26/8/22

-Stochastic gradient descent:-

drawbacks of gradient descent:
* trap into the local minima point.
* convergence takes a long time when data is large.

→ Uses only one data sample to compute update.

update rule

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

→ slowly converges to the global minimum error point

-Newtons Method

-Approximation of quadratic cost function, then computes gradient of the approximated cost function.

- uses second order Taylor series expansion to approximate the cost function.

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

$$\therefore f(x) = f(a) + f'(a) f(x) + \frac{1}{2} f(x) H(x) f(x) + \cdots$$

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

$$\approx g(n) \Delta w(n) + \frac{1}{2} \Delta w(n) H(n) \Delta w(n)$$

$H(n) \rightarrow$ Hessian matrix

$$H = \begin{bmatrix} \frac{\partial c}{\partial w_1^2} & \frac{\partial c}{\partial w_1 \partial w_2} & \frac{\partial c}{\partial w_1 \partial w_3} & \cdots & \frac{\partial c}{\partial w_1 \partial w_n} \\ \frac{\partial c}{\partial w_2 \partial w_1} & \frac{\partial c}{\partial w_2^2} & \frac{\partial c}{\partial w_2 \partial w_3} & \cdots & \frac{\partial c}{\partial w_2 \partial w_n} \\ \vdots & & & & \\ \frac{\partial c}{\partial w_m \partial w_1} & \frac{\partial c}{\partial w_m \partial w_2} & \cdots & \cdots & \frac{\partial c}{\partial w_n^2} \end{bmatrix}$$

$$\frac{\partial f}{\partial \Delta w(n)} \quad g(n) + H(n) \Delta w(n) = 0$$

$$H(n) = -\frac{g(n)}{\Delta w(n)} \Rightarrow \Delta w(n) = \frac{-g(n)}{H(n)}$$

$\Delta w(n) = -g(n) H^{-1}(n)$

$w(n+1) = w(n) + \Delta w(n)$ → update amount

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

6/9/22

* Gauss-Newton Method:-

- $c(n) = \frac{1}{2} \sum_{i=1}^{n} e^2(i)$
- Linearize cost fn (or) e
- $e(i,w) = e(i) + \left[\frac{\partial e(i)}{\partial w}\right]_{w=w(n)} (w - w(n))$ → ① where $i = 1, 2, 3, \ldots, n$

Linearization

$L(x) = f(a) + f'(a)(x-a)$

$x \to w$, $a \to w(n)$

Matrix notation:-

$e(n,w) = e(n) + J(n)(w - w(n))$ → ②

$e(n)$ → is error vector.

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

$J(n)$ is a jacobian matrix.

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

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

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

Substitute in 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 (5) w.r.t (w)

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

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

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

(or)

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

$$\boxed{w = w(n) - e(n)\,J(n)\,(J^T(n)\,J(n))^{-1}} \rightarrow (9)$$

$w(n+1)$

7/9/22

* Perceptron convergence:-

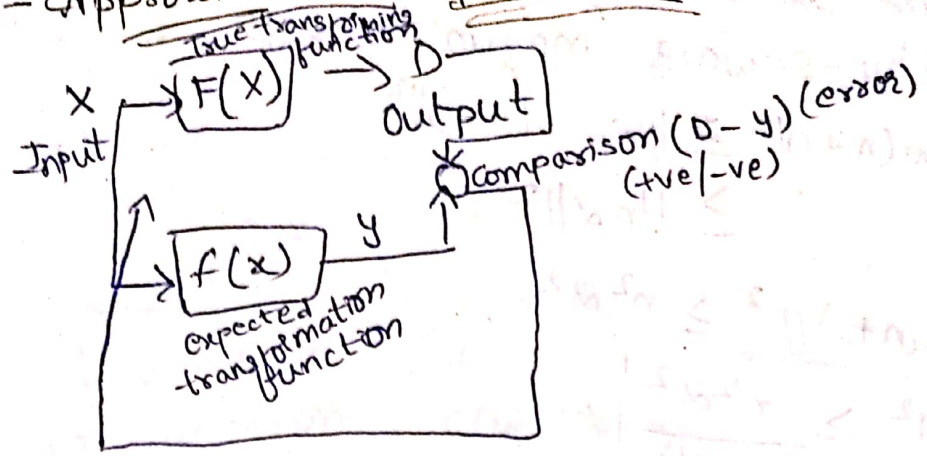
- perceptron is used to perform classification task.
- Simple perceptron can perform binary classification only
- It can classify data if there is linearly seperable boundary.
- Error $e_i = t_i - y_i$
  target output, actual output.
- $y = w^T x + b \Rightarrow y > 0$ 1 C_1 (Assumption)
  $y \leq 0$ 0 C_2
- when $\langle x, C1 \rangle$ is data we have, if perceptron output for input x as $C1$ then we don't change weight.
  $w(n+1) = w(n)$
- If there is a misclassification, then we need to update free parameters
  $w(n+1) = w(n) - \eta(n) \cdot x(n)$

Sample $\langle x, C1 \rangle$
Perceptron $\langle x, C2 \rangle$

- Sample $\langle x, c_2 \rangle$
perceptron o/p $\langle x, c_1 \rangle$

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

- Approximation procedure:-



9/9/22

Let $\eta = 1$

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

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

Initially we assume that

$\eta = 1$; $w(0) = 0$

$w_0 \rightarrow$ weight vector where perceptron gives correct output.

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

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

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

$\vdots$

$w(n+1) = x(0) + x(1) + x(2) + \cdots + x(n) \rightarrow (3)$

Let $x(0) = 0$

$w(n+1) = x(1) + x(2) + \cdots + x(n) \rightarrow (4)$

Multiply (4) with w_0

$w_0 w(n+1) = w_0 x(1) + w_0 x(2) + w_0 x(3) + \cdots + w_0 x(n)$

Let α be positive constant.

$\alpha = \min w_0 x(n)$ for n in $1, 2, \ldots, n$

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

Now Compute the boundary for data items.

Apply cauchy-scwarz inequality rule

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

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

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

- Alternative method:-

$w(n+1) = w(n) + x(n)$ → (5)

⇒ Apply euclidian norm to (5)

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

$\|w(n+1)\|^2 = \|x(n)\|^2$ for $n = 1, 2, \ldots, m$

which is max $x(n)$

Let β be constant

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

- perceptron produces correct output after n_{max} iterations.

- At $n_{max}+1$ iteration

$$\boxed{\frac{n_{max}^2 \alpha^2}{\|w_0\|^2} = n_{max}\beta}$$

13/9/22

*Implementing gradient descent algorithm.

Residual errors

$$e = d - y$$

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

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

$$\boxed{y = w \cdot x + b} \rightarrow (1)$$

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

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

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

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

$$= -e_i x_i$$

$$w_{new} = w_{old} + \eta e_i x_i$$

$$\boxed{w(n+1) = w(n) + \eta e x}$$

$x = df[[c_1, c_2, \ldots, c_n]]$

$y = df[c_j]$

code

```
def predict(x, w, b):
    y = np.dot(x, w) + b
    return y
def update(x, w, b, ypred, y, lrate):
    gw = np.dot(x, (y - ypred))
    wnew = w + lrate * gw
    return wnew
def gradient_descent(x, y, lrate, niter):
    #initialize b, w
```

100 rows
5 colum
(5, w)

100 x 5
5 x 1

[w1, w2, ..., w5]

```
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)
    w = update (x,w,b,ypre,y,lrate)
print(w)
print(y-ypre)
```

- Implement gradient descent algorithm for given activation function.

14/9/22

UNIT-IV

Multi Layer Perceptron

- Multi layer perceptron is also known as feed forward neural network.
- It contains one or more hidden layers between inputs and output layer.
- It can transform non-linear seperable to linear seperable. with the help of hiddenlayers
- It can perform low-level feature extraction. using hidden layers.
- Some of the tasks that can be performed using multilayer perceptron are object detection, handwritten digits recognition etc..
- Each hidden layer extracts some features such as in object detection 1st hidden layer detects horizontal line & 2nd hidden layer vertical lines and so on.
- The output of each hidden layer is passed to next hidden layer. The output layer gets aggregation of all these and it recognises the object.

- Advantages:-
- MLP can perform complex tasks such as multiclass classification.
- It can perform transformations

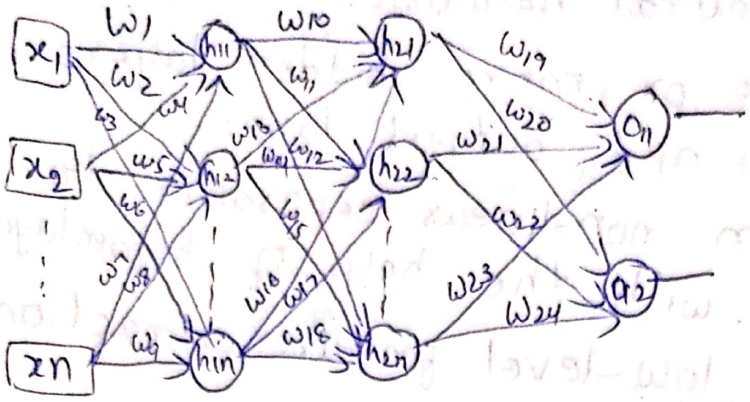
(Single layer MLP)

- Importance of hidden layers:-
- Hidden layers in MLP let it to apply non-linear transformations so that the data is linear seperable.
- can change dimensionality of data.
- extract low level features.

compute gradient

-structure of MLP:-

Input layer | hidden layer1 | hidden layer2 | output layer



- conventions :-

→ w_{ji} → weight of the input from neuron i to j

→ y_j → actual output of neuron j

→ d_j → target output associated with neuron j

→ e_j → error value for neuron j

→ $\psi_j(\,)$ → activation function for neuron j

→ v_j → induced local field value for neuron j

$\therefore y_j = \psi_j(v_j)$

$$v_j = \sum_{i=1}^{n} (w_{ji} * x) + b$$

$$e_j = d_j - y_j$$

$$c(w_{ji}) = \frac{1}{2} \sum_{i=1}^{n} e_j^2$$

$$C_{avg}(w_{ji}) = \frac{1}{N} \sum_{i=1}^{n} e_j^2$$

16/9/22

* Implementation of MLP =

sklearn
└ neural_network
└ MLPClassifier

Constructors of MLP classifier

parameters

① hidden_layer_sizes = tuple

default tuple is (100,) ⇒ 100 neurons & only 1 hidden layer by default.

Ex: tuple is (10, 5, 3) ⇒ then it creates 3 hidden layers in which 1st hidden layer has 10 neurons, 2nd has 5 neurons & 3rd has 3 neurons.

- No. of neurons in output layer = No. of classes in target variable.

- To decide no. of neurons in each hidden layer can be 70% (or) 90% rule

1st hidden layer - no. of neurons
= 70% of input features
(or)
90% of input features.

2nd hidden layer - 70% (or) 90% of neurons in 1st hidden layer.

② activation
- identity
- logistic
- tanh
- relu (default)

③ solver
- To specify optimization algorithm.
- lbfgs - Newton's
- sgd - stochastic gradient descent
- adam - variant of sgd (default)

④ max_iter

(specifies maximum no. of iterations

default it is 200

⑤ tol (tolerance value) $1e^{-4}$

(Specifies stopping criteria)

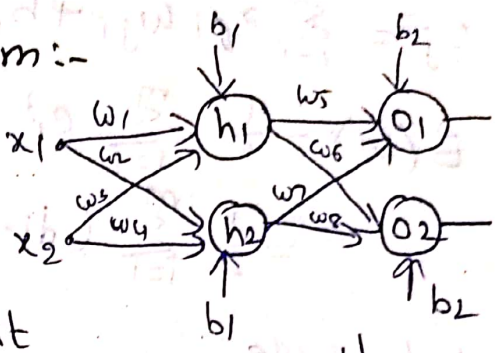
Code

```
from sklearn import datasets
d = datasets.load_iris()
d.data          #returns array present in d (input features)
d.target        #returns output feature array
d.feature_names #returns names of input feature
import pandas as pd
from sklearn.model_selection import
                        train_test_split
from sklearn.neural_network import
                        MLPClassifier
df = pd.DataFrame(d.data, columns=
        ["sepallength", "sepalwidth", "petallength",
         "petalwidth"])
df.head()
df.info()
df.columns
x = df[["sepallength", "sepalwidth", "petallength",
        "petalwidth"]]
y = d.target
xtr, xte, ytr, yte = train_test_split(x, y,
                        test_size = 0.2, random_state
```

```
mc = MLPClassifier(hidden_layer_sizes
     = (4, 3), max_iter = 500,
     activation = "identity", solver = "sgd")
mc.fit(xtr, ytr)
mc.predict(xte)
mc.score(xte, yte)
```

17/9/22

*Back propagation algorithm:-



- 2 phases
  1. Forward pass
  2. Backward pass

① ⇒ network fed with input
- Neurons in hidden layers process the input and pass to next layers.

② ⇒ It is the error correction pass
- error value is computed.
- when there is an error, then back propagation algorithm uses gradient descent to minimize error value.

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

- Computing gradient at o/p layer.

$$E_{total} = E_{O1} + E_{O2} + \dots + E_{On}$$

$$E_{O1} = \frac{1}{2}\sum_{i=1}^{m} e_i^2 \qquad m = \text{size of training set.}$$

$$= \frac{1}{2}\sum_{i=1}^{m} (d_{O1} - y_{O1})^2$$

$$E_{O2} = \frac{1}{2}\sum_{i=1}^{m} e_i^2 = \frac{1}{2}\sum_{i=1}^{m} (d_{O2} - y_{O2})^2$$

20/9/22

At output layer

output layer neuron j.

$e_j = d_j - y_j \rightarrow (1)$

$y_j = \varphi(v_j)$

$y_j = \text{sigmoid}(v_j) = \frac{1}{1+e^{-v_j}} \rightarrow (2)$ (If sigmoid activation function is used)

$v_j = \sum_{i=1}^{m} x_i w_{ji} + b_0 \rightarrow (3)$

$E(w) = \frac{1}{2} \sum_{i=1}^{m} e_i^2 \rightarrow (4)$

$= \frac{1}{2} \sum_{i=1}^{m} (d_i - y_i)^2$

update rule

- as per gradient descent

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

(or)

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

$g(n)$ is gradient of cost function

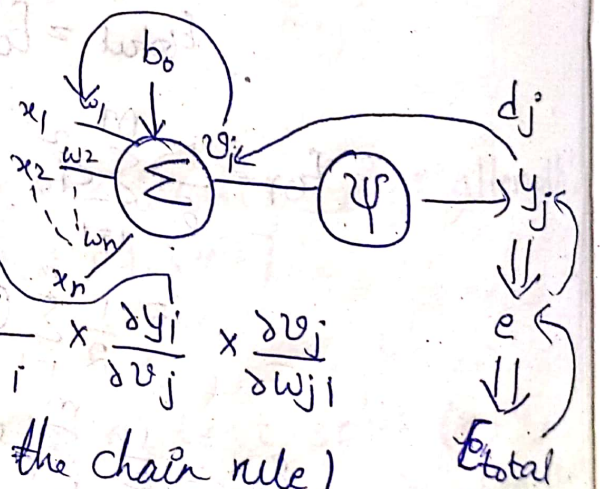
- a neural network with k neurons in output layer

$\mathcal{E}_{total}(w) = \sum_{i=1}^{k} E_i(w)$

$= \mathcal{E}_1(w) + \mathcal{E}_2(w) + \cdots + E_k(w) \rightarrow (5)$

gradient of cost function

$g(n) = \frac{\partial E_{total}}{\partial w_{ji}}$



$\frac{\partial \mathcal{E}_{total}}{\partial w_{ji}} = \frac{\partial \mathcal{E}_{total}}{\partial e_j} \times \frac{\partial e_j}{\partial y_i} \times \frac{\partial y_i}{\partial v_j} \times \frac{\partial v_j}{\partial w_{ji}}$

(by applying the chain rule)

$$\frac{\partial E_{total}}{\partial e_j} = \frac{\partial}{\partial e_j}(E_1(w) + E_2(w) + \cdots + E_k(w))$$

$$= \frac{\partial}{\partial e_j}\left(\frac{1}{2}\sum_{i=1}^{m} e_{i,1}^2 + \frac{1}{2}\sum_{i=1}^{m} e_2^2 + \cdots + \frac{1}{2}\sum_{i=1}^{m} e_j^2 + \cdots + \frac{1}{2}\sum_{i=1}^{m} e_k^2\right)$$

$$= e_j$$

$$\frac{\partial e_j}{\partial y_j} = \frac{\partial}{\partial y_j}(d_j - y_j) = -1$$

~~$\frac{\partial y_j}{\partial v_j} = \frac{\partial}{\partial v_j}\left(\sum_{i=1}^{n} x_i w_{ji} + b_0\right)$~~

$$\frac{\partial y_i}{\partial v_j} = \frac{\partial}{\partial v_j}\left(\frac{1}{1+e^{-v_j}}\right) = \frac{(1+e^{-v_j})(0) - 1(e^{-v_j})}{(1+e^{-v_j})^2}$$

$$= \frac{-e^{-v_j}}{(1+e^{-v_j})^2} = y_j(1-y_j)$$

$$\frac{\partial v_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\left(\sum_{i=1}^{n} w_{ji} x_i + b_0\right)$$

$$= x_i$$

$$g(n) = \frac{\partial E_{total}}{\partial w_{ji}} = e_j \times -1 \times y_j(1-y_j) \times x_j$$

$$= -e_j\, y_j(1-y_j)\, x_j$$

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

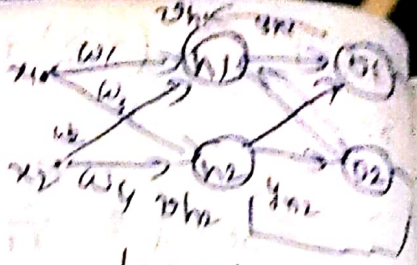
$$\boxed{w_{new} = w_{old} + \eta y(1-y) e_j x_j} \rightarrow (6)$$

21/9/22

At hidden layer

20/7

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



$\Rightarrow g(n) \rightarrow$ gradient of cost function w.r.t weight

ε_{total} = error caused by hidden layer neuron i at output layer neuron j

$$g(n) = \frac{\partial \varepsilon_{total}}{\partial w_{ji}} \rightarrow (1)$$

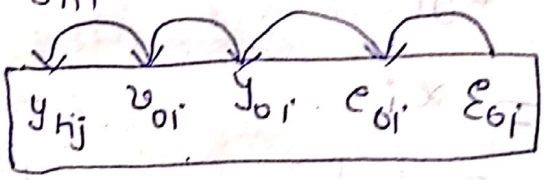
$$\frac{\partial \varepsilon_{total}}{\partial w_{ji}} = \frac{\partial \varepsilon_{total}}{\partial y_j} * \frac{\partial y_j}{\partial v_j} * \frac{\partial v_j}{\partial w_{ji}} \rightarrow (2)$$

$\varepsilon_{total} \rightarrow y_j \rightarrow v_j \rightarrow w_{ji}$

$$\frac{\partial \varepsilon_{total}}{\partial y_j} = \frac{\partial \varepsilon_{o1}}{\partial y_j} + \frac{\partial \varepsilon_{o2}}{\partial y_j} + \cdots + \frac{\partial \varepsilon_{ok}}{\partial y_j} \rightarrow (3)$$

y_j ↓ hidden layer output

$$= \frac{\partial \varepsilon_{o1}}{\partial y_{h1}} + \frac{\partial \varepsilon_{o2}}{\partial y_{h1}}$$



$$\frac{\partial \varepsilon_{oi}}{\partial y_j} = \frac{\partial \varepsilon_{oi}}{\partial e_{oi}} * \frac{\partial e_{oi}}{\partial y_{oi}} * \frac{\partial y_{oi}}{\partial v_{oi}} * \frac{\partial v_{oi}}{\partial y_j} \rightarrow (4)$$

$$\varepsilon_{oi} = \frac{1}{2} \sum_{i=1}^{n} e_{oi}^2$$

$$\frac{\partial \varepsilon_{oi}}{\partial e_{oi}} = e_{oi}$$

$$\frac{\partial e_{oi}}{\partial y_{oi}} = \frac{\partial}{\partial y_{oi}} (y_{oi} - y_{oi}) = -1$$

$$\frac{\partial y_{oi}}{\partial v_{oi}} = \frac{\partial}{\partial v_{oi}} (\varphi(v_{oi})) = y_{oi}(1 - y_{oi})$$

$$\frac{\partial v_{oi}}{\partial y_j} = \frac{\partial}{\partial y_j} \left(\sum_{i=1}^{m} w_{ji} y_j + b_o \right) = w_{ji}$$

$$\frac{\partial E_{oi}}{\partial y_j} = e_{oi}(-1)\, y_{oi}(y - y_{oi})\, w_{ji}$$

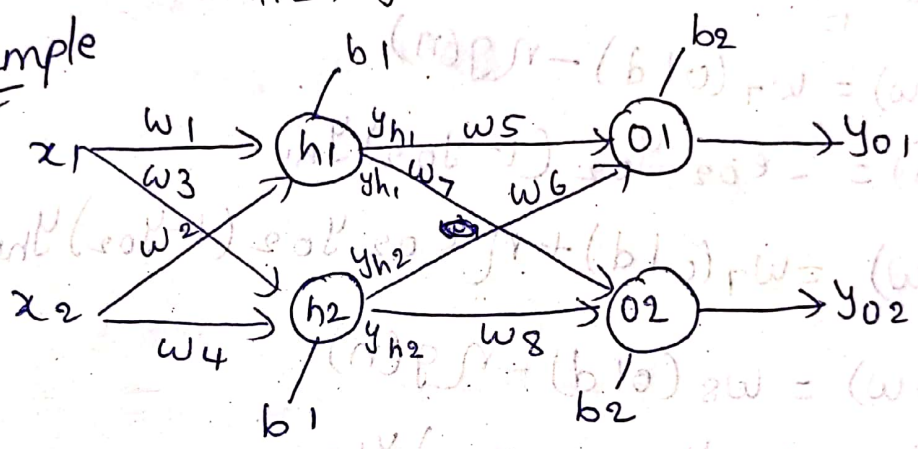
$$\frac{\partial E_{total}}{\partial y_j} = -e_{o1} y_{o1}(y - y_{o1}) w_{h11} - e_{o2} y_{o2}(y - y_{o2}) w_{h12}$$

$$\frac{\partial y_j}{\partial v_j} = y_j(1 - y_j)$$

$$\frac{\partial v_j}{\partial w_{ji}} = x_i$$

$$\therefore \frac{\partial E_{total}}{\partial y_j} = (-e_{o1} y_{o1}(y - y_{o1}) w_{h11} - e_{o2} y_{o2}(y - y_{o2}) w_{h12}) y_j(1 - y_j) x_i$$

Example



$$v_{o1} = y_{h1} \times w_5 + y_{h2} \times w_6 + b_2$$

$$y_{o1} = \text{sigmoid}(v_{o1}) = \frac{1}{1 + e^{-v_{o1}}}$$

$$e_{o1} = d_{o1} - y_{o1}$$

$$E_{o1} = \frac{1}{2} \sum_{i=1}^{n} e_{o1}^2(i)$$

$$= \frac{1}{2} \sum_{i=1}^{n} (d_{o1}(i) - y_{o1}(i))^2$$

$$w_5 \text{ new} = w_5 \text{ old} - \eta g(n)$$

$$g(n) = -e_{o1} y_{o1} (1-y_{o1}) y_{h1}$$

$$w_5(new) = w_5(old) + \eta e_{o1} y_{o1} (1-y_{o1}) y_{h1}$$

$$w_6(new) = w_6(old) - \eta g(n)$$

$$g(n) = -e_{o1} y_{o1} (1-y_{o1}) y_{h2}$$

$$v_{o2} = y_{h1} \times w_7 + y_{h2} \times w_8 + b_2$$

$$y_{o2} = \text{sigmoid}(v_{o2}) = \frac{1}{1+e^{-v_{o2}}}$$

$$e_{o2} = d_{o2} - y_{o2}$$

$$\varepsilon_{o2} = \frac{1}{2} \sum_{i=1}^{n} e_{o2}^2(i)$$

$$= \frac{1}{2} \sum_{i=1}^{n} (d_{o2}(i) - y_{o2}(i))^2$$

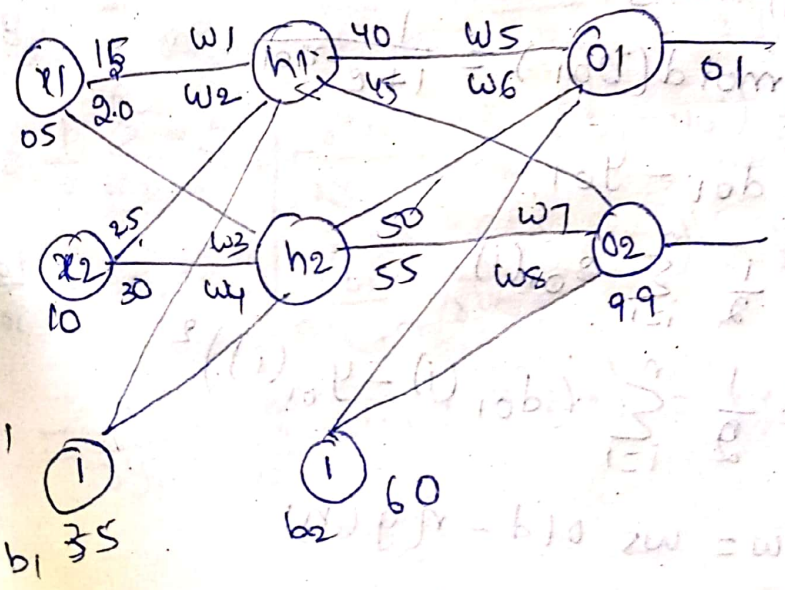
$$w_7(new) = w_7(old) - \eta g(n)$$

$$g(n) = -e_{o2} y_{o2} (1-y_{o2}) y_{h1}$$

$$w_7(new) = w_7(old) + \eta e_{o2} y_{o2} (1-y_{o2}) y_{h1}$$

$$w_8(new) = w_8(old) - \eta g(n)$$

$$g(n) = -e_{o2} y_{o2} (1-y_{o2}) y_{h2}$$



AKV

Iteration 1

forward pass

at h1

$v_{h1} = w_1 x_1 + w_3 x_2 + b_1$

$= 0.15 \times 0.05 + 0.25 \times 0.10 + 0.35$

$= 0.3825$

$y_{h1} = \psi(v_{h1}) = \frac{1}{1+e^{-v_{h1}}} = 0.5945$

at h_2

$v_{h2} = w_2 x_1 + w_4 x_2 + b_1$

$= 0.20 \times 0.05 + 0.30 \times 0.10 + 0.35$

$= 0.39$

$y_{h2} = \psi(v_{h2}) = 0.5962$

	d	y
01	0.01	0.7569
02	0.99	0.769

at 01

$v_{01} = y_{h1} w_5 + y_{h2} w_7 + b_2$

$= 0.594 \times 0.40 + 0.5962 \times 0.5 + 0.60$

$= 1.1359$

$y_{01} = \psi(v_{01}) = 0.7569$

at 0^2

$v_{02} = y_{h1} \times w_6 + y_{h2} \times w_8 + b_2$

$= 0.594 \times 0.45 + 0.5962 \times 0.55 + 0.60$

$= 1.1954$

$y_{02} = \psi(v_{02}) = 0.767$

	d	y
0_1	01	0.7569
0_2	99	0.767

Backward pass

Iteration 1 (o/p layer)

$$w_5(new) = w_5(old) - \eta g(n)$$

$$g(n) = \frac{\partial E_{total}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial e_{o1}} \times \frac{\partial e_{o1}}{\partial y_{o1}} \times \frac{\partial y_{o1}}{\partial v_{o1}} \times \frac{\partial v_{o1}}{\partial w_5}$$

$$= e_{o1} \times -1 \times y_{o1}(1-y_{o1}) \times y_{h1}$$

$$= -e_{o1} \times y_{o1} \times (1-y_{o1}) \times y_{h1}$$

$$= -(d-y)\left(y_{o1} \times (1-y_{o1})\right) y_{h1}$$

$$= -(0.01-0.7569)(0.7569(1-0.7569))(0.5945)$$

$$= 0.0817$$

$$w_5(new) = w_5(old) - \eta g(n)$$

$$= 0.40 - 0.05 \times 0.0817 = 0.3591$$

Updating w_7

$$w_7(new) = w_7(old) - \eta g(n)$$

$$g(n) = \frac{\partial E_{total}}{\partial w_7}$$

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial e_{o1}} \times \frac{\partial e_{o1}}{\partial y_{o1}} \times \frac{\partial y_{o1}}{\partial v_{o1}} \times \frac{\partial v_{o1}}{\partial w_7}$$

$$= e_{o1} \times (-1) \times (y_{o1} \times (1-y_{o1})) \times y_{h2}$$

$$= -(d_{o1}-y_{o1}) y_{o1}(1-y_{o1}) y_{h2}$$

$$= -(0.01-0.7569) \times 0.7569 \times (1-0.7569) \times 0.59612$$

$$= 0.0819$$

$$_1(new) = 0.50 - 0.5 \times 0.0819$$

$$= 0.4590$$

At O_2 :

$$w_6(new) = w_6(old) - \eta g(n)$$

$$g(n) = \frac{\partial E_{total}}{\partial w_6}$$

$$\frac{\partial E_{total}}{\partial w_6} = \frac{\partial E_{total}}{\partial e_{o_2}} \times \frac{\partial e_{o_2}}{\partial y_{o_2}} \times \frac{\partial y_{o_2}}{\partial v_{o_2}} \times \frac{\partial v_{o_2}}{\partial w_6}$$

$$= e_{o_2} \times (-1) \times y_{o_2}(1 - y_{o_2}) \times y_{h_1}$$

$$= (d_{o_2} - y_{o_2})(-1)\, y_{o_2}(1 - y_{o_2}) \times y_{h_1}$$

$$= -(0.99 - 0.767)(0.767)(1 - 0.767)(0.5945)$$

$$= -0.0235$$

$$w_6(new) = 0.45 - (0.5)(0.0235)$$

$$= 0.45 + 0.011175$$

$$= 0.46175$$

$$w_8(new) = w_8(old) - \eta g(n)$$

$$g(n) = \frac{\partial E_{total}}{\partial w_8}$$

$$\frac{\partial E_{total}}{\partial w_8} = \frac{\partial E_{total}}{\partial e_{o_2}} \times \frac{\partial e_{o_2}}{\partial y_{o_2}} \times \frac{\partial y_{o_2}}{\partial v_{o_2}} \times \frac{\partial v_{o_2}}{\partial w_8}$$

$$= -(d_{o_2} - y_{o_2})\, y_{o_2}(1 - y_{o_2})\, y_{h_2}$$

$$= 0.0234$$

$$w_8(new) = 0.55 - (0.5)(-0.0234)$$

$$= 0.5617$$

At hidden layer

at h_1

Updation of w_1

$$w_1(new) = w_1(old) - \eta g(n)$$

$$g(n) = \frac{\partial \varepsilon_{total}}{\partial w_1}$$

$$\frac{\partial \varepsilon_{total}}{\partial w_1} = \frac{\partial \varepsilon_{total}}{\partial y_{h1}} \times \frac{\partial y_{h1}}{\partial v_{h1}} \times \frac{\partial v_{h1}}{\partial w_1}$$

$$\frac{\partial \varepsilon_{total}}{\partial y_{h1}} = \frac{\partial \varepsilon_{o1}}{\partial y_{h1}} + \frac{\partial \varepsilon_{o2}}{\partial y_{h1}}$$

$$\frac{\partial \varepsilon_{o1}}{\partial y_{h1}} = \frac{\partial \varepsilon_{o1}}{\partial c_{o1}} * \frac{\partial c_{o1}}{\partial y_{o1}} * \frac{\partial y_{o1}}{\partial v_{o1}} * \frac{\partial v_{o1}}{\partial y_{h1}}$$

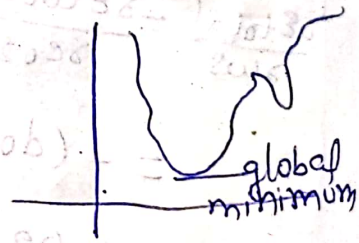
$$\frac{\partial \varepsilon_{o2}}{\partial y_{h1}} = \frac{\partial \varepsilon_{o2}}{\partial c_{o2}} * \frac{\partial c_{o2}}{\partial y_{o2}} * \frac{\partial y_{o2}}{\partial v_{o2}} * \frac{\partial v_{o2}}{\partial y_{h1}}$$

Continuation in papers

* Practical & design issues of Back propagation algorithm :-

① Rate of learning
- specifies a step size taken by learning algorithm
- takes a value in between 0-1

$0 \le \eta \le 1$



η is very small

$$w(new) = w(old) - \eta g(n)$$

↓ learning rate

slowly converges to global minimum.

It increase no. of iterations required for converging.

- algorithm can trapped into local minimum point.
- n very large
- algorithm can overshoot global minimum point
- can oscillate
- avoids getting trap into local minima point.

Solution

momentum:-

- $\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \Delta w_{ji}(n)$
- provides solution to problems associated with learning rate.
- Adds small amount of update from previous iteration controlled by momentum term(α)

Sequential or Batch mode:-

epoch

↓

presentation of complete set of examples or samples

- also called online mode.
- update for weight vector computed sequentially by considering only one sample at a time.
- It requires less memory space as it slowly converges towards global minima point.

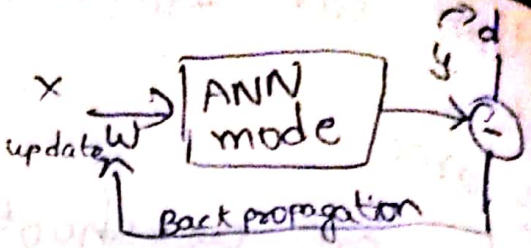
drawback

- It is theoritically difficult to determine stopping criteria.

1.10.22

② Stopping Criteria

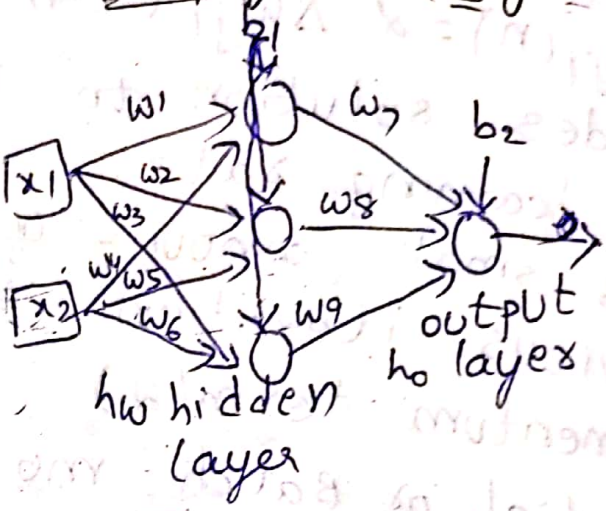
- When model produces error that is accepted.
- Euclidean norm of y-d reaches to minimum value.



③ New sample not there in training set.
- Generalization. (By creating power set)

Implementation of Backpropagation algorithm:

$$h_w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix}_{6\times1} \qquad h_o = \begin{bmatrix} w_7 \\ w_8 \\ w_9 \end{bmatrix}_{3\times1}$$



$v_{h1} = x_1 w_1 + x_2 w_4$

$v_{h2} = x_1 w_2 + x_2 w_5$

$v_{h3} = x_1 w_3 + x_2 w_6$

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad h_w = \begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_6 \end{bmatrix}$$

11/10/22

Design Issues in BackPropagation Algorithm -

1) learning rate (η) - due to this algorithm converges slowly or at fast rate

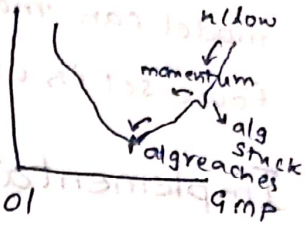
low

→ $\Delta w(new) = w(old) - \eta g(n)$

* alg struck in local min point
* momentum is applied to accelerates alg and alg move forward from local min point and to reach global min point

then update formula changes for ★ (i.e., update value of previous iteration is added to update)

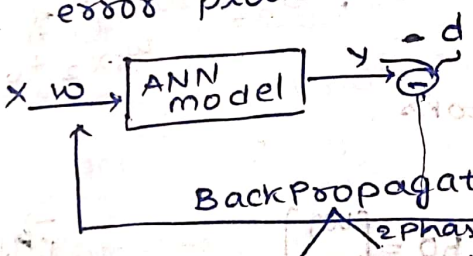
$\Delta w = \Delta w(n) + \alpha \Delta w(n-1) - \eta g(n)$



★ α - momentum term
- lies b/w 0 to 1
- mostly $\alpha = 0.001$ or 0.01
- α is a multiplied with $\Delta w(n-1)$ because if $\Delta w(n-1)$ is too high then alg takes fewer & faster steps to reach GMP. So, smallest value α is multiplied with $\Delta w(n-1)$.

2) Stopping Criteria -

BackPropagation is a learning/optimization Alg used to reduce error produced by AN.



d - y = e → simple perception — Gradient Descent Alg
↓ multiple layers

goes to iteration process.

goes to feed forward phase (read i/p, produces o/p & identifies error)

This process occurs iteratively.

If no stopping criteria exists, then Back Propagation occurs & iterates infinitely. infinite loop i.e., BP when y=d - but leads to sto

General stopping Criteria used is to

Alg is stopped when error value reaches a Particular threshold value. (98% accepted - stopping criteria = 5%

95% accurate.

when euclidean norm of d & y reaches to a min. value.

correct values should predicted)

3) new sample not there in training set.

- Backpropagation goes through Generalization

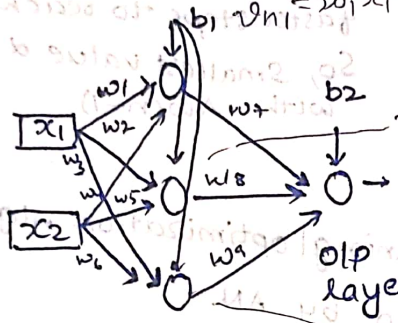
Process (It tries to match (compare) data with already trained (Previous sample) data and make conclusions (given sample)

*Powerset is created for the training data using which model is trained. Powerset contains all features of dataset so that based on it, model can make predictions for new data to. Powerset is used in Generalisations.

Implementation of Backpropagation Algorithm

Before implementation following tasks must be done-

① intialize freee parameters (w, b, η) # random package randint method in Python.



① i) construct weight vectors

$$hw = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} \qquad ho = \begin{bmatrix} w_7 \\ w_8 \\ w_9 \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

② Construct neural network models.

→ no. of inputs

→ no. of hidden layers

- no. of neurons in each h.l

→ no. of neurons in olp layer.

③ Compute error value

$e = d - y$

Update the weights.

$$\begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

wt matrix size: no.of neurons in h.l × no.of i/p's

i/p vector/matrix → no.of i/p's

$$w = \begin{bmatrix} w_7 \\ w_8 \\ w_9 \end{bmatrix}_{3\times 1} \qquad \begin{bmatrix} y_{h_1} \\ y_{h_2} \\ y_{h_3} \end{bmatrix}$$

$[w_7\ w_8\ w_9]_{3\times 1}$

Program:

Initializing Parameters:

```
import numpy as np
inputSize = 2
hiddensize = 3
outputsize = 1
lr = 0.1
w1 = np.random.randn(inputsize, hiddensize)
w2 = np.random.randn(hiddensize, outputsize)
b1 = np.random.randn(hiddensize,1)*0.01
b2 = np.random.randn(outputsize,1)*0.01
```

define activation fn:

```
def sigma(v):
    y = 1/(1+np.exp(-v))
    return y.
def derivate(y):
    return (y*(1-y))
def msqerror(yP,y):
    E = ((yp - y)**2).sum()/2
    return E
```

define a dataset

x = np.array([[7,2],[2,6],[9,1]])

y = np.array([[80],[50],[95]])

hrs studied	hrs enjoyed	result
7	2	80
2	6	50
9	1	95
x		y

normalise

X = x/np.amax(x, axis=0)

y = y/100

create a neural n/w and train

def train(x,y):

global w1, w2, b1, b2, lr

forward phase

vh = np.dot(x, w1) + b1 (or) x@w1

yh = sigm(vh)

vo = np.dot(yh, w2) + b2 (or) (yh@w2) + b2

yo = sigm(vo)

~~return yo~~ ✓

~~train(X,y)~~ ✓

$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial e_0} \cdot \frac{\partial e_0}{\partial y_0} \cdot \frac{\partial y_0}{\partial E} \cdot \frac{\partial v_0}{\partial w_2}$

Back Propagation phase:

find error value (d-y):

e1 = y - yo

delta_0 = e1 * yo * (1 - yo)

e2 = delta0 @ w2.T

delta_h = e2 * yh * (1 - yh)

g0 = yh.T @ delta_0

final gradient

```
gh = x.T@delta_h
w1 = w1 - lr*gh
w2 = w2 - lr*go
b1 = b1 - lr*delta_h
b2 = b2 - lr*delta_o
```

#updating weights & biases

```
for i in range(2000):
    train(x,y)

def forwardtest(x,y):
    global w1,w2,b1,b2
    #forward phase
    vh = np.dot(x,w1)+b1  (or) (x@w1)+b1
    yh = sigm(vh)
    vo = np.dot(yh,w2)+b2  (or) (yh@w2)+b2
    yo = sigm(vo)
    return yo

forwardtest(x,y)
```

UNIT-V Regression using MLP

- independent variables - i/p variables
- dependent variables - o/p variables.

Regression -

- It is a technique used to determine relationship b/w independent and dependent variables.
- 3 techniques -

① Simple linear regression:

- determines relationship between independent & dependent variables.

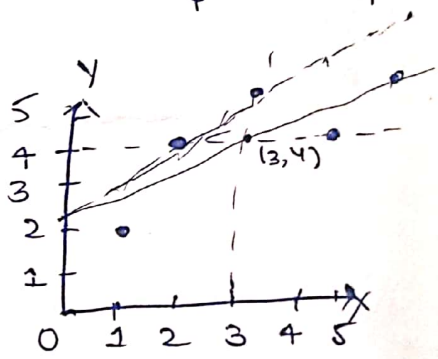
$$y = B_0 + B_1 x$$

x - independent variable
y - dependent variable
B_0 - y intercept
B_1 - slope

Eg:

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

$\bar{x} = 3$ $\bar{y} = 4$



$$B_1 = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sum (x-\bar{x})^2} = \frac{6}{10} = 0.6$$

$$\bar{y} = B_0 + B_1 * \bar{x}$$

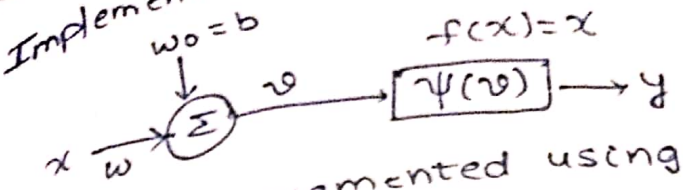
$$4 = B_0 + 0.6 * 3$$

$$B_0 = 4 - 0.6 \times 3$$

$$= 4 - 1.8$$

$$B_0 = 2.2$$

Implementation using perceptron -



- can be implemented using single layer perceptron with linear activation fn

$$y = wx + w_0$$

$$y(v) = wx + w_0$$