Multilayer Perceptron
and
Non-linear Regression Models

Cesar Acosta, PhD.

OVERVIEW

- Neural Network defined
- Multilayer perceptron (MLP)
- Activation functions
- Neural Network as a NL regression model
- Examples

NEURAL NETWORK

- The neural network is a class of machine learning model that uses stacked layers of connected nodes to learn high-level representations from the data
- It usually contains a large number of parameters
- The number of parameters in the model is usually more than the data available (therefore it is called an overparameterized model)

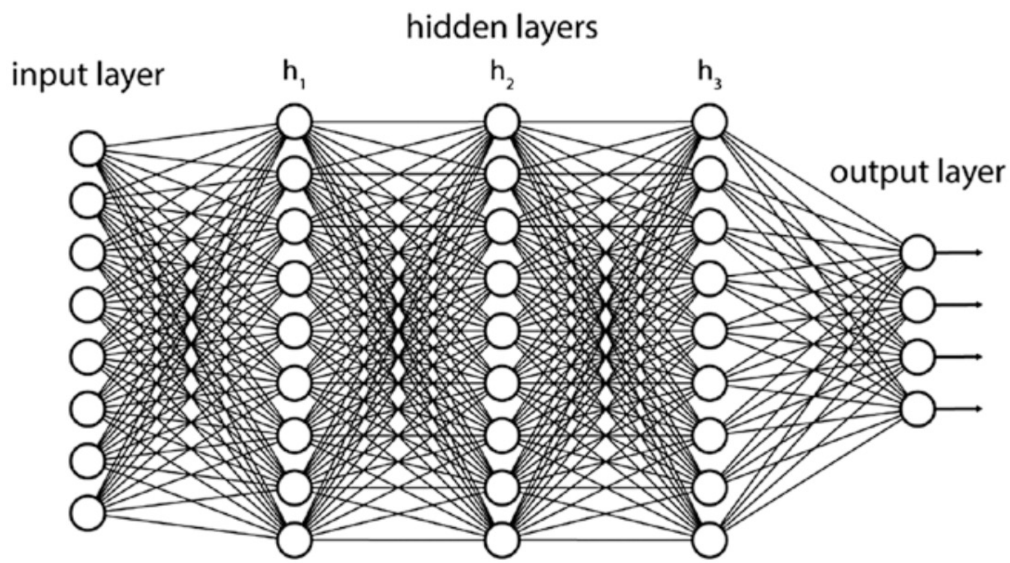
MULTILAYER PERCEPTRON

MULTILAYER PERCEPTRON

- There are many different types of neural networks
- One way to show these types is to classify the neural networks by their architecture
- The basic architecture is the fully connected multiple layer neural network (also called Densely connected neural network)
- It is called the perceptron if it does not have hidden layers
- If the perceptron is supplemented with hidden layers it becomes the multilayer perceptron (MLP)

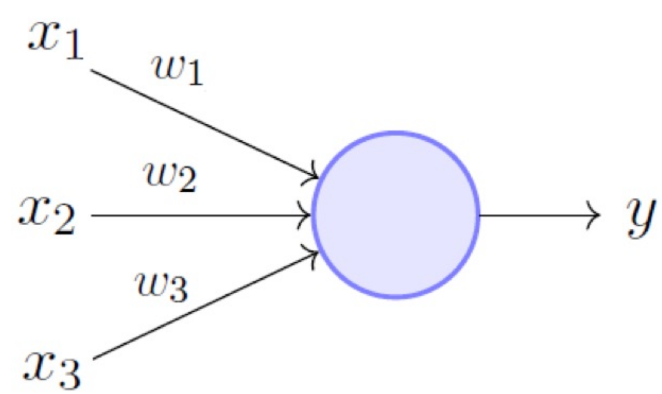
14.1.6 DENSELY CONNECTED NEURAL NETWORK

- There are many different types of neural networks
- One way to show these types is to classify the neural networks by their architecture
- The basic architecture is the fully connected multiple layer neural network (also called Densely connected neural network)



PERCEPTRON

- The Densely connected neural network is called the perceptron if it does not have hidden layers
- If the perceptron is supplemented with hidden layers it becomes the multilayer perceptron

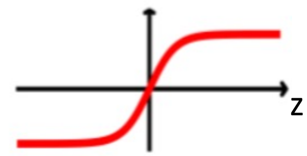


ACTIVATION FUNCTIONS

ACTIVATION FUNCTIONS

- Activation functions are used to make the neural network a nonlinear model
- A nonlinear model is more flexible to adjust to different patterns
- There exist many different activation functions
- The best activation functions for typical ML problems are well known
- For new ML problems, data scientist experiment with new activations

ACTIVATION FUNCTIONS

Logistic (sigmoid)	$\phi(z) = \dfrac{1}{1 + e^{-z}}$	
Hyperbolic Tangent (tanh)	$\phi(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	
ReLU (Rectified Linear Unit)	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	

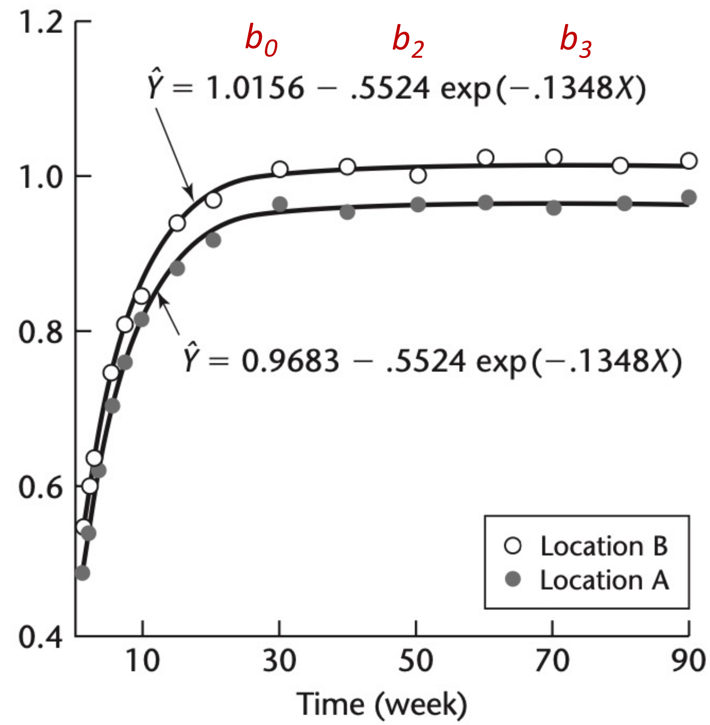
NONLINEAR REGRESSION MODELS

NON LINEAR REGRESSION MODELS

- A regression model that is nonlinear in the parameters, for example

$$\hat{Y} = b_0 + b_1 X_1 + b_3 e^{b_2 X_2}$$

- where X_1 is categorical (two categories A and B)

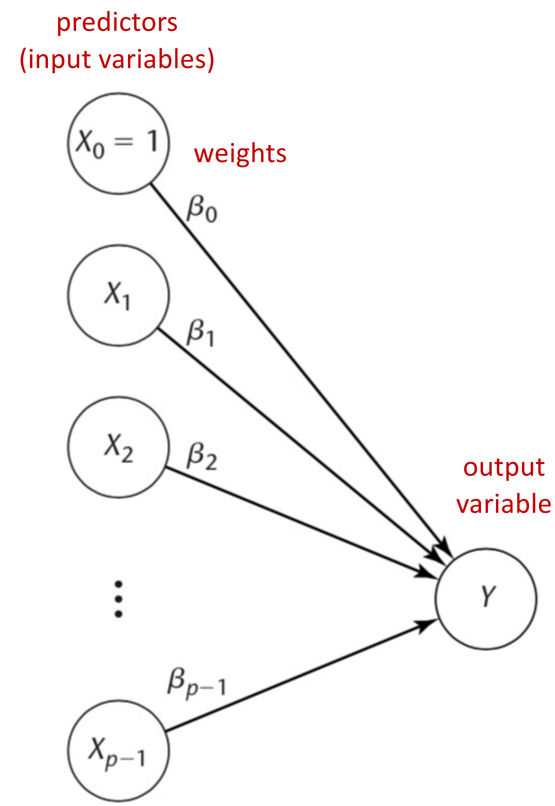


NEURAL NETWORK
AS A
NONLINEAR REGRESSION MODEL

LINEAR REGRESSION MODEL

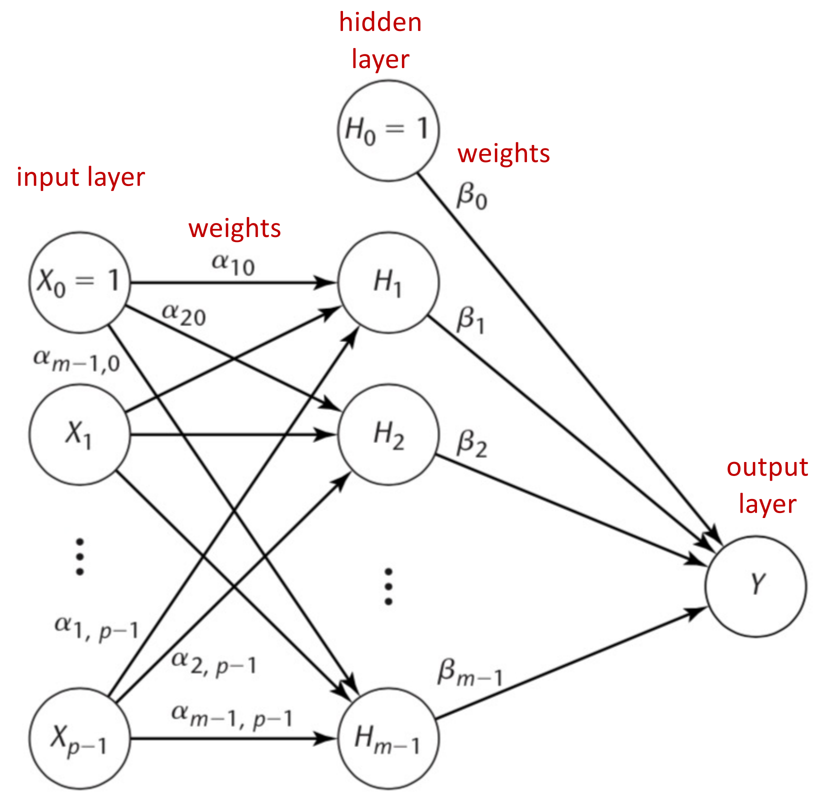
$$Y_i = \beta_0 + \beta_1 X_{i1} + \cdots + \beta_{p-1} X_{i,p-1} + \epsilon_i$$

$$E[Y] = \beta_0 + \beta_1 X_1 + \cdots + \beta_{p-1} X_{p-1}$$



- Nodes represent variables
- Arcs represent regression parameters

ONE HIDDEN LAYER NEURAL NETWORK

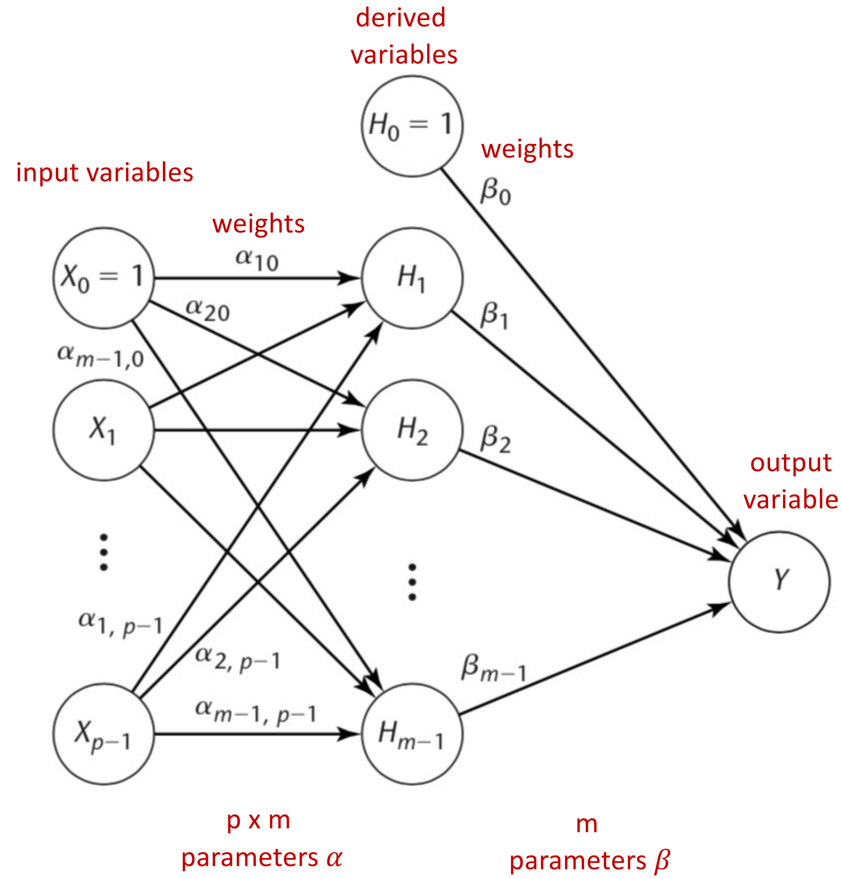


SINGLE HIDDEN LAYER NEURAL NETWORK

regression model

$$E[Y_i] = \beta_0 + \beta_1 H_{i1} + \cdots + \beta_{m-1} H_{i,m-1}$$

$$H_{ij} = \alpha_{j0} + \alpha_{j1} X_{i1} + \cdots + \alpha_{j,p-1} X_{i,p-1}$$



SINGLE HIDDEN LAYER NEURAL NETWORK

With no activation function

$$E[Y_i] = \beta_0 + \beta_1 H_{i1} + \cdots + \beta_{m-1} H_{i,m-1}$$

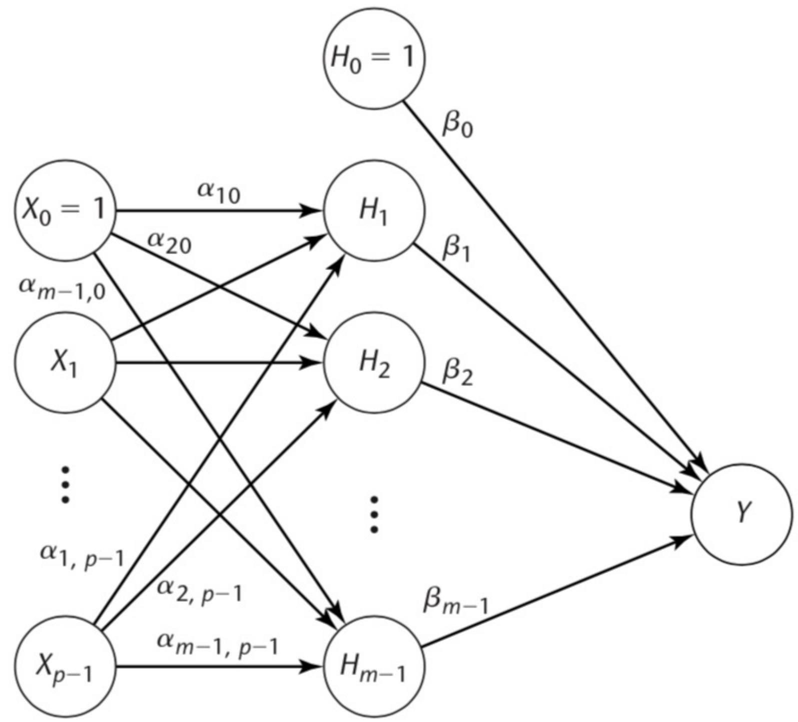
$$H_{ij} = \alpha_{j0} + \alpha_{j1} X_{i1} + \cdots + \alpha_{j,p-1} X_{i,p-1}$$

replacing H_{ij} into the first expression and rearranging

$$E[Y_i] = \beta_0 + \sum_{j=1}^{m-1} \beta_j \alpha_{j0} + \sum_{j=1}^{m-1} \beta_j \alpha_{j1} X_{i1} + \ldots + \sum_{j=1}^{m-1} \beta_j \alpha_{j,p-1} X_{i,p-1}$$

$$E[Y_i] = \left[\beta_0 + \sum_{j=1}^{m-1} \beta_j \alpha_{j0}\right] + \left[\sum_{j=1}^{m-1} \beta_j \alpha_{j1}\right] X_{i1} + \cdots + \left[\sum_{j=1}^{m-1} \beta_j \alpha_{j,p-1}\right] X_{i,p-1}$$

$$E[Y_i] = \beta_0^* + \beta_1^* X_{i1} + \cdots + \beta_{p-1}^* X_{i,p-1}$$



thus, the model with no activation function is a **linear regression model**

SINGLE HIDDEN LAYER NEURAL NETWORK

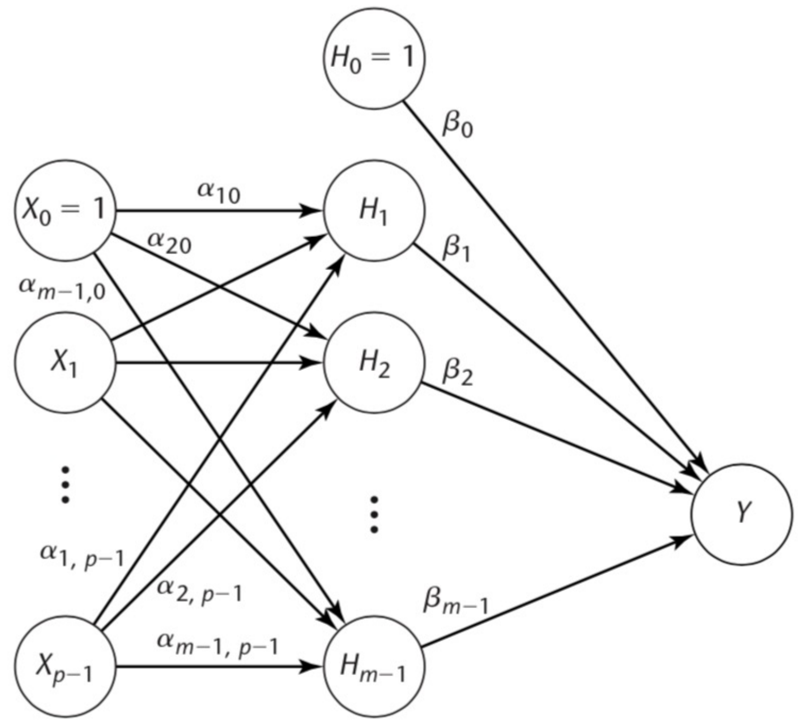
With activation functions $\boldsymbol{g_Y}$ ( ) and $\boldsymbol{g}$( )

$$E[Y_i] = g_Y(\beta_0 + \beta_1 H_{i1} + \cdots + \beta_{m-1} H_{i,m-1})$$

$$H_{ij} = g(\alpha_{j0} + \alpha_{j1} X_{i1} + \cdots + \alpha_{j,p-1} X_{i,p-1})$$

replacing H_{ij} into the first expression and rearranging

$$\begin{aligned} E[Y_i] = g_Y(\beta_0 + & \\ & \beta_1 g(\alpha_{10} + \alpha_{11} X_{i1} + \cdots + \alpha_{1,p-1} X_{i,p-1}) + \\ & \beta_2 g(\alpha_{20} + \alpha_{21} X_{i1} + \cdots + \alpha_{2,p-1} X_{i,p-1}) + \\ & \cdots \\ & \beta_{m-1} g(\alpha_{m-1,0} + \alpha_{m-1,1} X_{i1} + \cdots + \alpha_{m-1,p-1} X_{i,p-1})) \end{aligned}$$



SINGLE HIDDEN LAYER NEURAL NETWORK

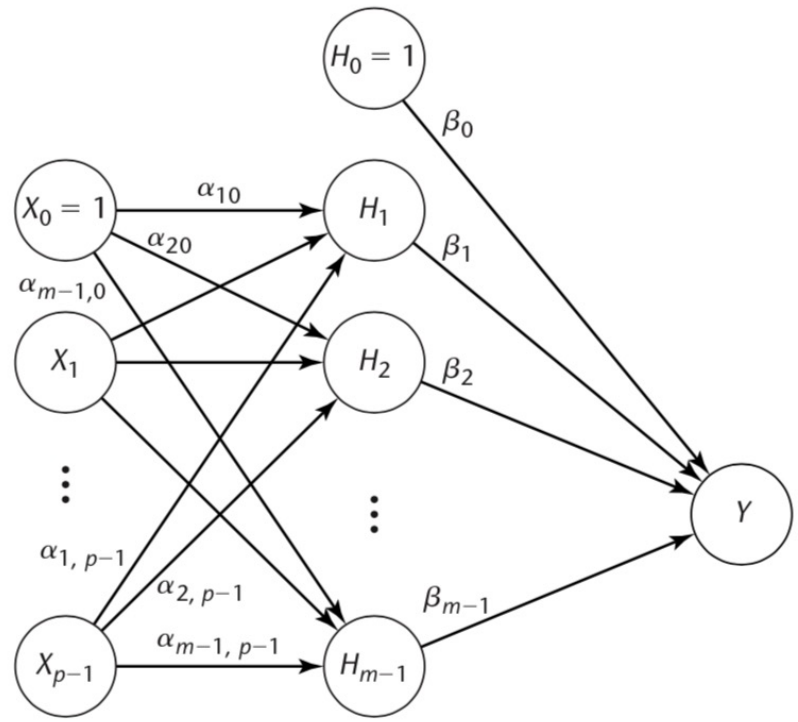
With activation functions g_Y ( ) and g( )

$$E[Y_i] \;=\; g_Y(\beta_0 + \beta_1 H_{i1} + \cdots + \beta_{m-1} H_{i,m-1})$$

$$H_{ij} \;=\; g\,(\alpha_{j0} + \alpha_{j1} X_{i1} + \cdots + \alpha_{j,p-1} X_{i,p-1})$$

replacing H_{ij} into the first expression and rearranging

$$E[Y_i] \;=\; g_Y\left[\beta_0 + \sum_{j-1}^{m-1} \beta_j\, g\,(X_i^t \alpha_j)\right]$$



SINGLE HIDDEN LAYER NEURAL NETWORK

With activation functions $g_Y(\,)$ and $g(\,)$

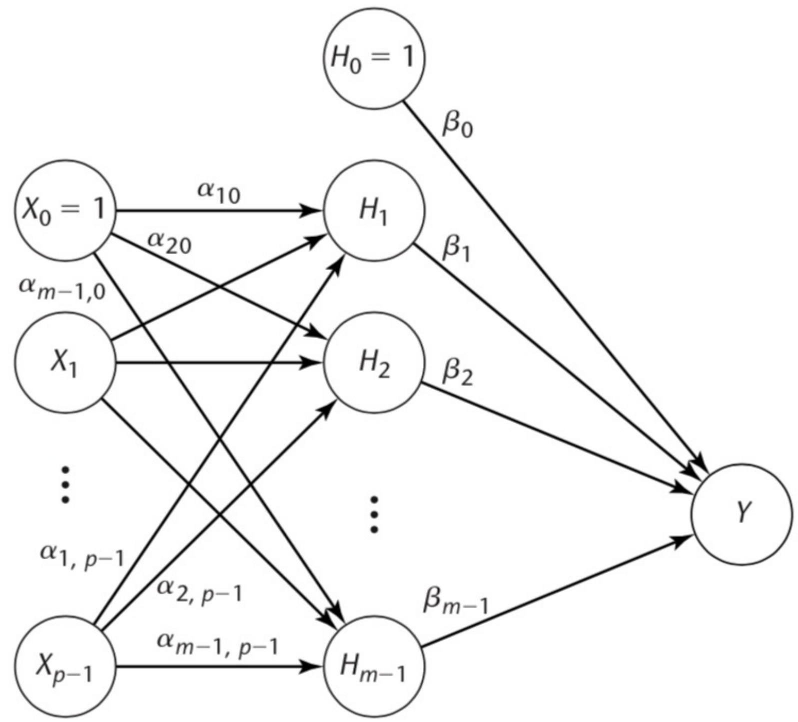
$$E[Y_i] = g_Y(\beta_0 + \beta_1 H_{i1} + \cdots + \beta_{m-1} H_{i,m-1})$$

$$H_{ij} = g(\alpha_{j0} + \alpha_{j1} X_{i1} + \cdots + \alpha_{j,p-1} X_{i,p-1})$$

replacing H_{ij} into the first expression and rearranging

$$E[Y_i] = g_Y\left[\beta_0 + \sum_{j-1}^{m-1} \beta_j \, g(X_i^t \alpha_j)\right]$$

if activation functions are $g(x) = \dfrac{1}{1+e^{-x}} = [1+e^{-x}]^{-1}$



SINGLE HIDDEN LAYER NEURAL NETWORK

With activation functions g_Y ( ) and g( )

$$E[Y_i] = g_Y(\beta_0 + \beta_1 H_{i1} + \cdots + \beta_{m-1} H_{i,m-1})$$

$$H_{ij} = g\left(\alpha_{j0} + \alpha_{j1} X_{i1} + \cdots + \alpha_{j,p-1} X_{i,p-1}\right)$$

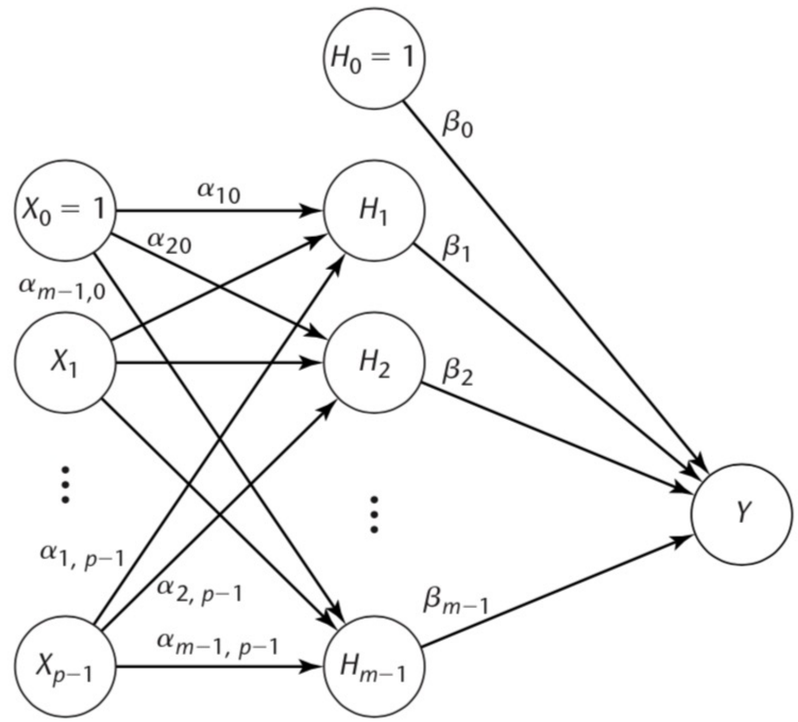
replacing H_{ij} into the first expression and rearranging

$$E[Y_i] = g_Y\left[\beta_0 + \sum_{j-1}^{m-1} \beta_j \, g\left(X_i^t \alpha_j\right)\right]$$

if activation functions are $g(x) = \dfrac{1}{1+e^{-x}} = [1+e^{-x}]^{-1}$

$$E[Y_i] = \left[1 + \exp\left[-\beta_0 - \sum_{j-1}^{m-1} \beta_j \left[1 + \exp(-X_i^t \alpha_j)\right]^{-1}\right]\right]^{-1}$$

thus, the NN model is a **nonlinear regression model**



LINEAR REGRESSION MODEL – LOSS FUNCTION

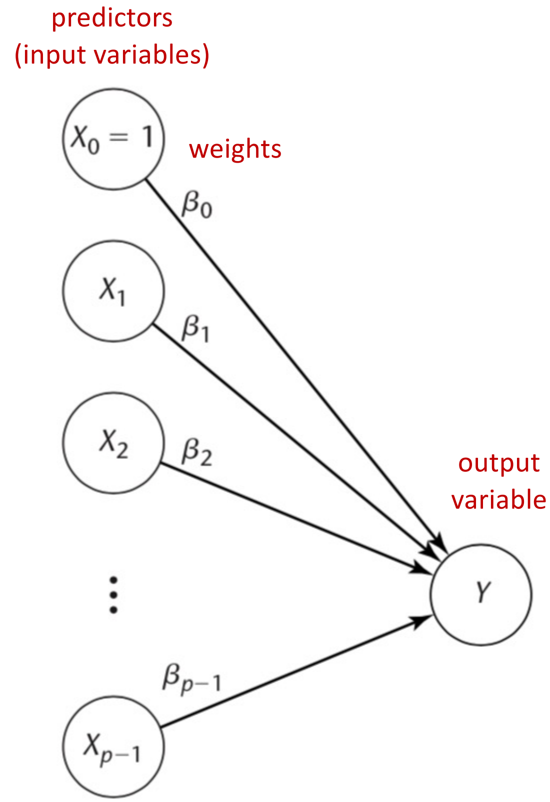
$$Y_i = \beta_0 + \beta_1 X_{i1} + \cdots + \beta_{p-1} X_{i,p-1} + \epsilon_i$$

$$E[Y] = \beta_0 + \beta_1 X_1 + \cdots + \beta_{p-1} X_{p-1}$$

Minimize the loss function to find the weights β_1,.., β_p

$$\underset{\beta_0,\ldots,\beta_p}{\text{Min}} \quad Loss = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\underset{\beta_0,\ldots,\beta_p}{\text{Min}} \quad Loss = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p))^2$$



LINEAR REGRESSION MODEL – LOSS FUNCTION

$$Y_i = \beta_0 + \beta_1 X_{i1} + \cdots + \beta_{p-1} X_{i,p-1} + \epsilon_i$$

$$E[Y] = \beta_0 + \beta_1 X_1 + \cdots + \beta_{p-1} X_{p-1}$$

Minimize the loss function to find the weights $\beta_1,.., \beta_p$

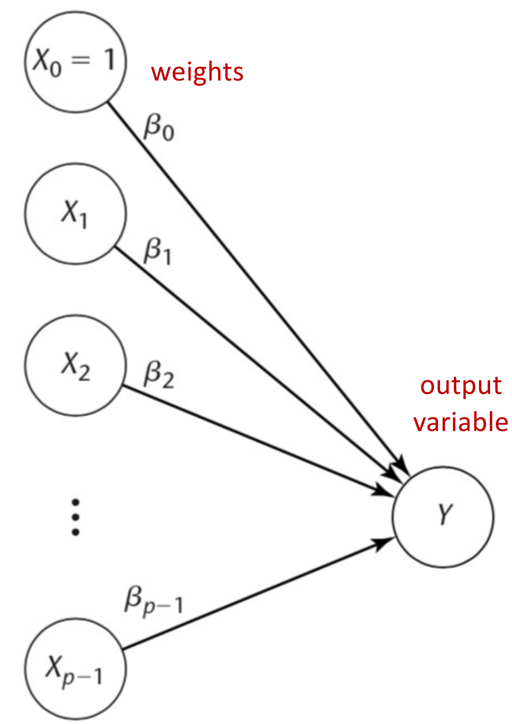
$$\underset{\beta_0,\ldots,\beta_p}{\text{Min}} \quad Loss = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\underset{\beta_0,\ldots,\beta_p}{\text{Min}} \quad Loss = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p))^2$$

$$\underset{\beta_0,\ldots,\beta_p}{\text{Min}} \quad Loss = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{p-1} \beta_i^2$$

to avoid overfitting

penalty term



SINGLE HIDDEN LAYER NEURAL NETWORK – LOSS FUNCTION

With activation functions $g_Y(\,)$ and $g(\,)$

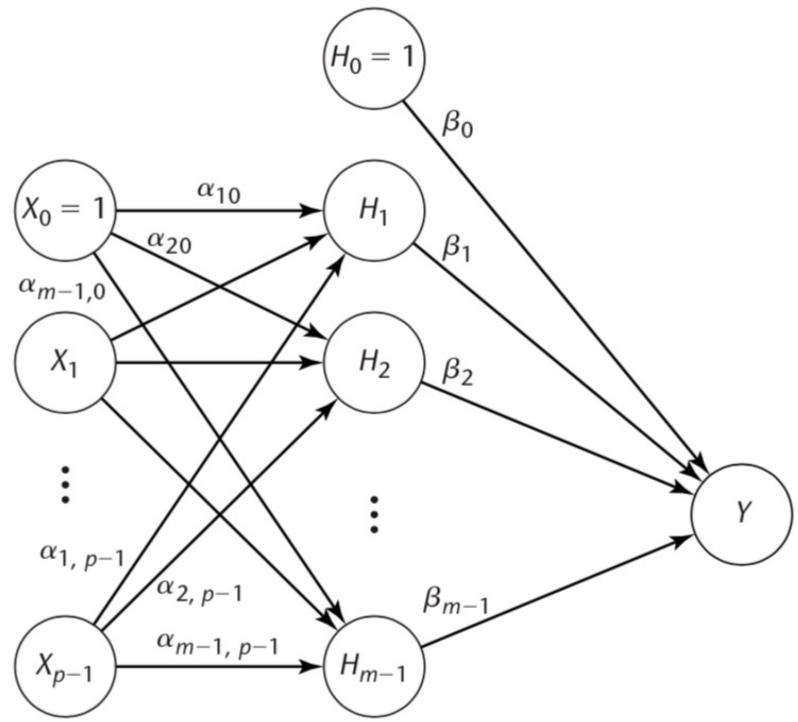
$$E[Y_i] \;=\; g_Y(\beta_0 + \beta_1 H_{i1} + \cdots + \beta_{m-1} H_{i,m-1})$$

$$H_{ij} \;=\; g\left(\alpha_{j0} + \alpha_{j1} X_{i1} + \cdots + \alpha_{j,p-1} X_{i,p-1}\right)$$

Minimize the loss function to find the weights $\alpha_{10},..,\alpha_{mp}$, $\beta_1,..,\beta_p$

$$\underset{\beta_0,\ldots,\beta_p}{\text{Min}} \quad Loss \;=\; \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\hat{y}_i \;=\; g_Y\left[\beta_0 + \sum_{j-1}^{m-1} \beta_j \, g\left(X_i^t \alpha_j\right)\right]$$



p x m
parameters α

SINGLE HIDDEN LAYER NEURAL NETWORK – LOSS FUNCTION

With activation functions $g_Y(\ )$ and $g(\ )$

$$E[Y_i] = g_Y(\beta_0 + \beta_1 H_{i1} + \cdots + \beta_{m-1} H_{i,m-1})$$

$$H_{ij} = g(\alpha_{j0} + \alpha_{j1} X_{i1} + \cdots + \alpha_{j,p-1} X_{i,p-1})$$

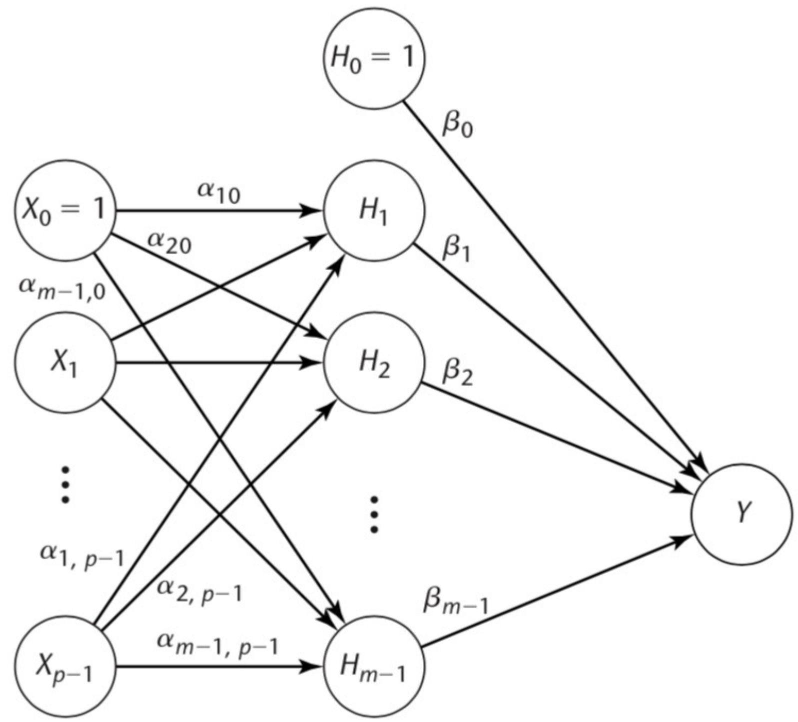
Minimize the loss function to find the weights $\alpha_{10},..,\alpha_{mp}$, $\beta_1,..,\beta_p$

$$\underset{\beta_0,\ldots,\beta_p}{\text{Min}} \quad Loss = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\hat{y}_i = g_Y\left[\beta_0 + \sum_{j-1}^{m-1} \beta_j\, g\left(X_i^t \alpha_j\right)\right]$$

or use a loss with regularization to avoid overfitting

$$\underset{\beta_0,\ldots,\beta_p}{\text{Min}} \quad Loss = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m-1} \beta_i^2 + \lambda \sum_{i=1}^{m-1} \sum_{j=0}^{p-1} \alpha_{ij}^2$$



p x m
parameters α

Cesar Acosta Ph.D.

SINGLE HIDDEN LAYER NEURAL NETWORK

With activation functions g_Y ( ) and g( )

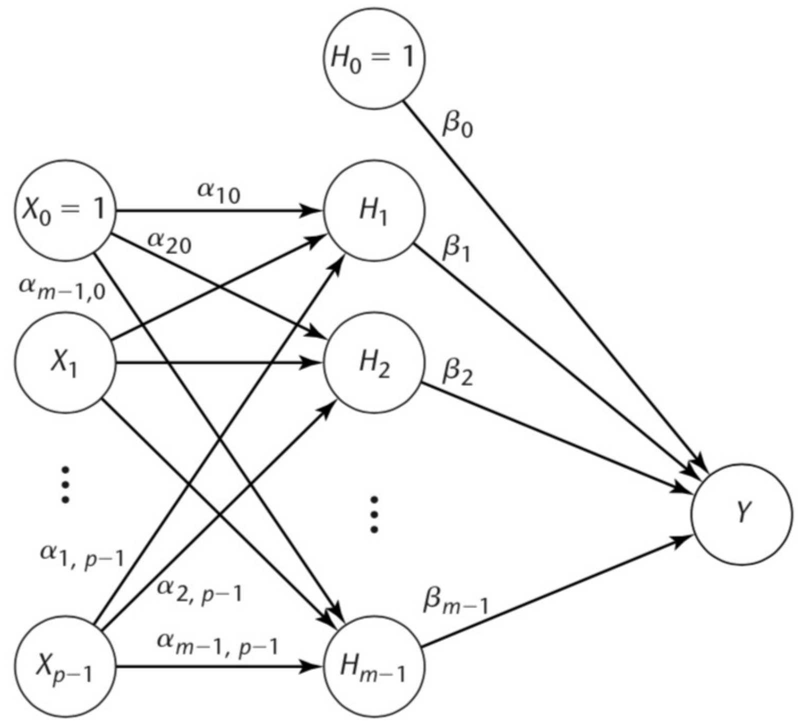
$$E[Y_i] = g_Y(\beta_0 + \beta_1 H_{i1} + \cdots + \beta_{m-1} H_{i,m-1})$$

$$H_{ij} = g(\alpha_{j0} + \alpha_{j1} X_{i1} + \cdots + \alpha_{j,p-1} X_{i,p-1})$$

Minimize the loss function to find the weights α_{10},..,α_{mp}, β_1,.., β_p

$$\underset{\beta_0,\ldots,\beta_p}{\text{Min}} \quad Loss = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m-1} \beta_i^2 + \lambda \sum_{i=1}^{m-1} \sum_{j=0}^{p-1} \alpha_{ij}^2$$

- It is not simple to find the global minimum of this function
- NN models use gradient descent to minimize this function
- This is an iterative process with many steps.
- At each step the algorithm adjusts the weights to a direction that mostly reduces the current loss (the gradient vector direction)
- The process is complete when the loss stops decreasing.
- This process is called Backpropagation.

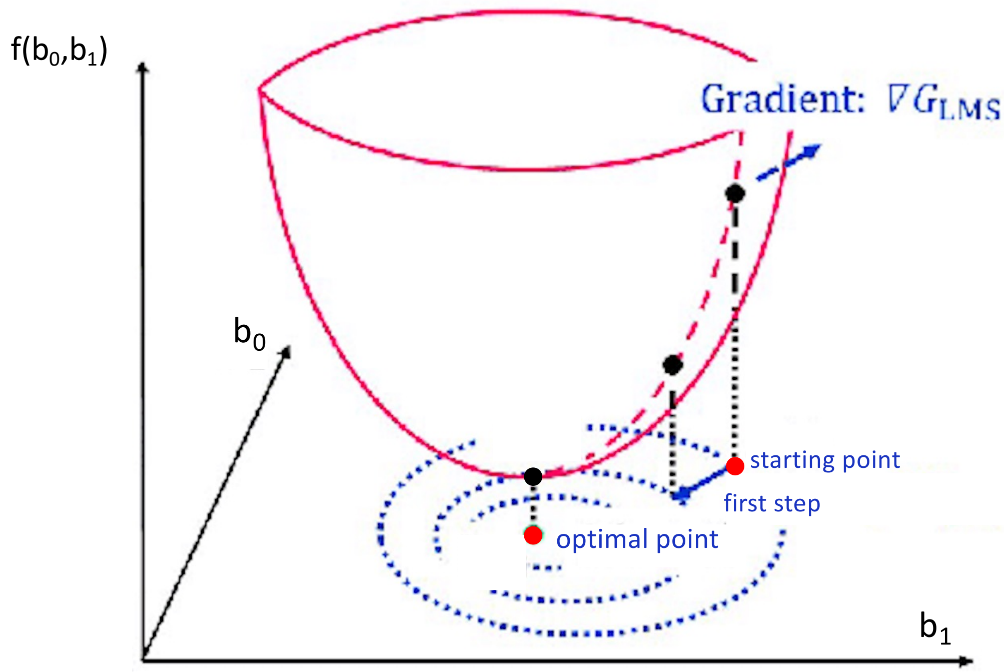


p x m
parameters α

GRADIENT DESCENT

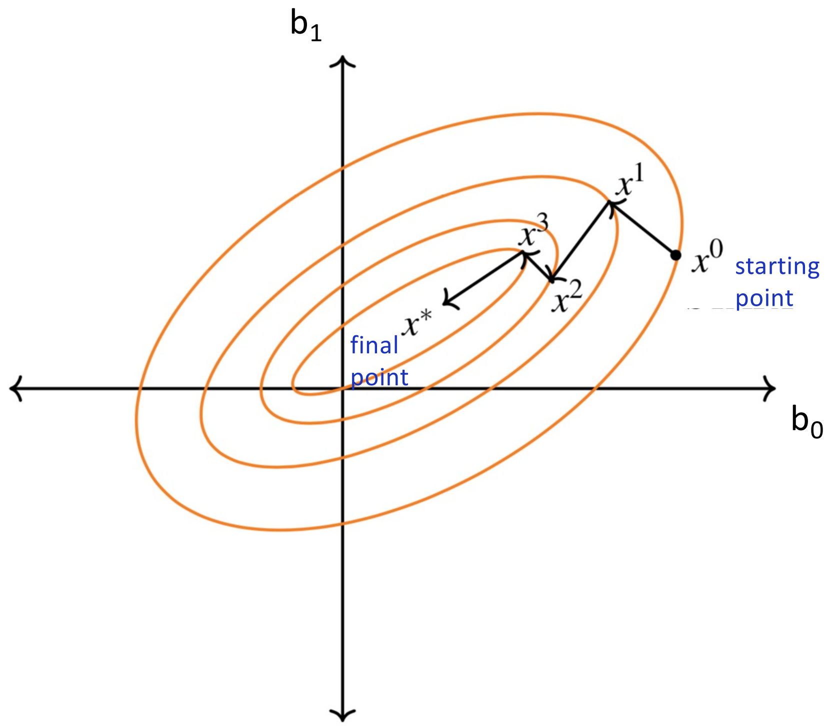
GRADIENT DESCENT

- A search algorithm to find the max (or min) of a function $f(w_1, \ldots, w_p)$
- By starting at an arbitrary location given by $w_1,..,w_p$ it adjusts these coordinates iteratively until the extreme point is found
- The adjustments are in the direction of the steepest slope
- The adjustment amount is called the step size (or learning rate)



GRADIENT DESCENT

- The direction of the steepest slope is given by the gradient vector of $f(w_1, \ldots, w_p)$
- This is a column vector containing the partial derivatives of $f(w_1, \ldots, w_p)$
- The step size is found by trial and error



GRADIENT DESCENT

The loss function for linear regression is

$$
\begin{aligned}
J(w) &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\
&= \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_1 + \cdots + b_p x_p))^2
\end{aligned}
$$

the partial derivatives are

$$
\begin{aligned}
\frac{\partial J(w)}{\partial b_0} &= -2 \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_1 + \cdots + b_p x_p))\, 1 \\
&\vdots \\
\frac{\partial J(w)}{\partial b_p} &= -2 \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_1 + \cdots + b_p x_p))\, x_p
\end{aligned}
$$

Using matrix notation $\nabla \boldsymbol{J}(\boldsymbol{w}) = -2\,\mathbf{X}^T \cdot (\mathbf{y} - \underbrace{\mathbf{X} \cdot \mathbf{b}}_{\hat{y}})$

GRADIENT DESCENT

The loss function for linear regression is

$$J(w) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$= \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_1 + \cdots + b_p x_p))^2$$

the partial derivatives are

$$\frac{\partial J(w)}{\partial b_0} = -2 \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_1 + \cdots + b_p x_p))\, 1$$

$$\vdots$$

$$\frac{\partial J(w)}{\partial b_p} = -2 \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_1 + \cdots + b_p x_p))\, x_p$$

Using matrix notation $\nabla \boldsymbol{J}(\boldsymbol{w}) = \mathbf{X}^T \cdot (\mathbf{y} - \underbrace{\mathbf{X} \cdot \mathbf{b}}_{\hat{y}})$

move to the
opposite direction
of the Gradient

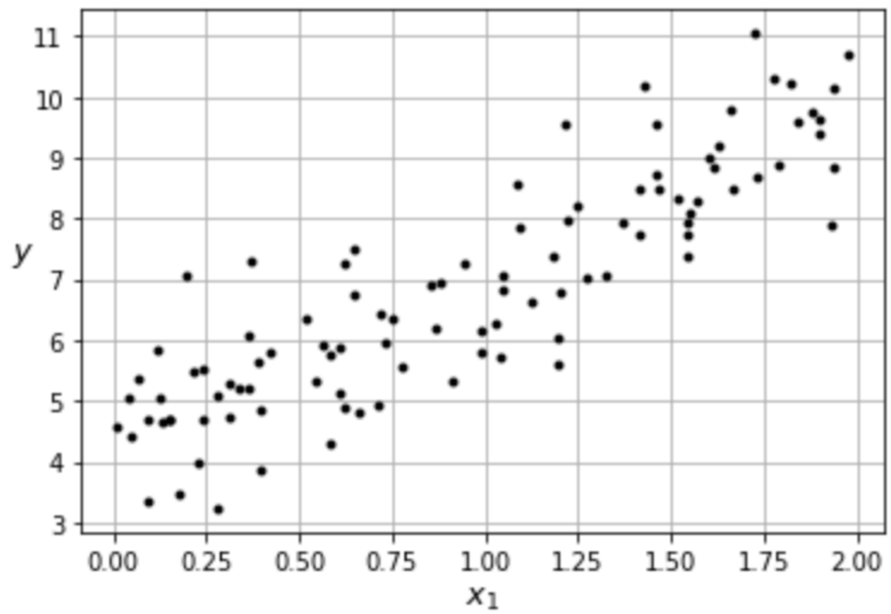
GRADIENT DESCENT

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

```
# Create a data set
```

```
n = 100
np.random.seed(42)
X = 2*np.random.rand(n, 1) # U(0,1) obs
y = 4 + 3 * X + np.random.randn(n, 1)
```

```
plt.plot(X, y, "k.")
plt.xlabel("$x_1$", fontsize=13)
plt.ylabel("$y$", rotation=0, fontsize=13)
plt.grid()
```



GRADIENT DESCENT

```
# add column of ones
X_b = np.c_[np.ones((n,1)), X]
X_b[:5]
```

```
array([[1.        , 0.74908024],
       [1.        , 1.90142861],
       [1.        , 1.46398788],
       [1.        , 1.19731697],
       [1.        , 0.31203728]])
```

```
b1 = np.dot(X_b.T,X_b)
b1 = np.linalg.inv(b1)
b2 = np.dot(X_b.T,y)
beta = np.dot(b1,b2)
beta
```

apply OLS formula

```
array([[4.21509616],
       [2.77011339]])
```

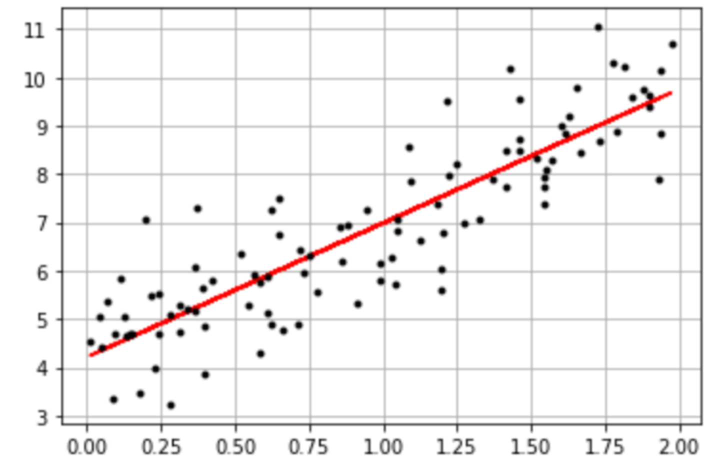
```
yhat = np.dot(X_b,beta)
error = y - yhat
```

find the residuals

```
# loss is the sum of squared errors
cost = np.dot(error.T,error)
cost
```

```
array([[80.6584564]])
```

```
plt.plot(X,yhat,'r')
plt.plot(X, y,"k.")
plt.grid()
plt.show()
```



GRADIENT DESCENT

```
eta = 0.001
```

step size

```
np.random.seed(1)
beta = np.random.randn(2,1)
print(beta[0],beta[1])
```

```
[1.62434536] [-0.61175641]
```

choose a starting pt.

```
for i in range(999):
    error = y - np.dot(X_b,beta)
    gradients = np.dot(X_b.T,error)
    beta = beta + eta * gradients
    cost = np.dot(error.T,error)
```

999 steps

∇J(w)

$$\nabla J(w) = -2\,\mathbf{X}^T \cdot (\mathbf{y} - \mathbf{X} \cdot \mathbf{b})$$

$\hat{y}$

error

gradient

GRADIENT DESCENT

```
eta = 0.001
```

step size

```
np.random.seed(1)
beta = np.random.randn(2,1)
print(beta[0],beta[1])
```

```
[1.62434536] [-0.61175641]
```

choose a starting pt.

```
for i in range(999):
    error = y - np.dot(X_b,beta)
    gradients = np.dot(X_b.T,error)
    beta = beta + eta * gradients
    cost = np.dot(error.T,error)
```

999 steps

$\nabla J(w)$

```
beta
```

```
array([[4.21509617],
       [2.77011338]])
```

```
cost
```

```
array([[80.6584564]])
```

OLS with formula $(X^TX)^{-1}(X^T y)$

```
b1 = np.dot(X_b.T,X_b)
b1 = np.linalg.inv(b1)
b2 = np.dot(X_b.T,y)
beta = np.dot(b1,b2)
beta
```

```
array([[4.21509616],
       [2.77011339]])
```

```
# loss is the sum of squared errors
cost = np.dot(error.T,error)
cost
```

```
array([[80.6584564]])
```

GRADIENT DESCENT

```
eta = 0.001
```

learning rate

```
np.random.seed(1)
beta = np.random.randn(2,1)
print(beta[0],beta[1])
```

```
[1.62434536] [-0.61175641]
```

```
for i in range(999):
    error = y - np.dot(X_b,beta)
    gradients = np.dot(X_b.T,error)
    beta = beta + eta * gradients
    cost = np.dot(error.T,error)
```

```
beta
```

```
array([[4.21509617],
       [2.77011338]])
```

```
cost
```

```
array([[80.6584564]])
```

$$\nabla J(w) = -2\,\mathbf{X}^T \cdot (\mathbf{y} - \mathbf{X} \cdot \mathbf{b})$$

```
yhat = np.dot(X_b,beta)
```

```
plt.plot(X,yhat,'r')
plt.plot(X, y,"k.")
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.show()
```