Logistic Regression with Gradient Descent

Logistic Regression with Gradient Descent (From Scratch)

This document explains logistic regression and gradient descent **from first principles**, with full mathematical derivations.

Problem Setup

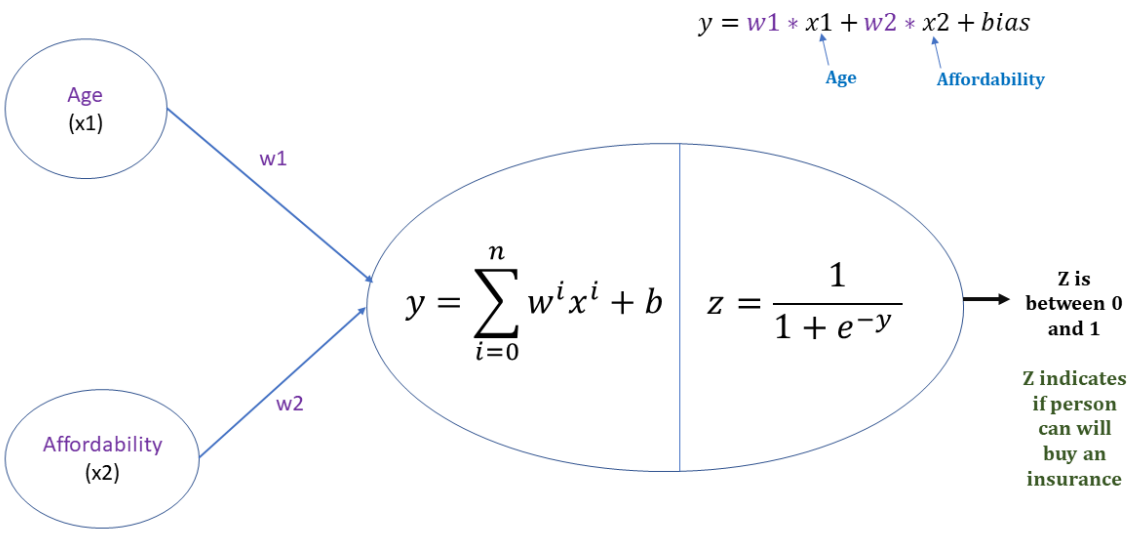

Figure 1: Neural Network Diagram

Source: https://github.com/codebasics/deep-learning-keras-tf-tutorial/blob/master/6_gradient_descent/nn.png

We solve a **binary classification** problem.

Inputs: - Age: x_1 - Affordibility: x_2

Output:

$$y \in \{0, 1\}$$

Model Definition

Linear Combination

$$z_i = w_1 x_{1,i} + w_2 x_{2,i} + b$$

Sigmoid Activation

$$\hat{y}_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}$$

Binary Cross-Entropy Loss

For one sample:

$$\ell_i = -\left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right]$$

For all samples:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \ell_i$$

Gradient Descent

Parameters are updated using:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}$$

Gradient Derivation for w_1

Using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \ell_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_1}$$

Individual Derivatives

Loss derivative:

$$\frac{\partial \ell_i}{\partial \hat{y}_i} = -\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right)$$

Sigmoid derivative:

$$\frac{\partial \hat{y}_i}{\partial z_i} = \hat{y}_i (1 - \hat{y}_i)$$

Linear term:

$$\frac{\partial z_i}{\partial w_1} = x_{1,i}$$

Simplification

The expression simplifies to:

$$\hat{y}_i - y_i$$

Final Gradient

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_{1,i}$$

Other Gradients

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_{2,i}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

Vectorized Form

$$\nabla_w \mathcal{L} = \frac{1}{n} X^T (\hat{y} - y)$$

Key Takeaways

- Logistic regression is a linear classifier with a nonlinear activation
- Sigmoid + cross-entropy leads to clean gradients
- Gradient descent minimizes loss iteratively
- This mirrors internal behavior of ML frameworks

```
import math
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras

from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split

df = pd.read_csv('/home/mpaul/projects/mpaul/mai_prep/interview_preparation/topics/deep_learni
print(df.head())
```

```
X = df[['age', 'affordibility']]
y = df['bought_insurance']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

X_train['age'] = X_train['age'] / 100
X_test['age'] = X_test['age'] / 100

_model= False
if _model:
    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.summary()
    model.fit(X_train, y_train, epochs=100)
    model.evaluate(X_test, y_test)

    print(model.predict(X_test))
    print(y_test)

    coeff, bias = model.get_weights()
    print(coeff)
    print(bias)


# ========================= User Defined Prediction Function ========================
def prediction_function(age, affordibility):
    weighted_sum = coef[0]*age + coef[1]*affordibility + intercept
    z = sigmoid(weighted_sum)
    return z

# print(prediction_function(X_test['age'], X_test['affordibility']))

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def log_loss(y_true, y_predicted):
    epsilon = 1e-15
```

```
    y_predicted_new = [max(i,epsilon) for i in y_predicted]
    y_predicted_new = [min(i,1-epsilon) for i in y_predicted_new]
    y_predicted_new = np.array(y_predicted_new)
    loss = -np.mean(y_true*np.log(y_predicted_new)+(1-y_true)*np.log(1-y_predicted_new))
    return loss

def gradient_descent(age, affordibility, y_true, epochs, loss_threshold):
    w1 = w2 = 1
    bias = 0
    rate = 0.5
    n = len(age)

    for i in range(epochs):
        weighted_sum = w1 * age + w2 * affordibility + bias
        y_predicted = sigmoid(weighted_sum)
        loss = log_loss(y_true, y_predicted)
        print(loss)
        w1d = (1/n)*np.dot(np.transpose(age),(y_predicted - y_true))
        w2d = (1/n)*np.dot(np.transpose(affordibility),(y_predicted - y_true))
        bias_d = np.mean(y_predicted-y_true)
        w1 = w1 - rate * w1d
        w2 = w2 - rate * w2d
        bias = bias - rate * bias_d
        print (f'Epoch:{i}, w1:{w1}, w2:{w2}, bias:{bias}, loss:{loss}')

        if loss <= loss_threshold:
            print (f'Epoch:{i}, w1:{w1}, w2:{w2}, bias:{bias}, loss:{loss}')
            break
    return w1, w2, bias

gradient_descent(X_train['age'],X_train['affordibility'],y_train,1000, 0.4631)
```