$w(n+1) = w(n) + \Delta w(n)$

$$\boxed{w(n+1) = w(n) - \eta\, g(n) H^{-1}(n)}$$

Gauss-Newton's Method:

$$\to C(n) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

→ Linearize cost function (n) e

$$\to e(i,w) = e(i) + \left[\frac{\partial e(i)}{\partial w}\right]_{w=w(n)} (w - w(n)) \to ①$$

Linearization
$L(x) = f(a) + f'(a)(x-a)$
$\hookrightarrow w(n)$

where $i = 1, 2, 3 \ldots, n$

Matrix notation:

$$e(n,w) = e(n) + J(n)(w - w(n)) \to ②$$

$e(n) \to$ is error vector

$e(n) = [e(1), e(2), \ldots, e(n)]$

$J(n)$ is a Jacobian matrix

$$\frac{\partial e(i)}{\partial (w)} = \left[\frac{\partial e(i)}{\partial w_1}, \frac{\partial e(i)}{\partial w_2}, \ldots, \frac{\partial e(i)}{\partial w_n}\right]^T$$

$$J(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_m} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \frac{\partial e(2)}{\partial w_3} & \cdots & \frac{\partial e(2)}{\partial w_m} \\ \vdots & \vdots & \vdots & & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \frac{\partial e(n)}{\partial w_3} & \cdots & \frac{\partial e(n)}{\partial w(m)} \end{bmatrix}$$

$$w(n+1) = \arg\min \frac{1}{2}\|e(n,w)\|^2$$

sub in eq ②

$$\frac{1}{2}\|e(n,w)\|^2 = \frac{1}{2}e^2(n) + e(n)J(n)(w - w(n)) + \frac{1}{2}(w - w(n))^T J^T(n) J(n)(w - w(n)) \to ③$$

Take derivative of eq(3) w.r.t ω

$$e(n)\,J(n) + J^T(n)\,J(n)\,(\omega - \omega(n)) = 0$$

$$J^T(n)\,J(n)\,(\omega - \omega(n)) = -e(n)\,J(n)$$

$$\omega - \omega(n) = \frac{-e(n)\,J\cancel{(n)}}{J^T(n)\,J\cancel{(n)}}$$

or

$$\omega - \omega(n) = -e(n)\,J(n)\,(J^T(n)\,J(n))^{-1}$$

$$\boxed{\omega = \omega(n) - e(n)\,J(n)\,(J^T(n)\,J(n))^{-1}} \rightarrow (4)$$

Perceptron Convergence:

- Perceptron is used to perform classification task
- Simple perceptron can perform binary classification only.
- It can classify data if there is linearly seperable boundary.
- Error $e_i = t_i - y_i$

$$e_i = \underbrace{t_i}_{\text{target output}} - \underbrace{y_i}_{\text{actual output}}$$

- $y = w^T x + b \Rightarrow y > 0 \quad 1 \quad C_1$ (Assumption)

  $y \leq 0 \quad 0 \quad C_2$

- When $\langle x, c_1 \rangle$ is data we have, if perceptron output for input x as c_1 then we don't change weight.

  $\omega(n+1) = \omega(n)$

- If there is a misclassification, then we need to update free parameters

$w(n+1) = w(n) - \eta(n) \cdot x(n)$

sample: $\langle x, c_1 \rangle$

perception: $\langle x, c_2 \rangle$

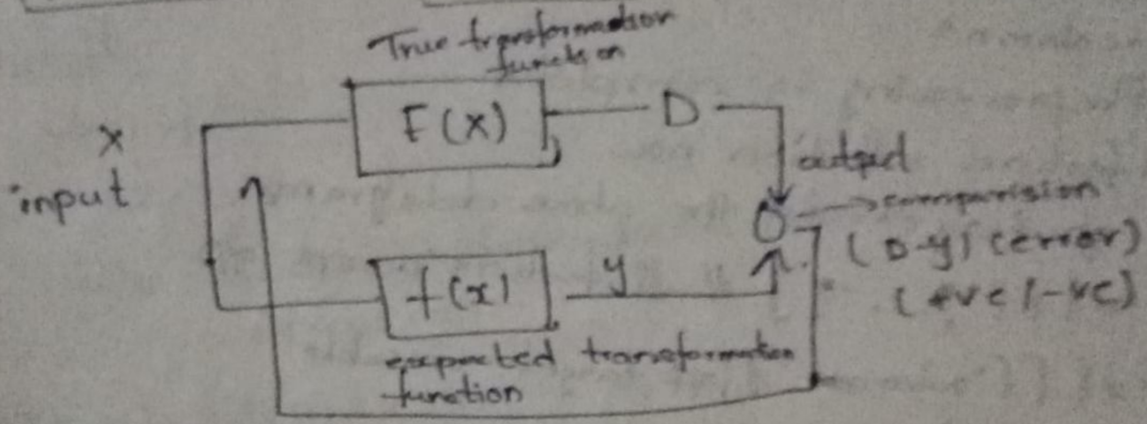
⇒ Sample $\langle x, c_2 \rangle$

perception o/p: $\langle x, c_1 \rangle$

$w(n+1) = w(n) + \eta(n) \cdot x(n)$

Optimization

Approximation procedure



$x \rightarrow F(x) \rightarrow D$

i/p (takes a function) o/p

→ Now we are choosing f(n) which we know then it produces o/p y.

→ The perception doing the approximation of the function which produces o/p in a single attempt (i.e f(x))

row data

↓

preprocessing

- handle NaN
- Standardization / Normalization

feature selection

↓

train, test set splitting.

```
import pandas as pd
df=pd.read_csv("\ 1.csv")
df.info()
df["Glucose"].fillna(df["Glucose"].mean(),inplace=True)
df["Blood Pressure"].fillna(df["B--P"].mean(), Inplace=True)
df["BMI"].fillna(df["BMI"].mean(),inplace=True)
df.head()
df.info()
df.columns
# Preprocessing is completed
# feature selection now .
# o/p-target in the above dataframe.
x=df[["...."]] #all feature accept o/p
y=df[["outcome"]]# target variable
#splitting data
from sklearn.model_selection import train_test_split
from sklearn.linear_model import perceptron
xtr, xte, ytr, yte = train_test_split(x, Y, test_size=0.2)
print (xtr.shape)   (614, 8)
print (xte.shape)   (154,8)
# Create model
p=perceptron(max_iter=1200)
p.fit (xtr, ytr)
p.predict (xte)
p.score(xte,yte)
# accuracy -> score()
o/P = 0.6336
  ≈ 63%
```

9/9/22

Perceptron Convergence

Misclassification

$\langle x, C_1 \rangle \quad C_2$

$\langle x, C_2 \rangle \quad C_1$

$w(n+1) = w(n) - x(n) \quad x \in C_2 \rightarrow ①$

$w(n+1) = w(n) + x(n) \quad x \in C_1 \rightarrow ②$

initially

$\rightarrow \eta = 1$

$\rightarrow w(0) = 0$

$w_0 \rightarrow$ weight vector where perceptron gives correct o/p

$w(n+1) = w(n) + x(n) \quad n = 1, 2, 3, \ldots m$

$w(1) = w(0) + x(0)$

$w(2) = x(0) + x(1)$

$w(3) = x(1) + x(2)$

$w(n+1) = x(1) + x(2) + \ldots x(m) \rightarrow ③$

Multiply ③ with w_0

$w_0 w(n+1) = w_0 x(1) + w_0 x(2) + \ldots w_0 x(m)$

$\alpha = \min w_0 x(n)$

$w_0 w(n+1) \geq n\alpha$

The unknown boundary can be measured by using Cauchy-Scwarz inequality rule

$\|w_0\|^2 \, \|w(n+1)\|^2 \geq \|w_0 w(n+1)\|^2$

$\|w_0\|^2 \, \|w(n+1)\|^2 \geq \|n\alpha\|^2$

$\geq n^2 \alpha^2$

9/9/22

Perceptron Convergence

Misclassification

$\langle x, C_1 \rangle \quad C_2$

$\langle x, C_2 \rangle \quad C_1$

$w(n+1) = w(n) - x(n) \qquad x \in C_2 \rightarrow (1)$

$w(n+1) = w(n) + x(n) \qquad x \in C_1 \rightarrow (2)$

initially

$\rightarrow \eta = 1$

$\rightarrow w(0) = 0$

$w_0 \rightarrow$ weight vector where perceptron gives correct o/p

$w(n+1) = w(n) + x(n) \qquad n = 1, 2, 3, \ldots m$

$w(1) = w(0) + x(0)$

$w(2) = x(0) + x(1)$

$w(3) = x(1) + x(2)$

$w(n+1) = x(1) + x(2) + \cdots x(m) \rightarrow (3)$

Multiply (3) with w_0

$w_0 w(n+1) = w_0 x(1) + w_0 x(2) + \cdots w_0 x(m)$

$\alpha = \min w_0 x(n)$

$w_0 w(n+1) \geq n\alpha$

The unknown boundary can be measured by using Cauchy - Scwarz inequality rule

$\|w_0\|^2 \|w(n+1)\|^2 \geq \|w_0 w(n+1)\|^2$

$\|w_0\|^2 \|w(n+1)\|^2 \geq \|n\alpha\|^2$

$\geq n^2\alpha^2$

$$||w(n+1)||^2 \geq \frac{n^2\alpha^2}{||w_0||^2}$$

Alternative method for determining a boundary

$w(n+1) = w(n) + x(n) \longrightarrow$ (1)

$\Rightarrow$ apply eucledian norm to eq (1)

$$||w(n+1)||^2 = ||w(n) + x(n)||^2$$

$$= ||w(n)||^2 + ||x(n)||^2 + 2 \times \xi\, w(n)\, x(n)$$

$$= ||w(x)||^2 + ||x(n)||^2$$

$$= ||x(n)||^2$$

$B = \max x(n)$

$$\boxed{||w(n+1)||^2 \leq n\beta}$$

$\rightarrow$ perceptron produces correct output after n_{max} iterations

$\rightarrow$ at $n_{max}+1$ iteration

$$\frac{n^2_{max}\,\alpha^2}{||w_0||^2} = n_{max}\beta$$

Implementing Gradient Descent Algorithm:

$$\boxed{e = d - y}$$

$$y = \Psi\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

$$y = \sum_{i=1}^{n} w_i x_i + b$$

$$\boxed{y = w \cdot x + b}$$

cost function

$$C(w) = \frac{1}{2}\sum_{i=1}^{n} e_i^2$$

$$\boxed{w_{new} = w_{old} - \eta g(n)}$$

$$w(n+1) = w(n) - \eta g(n)$$

$$\Rightarrow g(n) = -\sum_{i=1}^{n} e_i x_i$$

$$\boxed{w_{new} = w_{old} + \eta e x}$$

X = df [[c1, c2, ...cn]]

Y = df [cj]

$$\boxed{w_{new} = w + \eta (d - y) x}$$

To update bias

$$b_{new} = b_{old} - \eta e$$

Implementation in python

```
X = df [ [c1, c2, ...cn] ]
Y = df [cj]

def predict(x, w, b):
    y = np.dot(x, w) + b
    return y
def update (x, w, b, ypre, y, lrate):
    gw = np.dot((y-ypre), x)
    wnew = w + lrate * gw
    return wnew
def gradient_descent(X, Y, lrate, niter):
    #initialize b,w
    b = random.random( )
    w = np.random.rand(x.shape[1])
    w = w.reshape((x.shape, 1))
    for i in range(niter):
        ypre = predict(x, w, b)
        e = y - ypre
        if e != 0:
            w = update(x, w, ypre, y,
                       lrate)
        else:
            break
    print(w)
    print(y-ypre)
```