V. IMP

Gradient Descent for Neural Networks

This is the heart of supervised learning.

x	y
2	4
3	6
5	10
7	14
9	18

The relationship we may see is that $y = 2*x$.

Similarly, computers use Technique called Gradient Descent to find, $y = x * w_1 - w_2$

w_1 = weight

w_2 = bias.

*) Gradient descent is used in the training of a Neural Network.

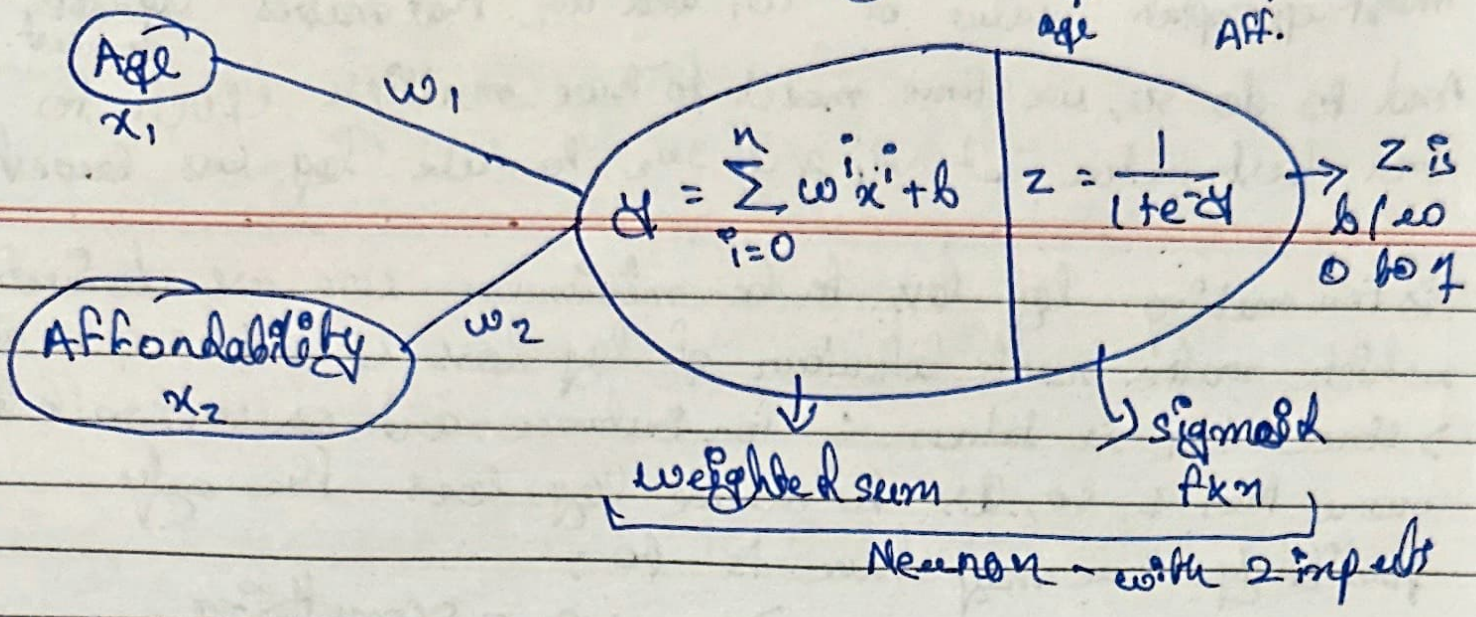
So understanding of Gradient Descent better with a Binary classification :-

age	affordability	have_insurance
23	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
:	:	:
	x	y

based on age and affordability come up with a fxn that can predict if person will buy insurance or not.

Neural Diagram :-

$y = w_1 * x_1 + w_2 * x_2 + bias$ (Age, Aff.)



→ Since we are terminatingly / finally going to make a prediction model hence Gradient Descent is a technique to find patterns existing sample training data & then make a prediction model.

step-1 :- input the training sample data with random w_1 and w_2 value let here be $w_1 = 1, w_2 = 1$ (bias = 0.)

Hence, Σ weighted sum $= y = w_1 * x_1 + w_2 * x_2 + bias$

Putting age = 22, Aff. = .1, $y = 1 * 22 + 1 * 1 + 0 = 23$

z or $\hat{y} = 0.99$ (predicted value) But have_insurance = 0 (Truth value.)

This there is error :-

In Gradient Descent (Logistic Regression) we consider log-loss, to measure loss

Here, error 1 $= -(y \log(\hat{y}) + (1-y) \log(1-\hat{y})) = 4.6$

↳ we find individual errors then total error

Repeating for further samples :- error 2 = 4.6

⋮

error 13 = 0.01

Total error = error 1 + error 2 + ... + error 13

(Then

Log loss on binary cross entropy $= -\frac{1}{n} \sum_{i=0}^{n} y_i \log(\hat{y}_i) + (1-y_i) \cdot \log(1-\hat{y}_i)$

Here we get log loss as : 4.31

Note -> our main goal is to reduce log-loss and to find the most appropriate value of w_1 and w_2. that makes log-loss lowest.

And to do so, we force model to have multiple epoch to find best value of w_1 and w_2 to make log-loss lowest.

→ For making log-loss to be minimum we use derivative which make mark behaviour of log loss w.r.t. w_1 or w_2.

→ since w_1 is taken 1 for instance and so w_1 can't be more than 1 so, as to reduce log loss the only possibility we may have is to :-

$w_1 = w_1 - \text{something}$.

→ $w_1 = w_1 - \text{learning rate} * \frac{\partial}{\partial \cdot w_1}$ or $\frac{d(\text{learning rate})}{d(w_1)}$

→ $w_1 = w_1 - \frac{d(\text{learning rate})}{d(w_1)}$

$w_1 = w_1 - \text{something}$

$= w_1 - \text{learning rate} * \frac{d(\text{loss})}{d(w_1)}$

(usually = 0.01)

similarly $w_2 = w_2 - \text{learning rate} * \frac{d(\text{loss})}{d(w_2)}$

$b = b - \text{learning rate} * \frac{d(\text{loss})}{d(b)}$

Now :- $\frac{d(\text{loss})}{d(w_1)} = \frac{1}{n} \sum_{i=1}^{n} x_i (\hat{y}_i - y_i)$ } same for w_2.

But for Bias :- $\frac{d(\text{loss})}{d(\text{bias})} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)$

Through this we get real new values :- $w_1 = 0.8$, $w_2 = 0.7$, bias $= -0.2$

Initially values were :- $w_1 = 1$, $w_2 = 1$, bias $= 0$

→ Here we are using Batch Gradient Descent.
Now, we will find individual errors then find loss
Remember our first epoch has given loss = 4.31
Now, using new w_1 and w_2 and bias we calculate new loss = that may be something } second epoch

→ Our main goal is to minimize loss and find appropriate value of w_1, w_2 and bias.

Using TensorFlow keras to find best value the code goes like this :-

```
model = keras.Sequential([
    keras.layers.Dense(1, input_shape(2,), activation='sigmoid'
, kernel_initializer='ones', bias_initializer='zeros')
])
    model.compile(optimizer='adam'
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
model.fit(X_train_scaled, y_train, epochs=5000)
```

we initialized w_1, w_2 as kernel_initializer → 'ones'
and " bias as bias_initializer → 'zeros'.
& log loss = binary_crossentropy.

→ After training the model with TF keras we may want to know values of w_1, w_2 and bias so to do so, →

```
coef, intercept = model.get_weights()
```

we got :- $w_1 = 5.060867$
$w_2 = 1.4086$
bias $= -2.9137$

Making Neural Network without Tensorflow:-

```
def sigmoid(x):
    import math
    return 1/(1+math.exp(-x))

def prediction_fxn(age, affordability):
    weighted_sum = coef[0]*age + coef[1]*affordability
                   + intercept
    return sigmoid(weighted_sum)
```

We successfully made a single Neuron.

Gradient Descent function in python from scratch:-

```
def log_loss(y_true, y_predicted):
    epsilon = 1e-15
    y_pred_new = [max(i, epsilon) for i in y_pred]
    y_pred_new = [min(i, 1-epsilon) for i in y_pred_new]
    y_pred_new = np.array(y_pred_new)
    return -np.mean(y_true*np.log(y_pred_new)
                    + (1-y_true)*np.log(1-y_pred_new))
```

```
def sigmoid_numpy(X):
    return 1/(1+np.exp(-X))
```

sigmoid_numpy(np.array([12,0,1])) ⟶ This gives the computation of sigmoid in array form.

def gradient_descent (age, affordability, y_true, epochs, loss_threshold):

here we will be getting w_1, w_2, bias.

$w_1 = w_2 = 1$

bias = 0.

rate = 0.5

n = len(age)

for i in range(epochs):

weighted fxn

$w_1 * age + w_2 * aff. + bias = weighted\_sum.$

$y\_pred = sigmoid\_numpy(weighted\_sum).$

sigmoid fxn

$loss = log\_loss(y\_true, y\_pred).$

$w1\_derivative = \left(\frac{1}{n}\right) * np.dot(np.transpose(age), (y\_pred - y\_true)).$

This is same as :- $\frac{d(loss)}{d(w_1)} = \frac{1}{n}\sum_{i=1}^{n} x_i(\hat{y}_i - y_i)$

$w2\_derivative = \left(\frac{1}{n}\right) * np.dot(np.transpose(affordability), (y\_pred - y\_true))).$

$bias\_derivative = np.mean(y\_pred - y\_true)$

This is same as :- $\frac{d(loss)}{d(bias)} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)$

$w_1 = w_1 - (rate * w1\_derivative)$

$w_2 = w_2 - (rate * w2\_derivative)$

$bias = bias - rate * bias\_derivative$

if(loss < loss_threshold) break

return w_1, w_2, bias

---

gradient_descent(X_train_scaled['age'], X_train_scaled['affo...'], y_train, 1000, 0.4631).