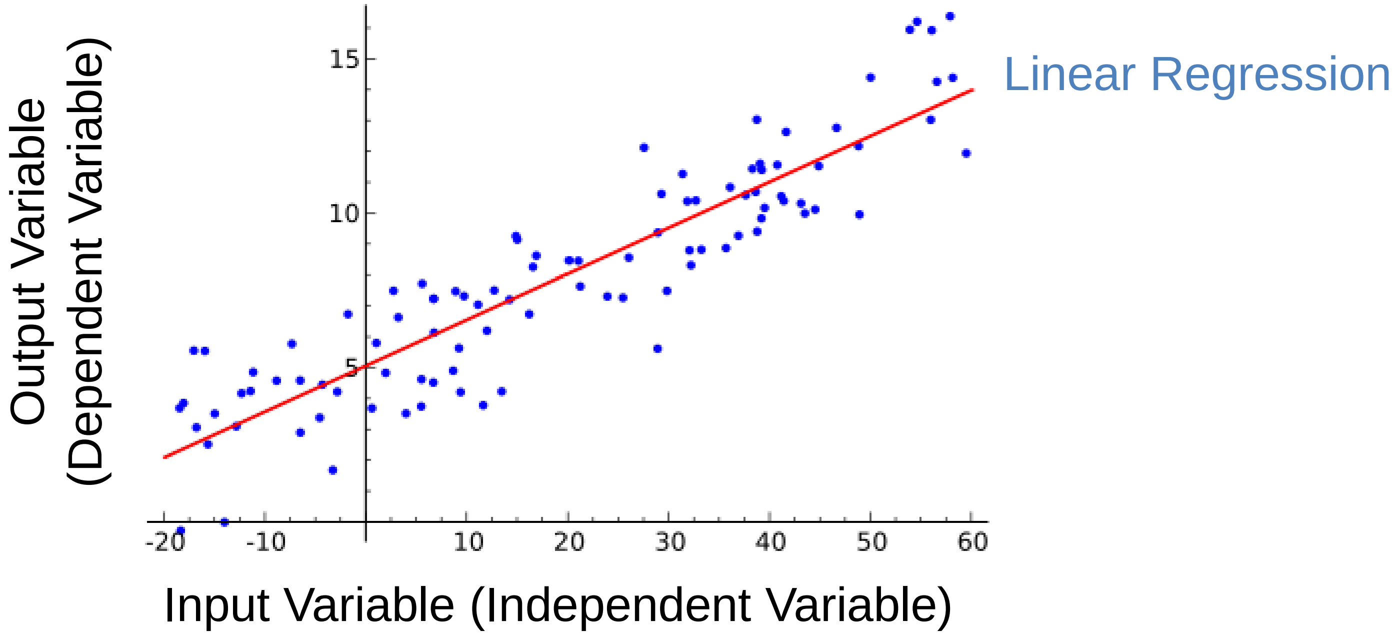
Source: Tony Jebara. Machine Learning Lecture Note, Columbia University

Linear Regression

What is Regression?

: Regression analysis is a statistical process to estimate the relationships between variables.

- Fit the data with the best hyper-plane which go through the points
- The output is continuous (for classification, the output is nominal).

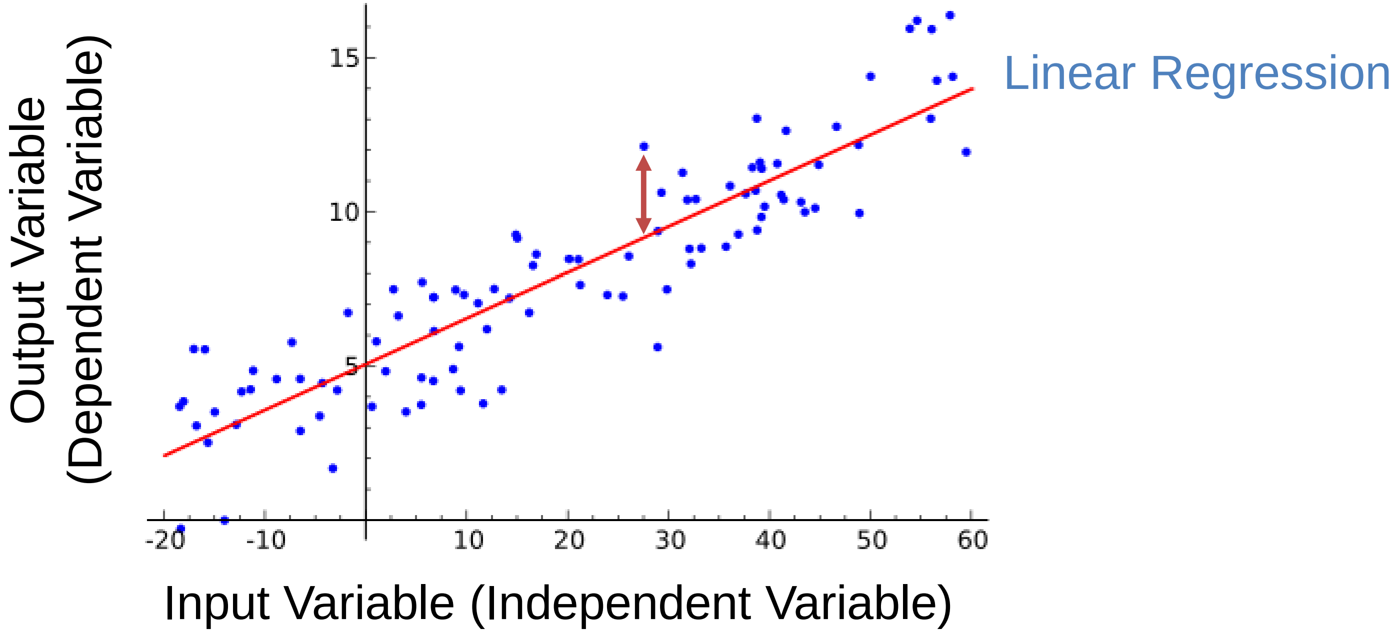


Source: Tony R. Martinez. Lecture Note: Regression, Brigham Young University.
Image Source: https://en.wikipedia.org/wiki/Regression_analysis#/media/File:Linear_regression.svg

Optimization for Linear Regression

What is Optimization?

- The optimization is used to minimize the errors between the estimated values and the data points.



Source: Tony R. Martinez. Lecture Note: Regression, Brigham Young University.
Image Source: https://en.wikipedia.org/wiki/Regression_analysis#/media/File:Linear_regression.svg

Optimization for Linear Regression

Linear Optimization

- A method to achieve the best outcome in mathematical model whose requirements are represented by linear relationships.

- Formulation of a linear programming problem
  - **Objective function**: Maximization or minimization (e.g., maximum profit or lowest cost)
  - **Constraints**: Available resources or requirements (e.g., available materials or targeted productions)

[Tang, S. L. (1999). Linear Optimization in Applications]

Optimization Example

- Example: Production optimization
  - A pipe manufacturing company produces two types of pipes, type I and type II. Determine the numbers of type I and II pipes produced to maximize the profit. The storage space, raw material requirement and production rate are given as below:

Resources	Type I	Type II	Availability
Storage space	5 m^2/pipe	3 m^2/pipe	750 m^2
Raw materials	6 kg/pipe	4 kg/pipe	800 kg/day
Production rate	30 pipes/hour	20 pipes/hour	8 hours/day
Profit	$10/pipe	$8/pipe	
Number of production	X pipes/day	Y pipes/day	

[Tang, S. L. (1999). Linear Optimization in Applications]

Optimization Example

- Example: Production optimization

  ➔ Problem formulation

  - Objective function
    - Total profit (Z) = 10 X + 8 Y

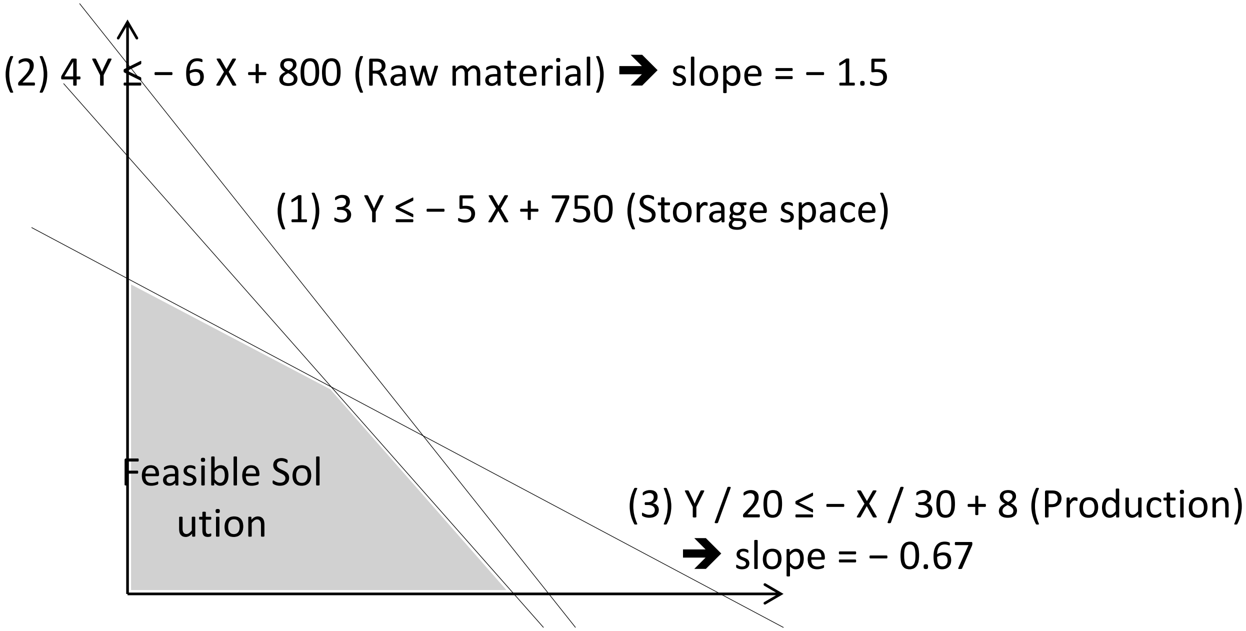
      ➔ Maximize total profit (Z)
  - Constraints
    - Storage space = $5X + 3Y \leq 750$ -- (1)
    - Raw material = $6X + 4Y \leq 800$ -- (2)
    - Production = $X / 30 + Y / 20 \leq 8$ -- (3)
    - Boundary conditions: $X \geq 0$ and $Y \geq 0$

[Tang, S. L. (1999). Linear Optimization in Applications]

Optimization Example

- Example: Production optimization

➔ Graphical Model

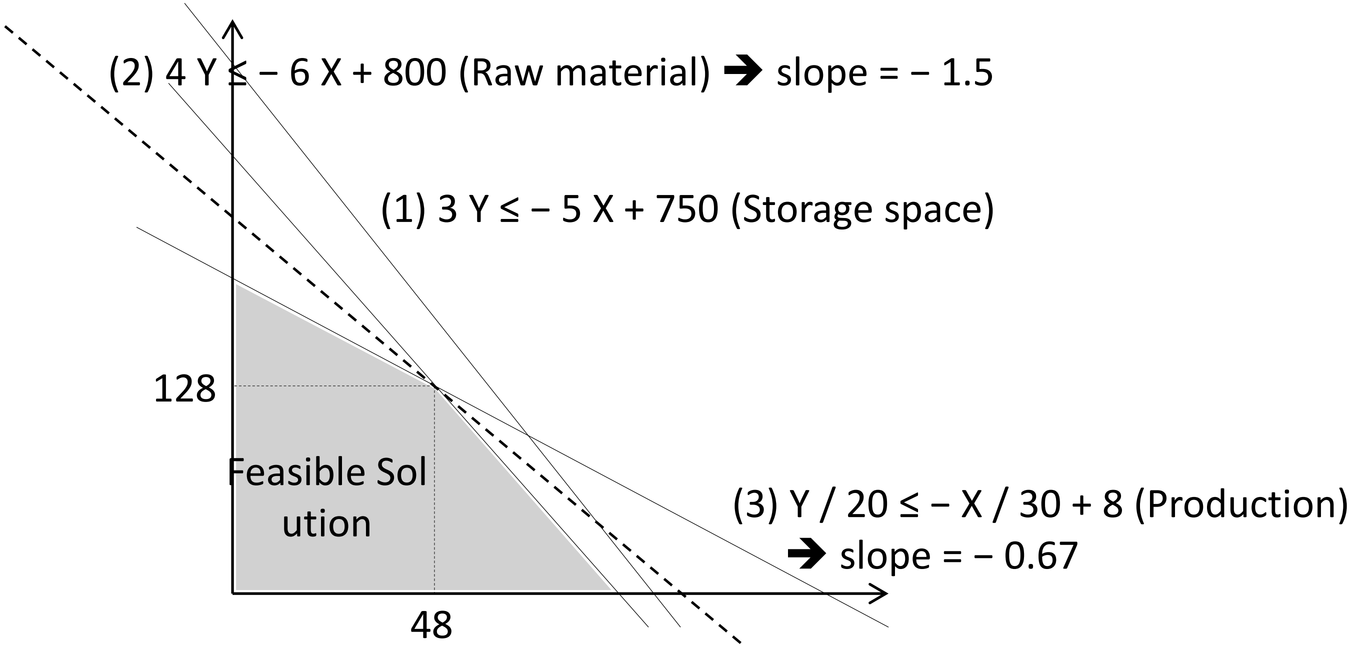


[Tang, S. L. (1999). Linear Optimization in Applications]

Optimization Example

- Example: Production optimization

➔ Graphical Model



Objective function:
8 Y = − 10 X + Z ➔ slope = − 1.25
Z = 1504

[Tang, S. L. (1999). Linear Optimization in Applications]

What is Regression?

Simple Linear Regression: Conceptual Approach

- Assume just 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 line (i.e. hyper-plane)
- Which line should we use?
  - Choose an objective function
  - For simple linear regression, we choose Sum Squared Error (SSE)

$$\sum (predicted_i - actual_i)^2 = \sum (residue_i)^2 \quad (i\text{: data point})$$

  - Thus, find the line which minimizes the sum of the squared residues (e.g. least squares)

Source: Tony R. Martinez. Lecture Note: Regression, Brighham Young University.

Linear Regression

What is Regression?

Simple Linear Regression: Conceptual Approach

- Which line should we use?
  - Choose an objective function
  - For simple linear regression, we choose Sum Squared Error (SSE)

    $\sum (predicted_i - actual_i)^2 = \sum (residue_i)^2$ (i: data point)

  - Thus, find the line which minimizes the sum of the squared residues (e.g. least squares)

[EXCEL Example]

Source: Tony R. Martinez. Lecture Note: Regression, Brighham Young University.

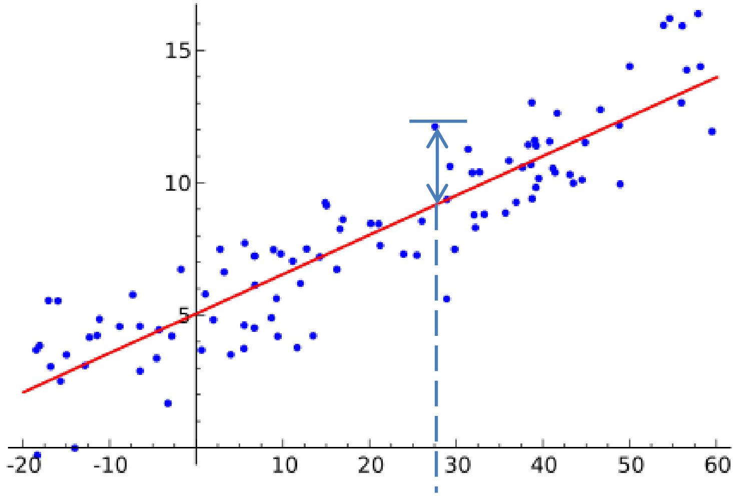
Linear Regression

What is Regression?

Simple Linear Regression: Parameters Calculation

- For the 2-D problem (line), there are coefficients for the bias and the independent variable (y-intercept and slope)

$$Y = \beta_0 + \beta_1 X$$



- To find the values for the coefficients which minimize the objective function, we take the partial derivates of the objective function (SSE) with respect to the coefficients. Set these to 0, and solve.

$$\beta_1 = \frac{\sum(\boldsymbol{x} - \overline{\boldsymbol{x}})(\boldsymbol{y} - \overline{\boldsymbol{y}})}{\sum(\boldsymbol{x} - \overline{\boldsymbol{x}})^2} \qquad \beta_0 = \frac{\sum y - \beta_1 \sum x}{n} = \frac{\sum y}{n} - \beta_1 \frac{\sum x}{n}$$

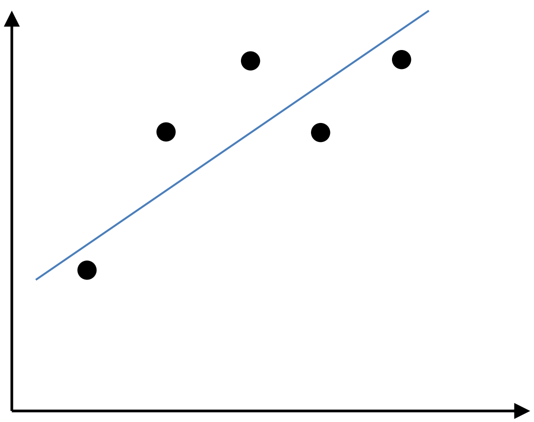
Source: Tony R. Martinez. Lecture Note: Regression, Brighham Young University.

Linear Regression

What is Regression?

Simple Linear Regression: Conceptual Approach

$$Y = \beta_0 + \beta_1 X$$



x	y	$x - \bar{x}$	$y - \bar{y}$	$(x - \bar{x})^2$	$(x - \bar{x})(y - \bar{y})$
1	2	-2	-2	4	4
2	4	-1	0	1	0
3	5	0	1	0	0
4	4	1	0	1	0
5	5	2	1	4	2
Mean 3	4		Sum	10	6

$$\beta_1 = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2} = \frac{6}{10} = 0.6$$

$$\beta_0 = \frac{\sum y - \beta_1 \sum x}{n} = \frac{\sum y}{n} - \beta_1 \frac{\sum x}{n}$$

$$= 4 - 0.6 \times 3 = 2.2$$

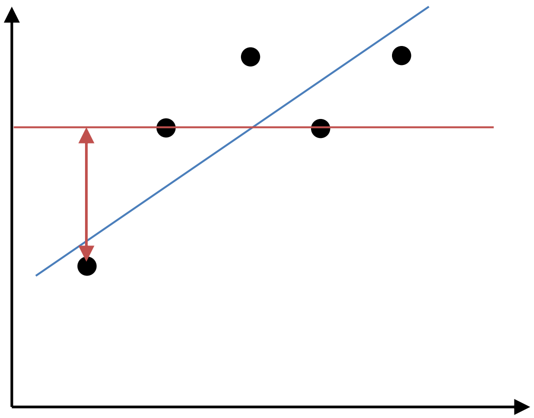
Source: https://www.youtube.com/watch?v=JvS2triCgOY

Linear Regression

What is Regression?

Simple Linear Regression: Conceptual Approach

$Y = 2.2 + 0.6X$



x	y	$y - \bar{y}$	$(y - \bar{y})^2$	$\hat{y}$	$\hat{y} - \bar{y}$	$(\hat{y} - \bar{y})^2$
1	2	-2	4	2.8	-1.2	1.44
2	4	0	0	3.4	-0.6	0.36
3	5	1	1	4	0	0
4	4	0	0	4.6	0.6	0.36
5	5	1	1	5.2	1.2	1.44
Mean	4	Sum	6			3.6

$$R^2 = \frac{(\hat{y} - \bar{y})^2}{(y - \bar{y})^2} = \frac{3.6}{6} = 0.6$$

Source: https://www.youtube.com/watch?v=w2FKXOa0HGA

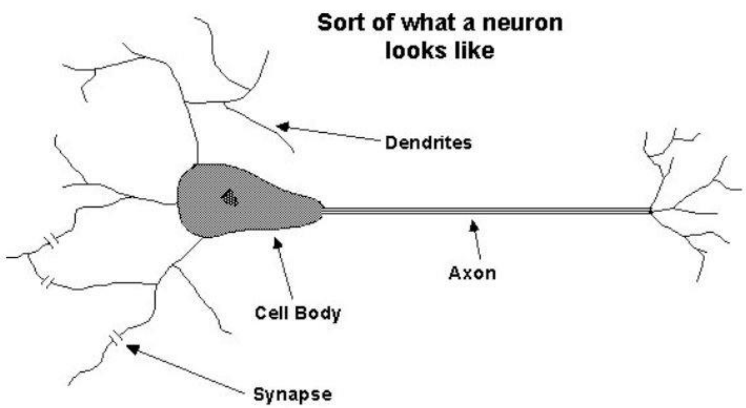
Neural Network

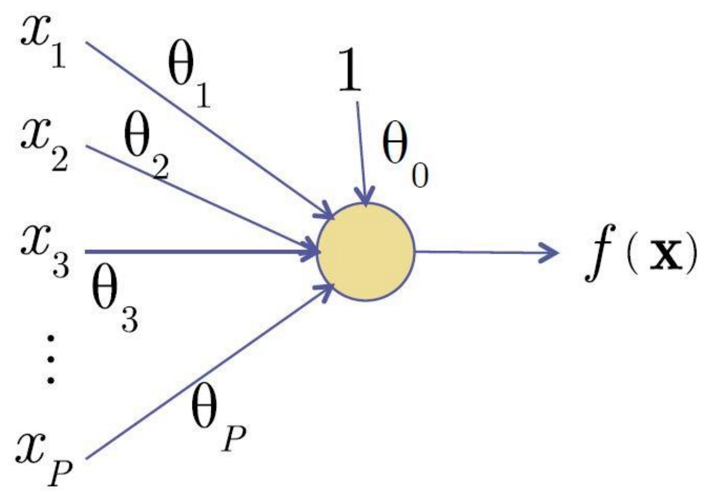
Neuron as Regression

- The *McCullouch-Pitts* Neuron is a graphical representation of linear regression

$$f(x;\ \theta) = \sum_{P=1}^{P} \theta_P x_P + \theta_0$$

  - Edges multiple signal by scalar weight (θ)
  - Nodes sum inputs here
  - Parameters: $\theta_1, \ldots, \theta_P$ = weight; θ_0 = bias





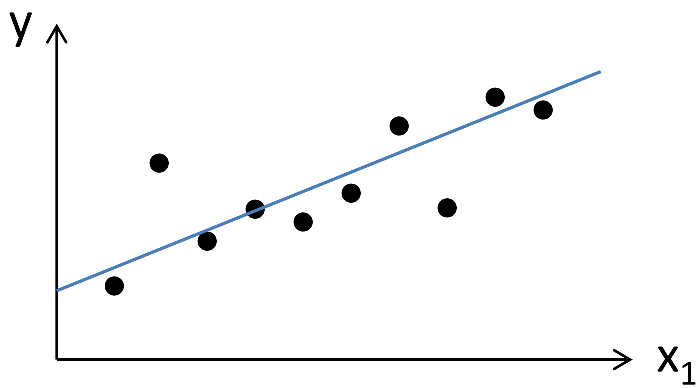
Source: Tony Jebara. Machine Learning Lecture Note, Columbia University

Perceptron

Perceptron for Linear Regression

: Single Layer Neural Network

- Simple Linear Regression



Notation

- Superscript: Index of the data point in the training dataset; k=kth training data point
- Subscript: Coordinate (dimension) of the data point; x_1^k = coordinate 1 of data point k.

- We have training data $X = \{x_1^k\} = \{x_1^1, x_1^2, \ldots, x_1^N\}$ (k=1, 2, …, N data points) with corresponding output $Y = \{y^k\} = \{y^1, y^2, \ldots, y^N\}$
- We want to find the parameters (w) that predict the output Y from the data X in a linear fashion:

$$y^k \approx w_0 + w_1 x_1^k$$

Source: Martial Hebert. Lecture Note: Artificial Intelligence, Neural Networks, Carnegie Mellon University.

Perceptron for Linear Regression

- Simple Linear Regression
  - It is convenient to define an additional "fake" attribute (feature/dimension) for the input data: $x_0 = 1$
  - We want to find the parameters (w) that predict the output Y from the data X in a linear fashion:

$$y^k \approx w_0 + w_1 x_1^k = w_0 x_0^k + w_1 x_1^k$$

  - **Vector** of attributes for each training data point:

$$\mathbf{x}^k = \{x_0^k, x_1^k, \ldots, x_M^k\} \qquad \text{(M=1, 2, ..., M dimensions)}$$

  - We seek a **vector** of parameters:

$$\mathbf{w} = \{w_0, w_1, \ldots, w_M\} \qquad \text{(M=1, 2, ..., M dimensions)}$$

  - Such that we have a linear relation between prediction Y and attributes X:

$$y^k \approx w_0 x_0^k + w_1 x_1^k + \ldots + w_M x_M^k = \sum_{i=0}^{M} w_i x_i^k = \mathbf{w} (\mathbf{x}^k)^T$$

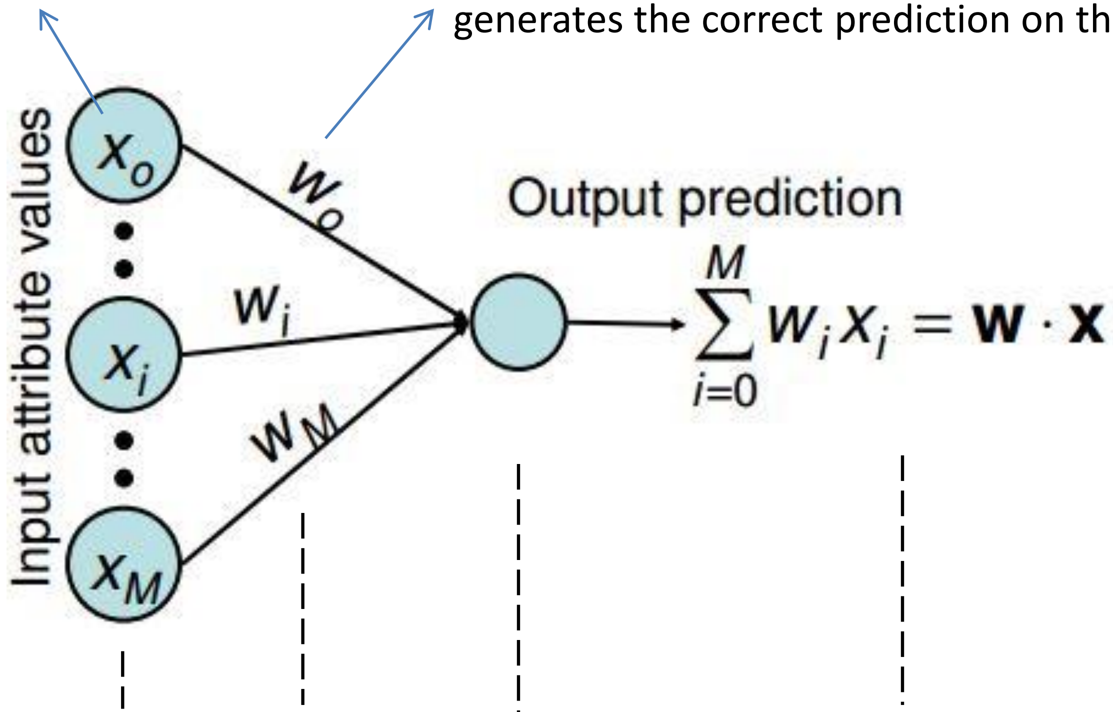
Source: Martial Hebert. Lecture Note: Artificial Intelligence, Neural Networks, Carnegie Mellon University.

Perceptron for Linear Regression

- Simple Linear Regression

Bias: This input unit corresponds to the "fake" attribute $x_0 = 1$

Connection with Weight: Neural Network learning program: Adjust the connection weights so that the network generates the correct prediction on the training data

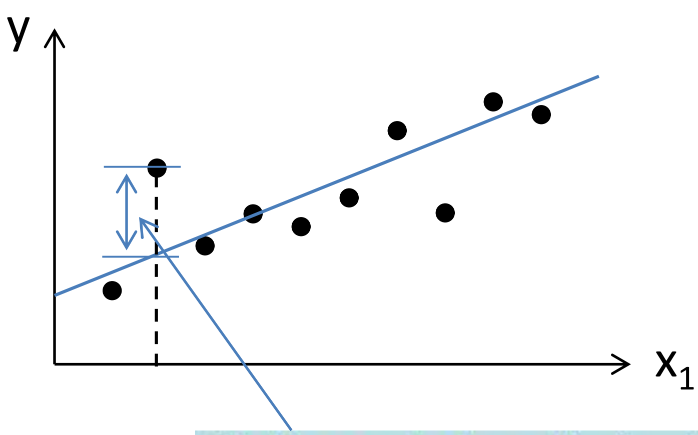


Input **Edge** **Node** **Output**

Source: Martial Hebert. Lecture Note: Artificial Intelligence, Neural Networks, Carnegie Mellon University.

Perceptron for Linear Regression

- Simple Linear Regression
  - We seek a vector of parameters **w** = {w_0, w_1, …, w_M} that minimizes the error between the perdition Y and the estimation using data X



δ_k is the error between the input **x** and the prediction *y* at data point *k*.
Graphically, it the "vertical" distance between data point *k* and the prediction calculated by using the vector of linear parameters **w**.

$$E = \sum_{k=1}^{N} \left(y^k - (w_o x_o^k + w_1 x_1^k + \cdots + w_M x_M^k)\right)^2$$

$$= \sum_{k=1}^{N} \left(y^k - \mathbf{w} \cdot \mathbf{x}^k\right)^2$$

$$= \sum_{k=1}^{N} \delta_k^2$$

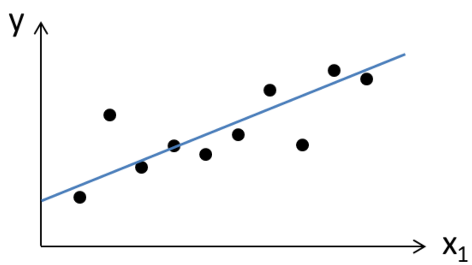
$$\delta_k = y^k - \mathbf{w} \cdot \mathbf{x}^k$$

Source: Martial Hebert. Lecture Note: Artificial Intelligence, Neural Networks, Carnegie Mellon University.

Perceptron

Perceptron for Linear Regression

- Simple Linear Regression
  - Optimization



Error

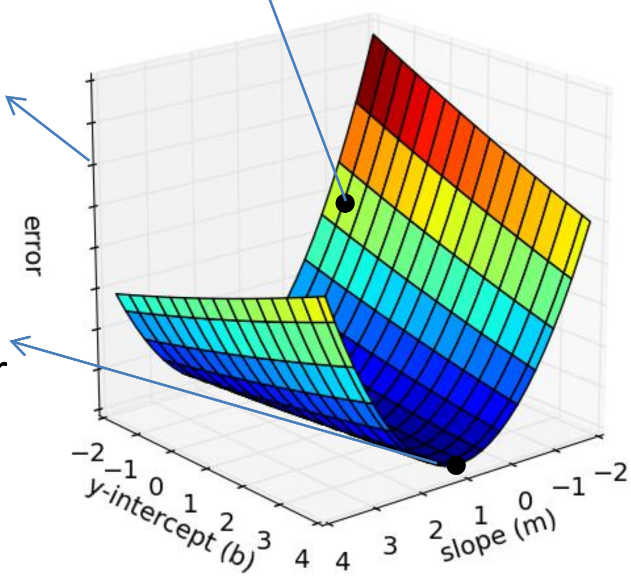
$$\delta_k = y^k - \mathbf{w} \cdot \mathbf{x}^k$$

➔ Minimize δ_k
by searching for optimal parameter **w**

Each point represents a line.

The height (y) is an error value.

The one yielding the smaller error value.



[Example of $x_{M=1}$ Case (1D)]

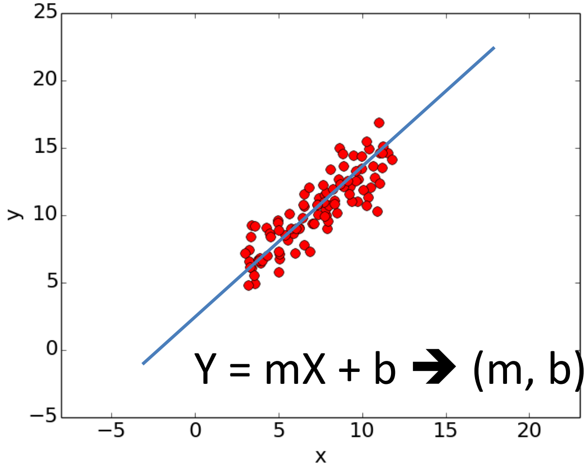
Perceptron

Perceptron for Linear Regression

- Simple Linear Regression
  - Optimization: Gradient Descent Algorithm

Gradient descent is an algorithm that minimizes functions. Given a function defined by a set of parameters, gradient descent starts with an initial set of parameter values and iteratively moves toward a set of parameter values that minimize the function. This iterative minimization is achieved using calculus, taking steps in the negative direction of the function gradient.

Example:



Y = mX + b ➔ (m, b)?

$$\mathrm{Error}_{(\mathrm{m,b})} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

$$\frac{\partial}{\partial \mathrm{m}} = \frac{2}{N}\sum_{i=1}^{N} -x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial \mathrm{b}} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (mx_i + b))$$

Source: Matt Nedrich. https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression.

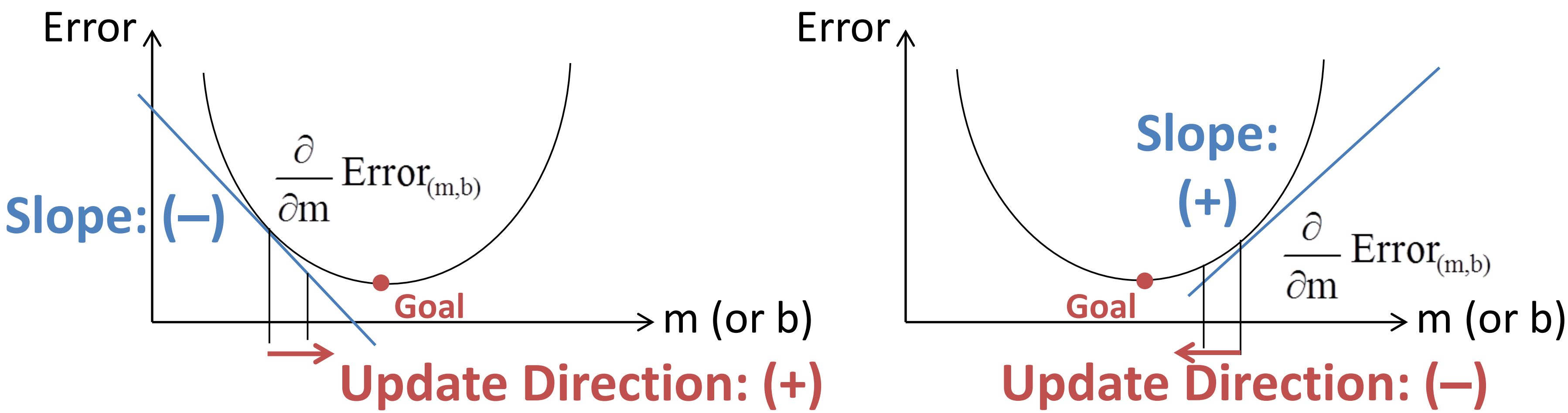
Perceptron

Perceptron for Linear Regression

- Simple Linear Regression
  - Optimization: Gradient Descent Algorithm

    Example:

    - Start at any pair of m and b values (i.e., any line); E.g., (m , b) = (-1, 0)



- Each iteration will update m and b to a line that yields slightly lower error than the previous iteration. The direction to move in for each iteration is calculated using the two partial derivatives

Source: Matt Nedrich. https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression.

Perceptron

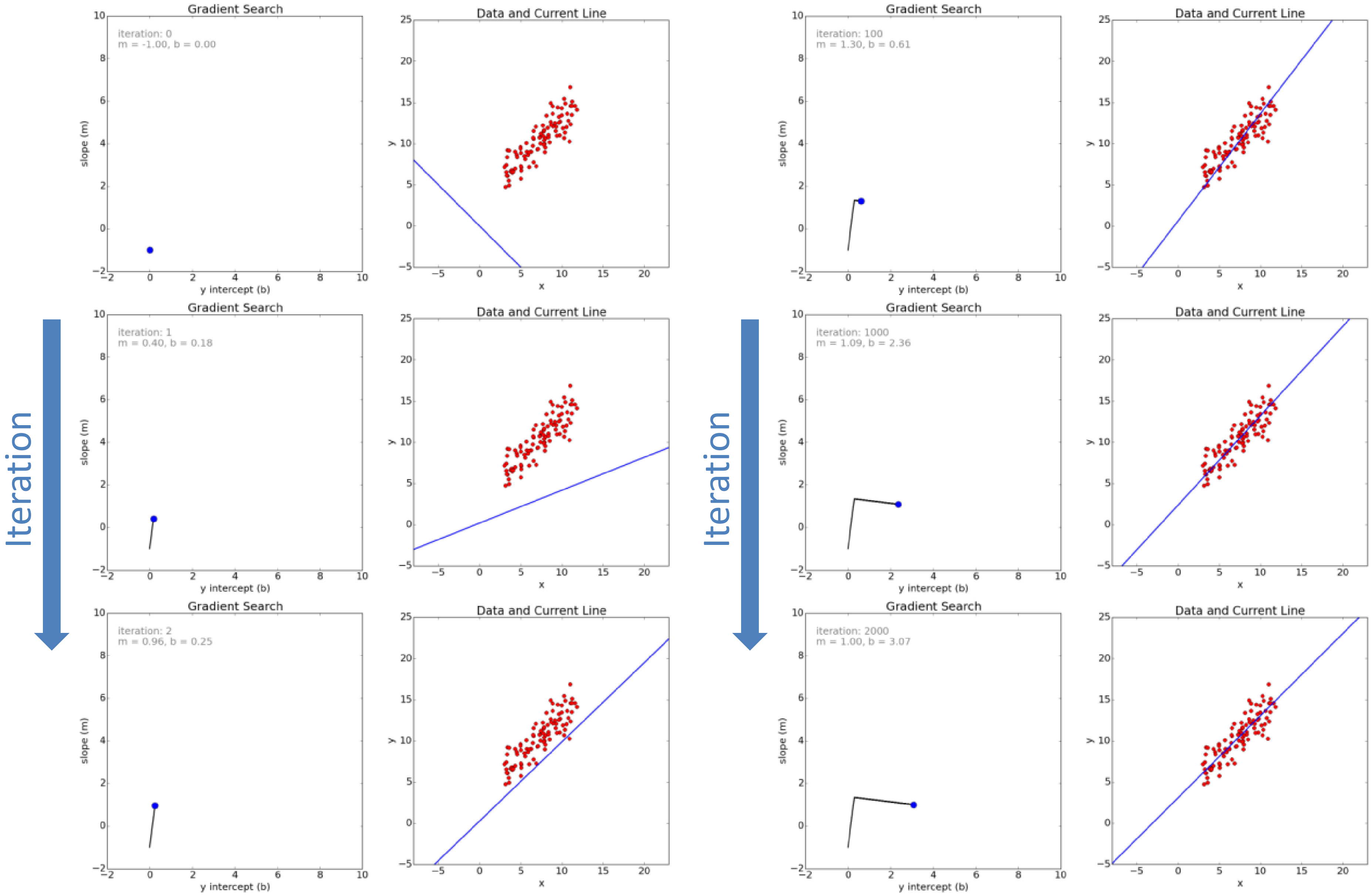
Calculating gradients of the function

```
import numpy as np
def numerical_gradient(function, x):
   h = 0.0001
   grad = np.zeros_like(x)

   for idx in range(x.size):
      tmp_val = x[idx]

      # calculating f(x+h)
      x[idx] = float(tmp_val) + h
      fxh1 = function(x)

      # calculating f(x-h)
      x[idx] = tmp_val - h
      fxh2 = function(x)

      grad[idx] = (fxh1 - fxh2) / (2*h)
      x[idx] = tmp_val  # save the original value

   return grad

def function_2(x):
   return x[0]**2 + x[1]**2

>> numerical_gradient(function_2, np.array([3.0, 4.0]))
>> numerical_gradient(function_2, np.array([0.0, 2.0]))
>> numerical_gradient(function_2, np.array([3.0, 0.0]))
```

Gradient Descent Algorithm

```
def gradient_descent(f, init_x, lr=0.01, step_num=100):
   x = init_x
   x_history = [] # to plot

   for i in range(step_num):
      x_history.append( x.copy() ) # to plot
      grad = numerical_gradient(f, x)
      x -= lr * grad
   return x, np.array(x_history)


>> init_x = np.array([-3.0, 4.0])
>> x, x_history = gradient_descent(function_2,
init_x=init_x, lr=0.1, step_num=100)
>> x


import matplotlib.pylab as plt
plt.plot( [-5, 5], [0,0], '--b')
plt.plot( [0,0], [-5, 5], '--b')
plt.plot(x_history[:,0], x_history[:,1], 'o')

plt.xlim(-3.5, 3.5)
plt.ylim(-4.5, 4.5)
plt.xlabel("X0")
plt.ylabel("X1")
plt.show()
```

Perceptron



Source: Matt Nedrich. https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression.

Perceptron for Linear Regression

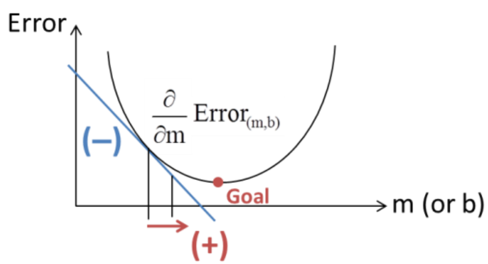
- Simple Linear Regression
  - Optimization: Gradient Descent Algorithm

    - The minimum of E is reached when the derivatives with respect to each of the parameters w_i is zero

$$E = \sum_{k=1}^{N}\left(y^k - (w_o x_o^k + w_1 x_1^k + \cdots + w_M x_M^k)\right)^2$$
$$= \sum_{k=1}^{N}\left(y^k - \mathbf{w}\cdot\mathbf{x}^k\right)^2$$
$$= \sum_{k=1}^{N}\delta_k^2 \qquad \delta_k = y^k - \mathbf{w}\cdot\mathbf{x}^k$$

$$\frac{\partial E}{\partial w_i} = -2\sum_{k=1}^{N}\left(y^k - (w_o x_o^k + w_1 x_1^k + \cdots + w_M x_M^k)\right)x_i^k$$
$$= -2\sum_{k=1}^{N}\left(y^k - \mathbf{w}\cdot\mathbf{x}^k\right)x_i^k$$
$$= -2\sum_{k=1}^{N}\delta_k x_i^k$$

    - Update rule: Move in the direction opposite to the gradient direction



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

Perceptron

Perceptron for Linear Regression

- Simple Linear Regression: Perceptron Training
  - Given input training data x^k with corresponding value y^k

1. Compute error:

$$\delta_k = y^k - \mathbf{w} \cdot \mathbf{x}^k$$

2. Update Neural Network weights:

$$w_i \leftarrow w_i + \alpha \delta_k x_i^k$$

α is the learning rate

- α is too small: May converge slowly and may need a lot of training examples
- α is too large: May change **w** too quickly and spend a long time oscillating around the minimum

Source: Martial Hebert. Lecture Note: Artificial Intelligence, Neural Networks, Carnegie Mellon University.

Final Exam

Time and Location

- December 14 (Fri), 2018, 3-5pm (2 hours)
- Jaesung Civil Eng. Bldg. #201 (morning class) and #204 (afternoon class)

Scope

- Lectures 7-11 + Lab Practice 5-8 (basically, materials after midterm)
- Note that for programming language, basic materials should be known to understand more advanced materials.

Note

- Please bring your own calculator.
- No smart phone allowed.
- Please arrive by 2:50pm so that we can start at 3pm.