3 Cost Function and optimization algorithms in Machine Learning

Basic concepts

Chapter outline

1. Cost Function in Machine Learning
2. Optimization algorithms
    1. Gradient Descent and its application to the linear regression
    2. Stochastic Gradient Descent

1. Cost function in Machine Learning

A **cost function** determines the **performance** of a machine learning model for a given data set by a real number.

It computes the **difference or distance** between actual output and predicted output.

The **main objective** of a ML model is to determine the parameters or weights that can **minimize** the **cost function**.

*"**Cost function** determines the performance of a Machine Learning Model using a single real number, known as **cost value/model error.** This value depicts the average error between the actual and predicted outputs."*

1. Cost function in Machine Learning

There are many functions used for different purposes:

- Regression
    - MAE (Mean Absolute Error)
    - MSE (Mean Squared Error)
    - Hubber loss
- Classification
    - Binary cross-entropy
    - Categorical cross-entropy

1. Cost function in Machine Learning

1.1. Mean Absolute Error/ L1 loss

$$mae = \frac{1}{N} \cdot \sum_{1}^{N} |actual - pred|$$

1.2. Mean Squared Error / Squared loss / L2 loss

$$mse = \frac{1}{N} \cdot \sum_{1}^{N} (actual - pred)^2$$

1.3. Huber Loss

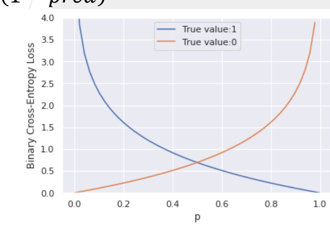
$$huber = \frac{1}{N} \cdot \sum_{1}^{N} \frac{1}{2} \cdot (actual - pred)^2$$

1. Cost function in Machine Learning

1.4. Binary Cross Entropy/log loss

$\log loss = -\frac{1}{N} \cdot \sum_1^N actual \cdot log(pred) + (1 - actual) \cdot log(1 - pred)$

$\log loss \geq 0$



1.5. Categorical Cross Entropy

It is used for Multiclass classification and SoftMax regression.

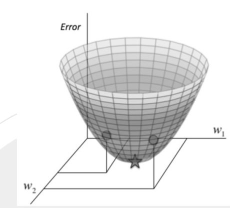
$$loss = -\sum_{1}^{N} actual \cdot log(pred) \geq 0$$

2. Optimization algorithms

To **optimize** a machine learning model, we check the results at each iteration by **changing** the **hyperparameters** at each step until we reach the **optimal** results.

There are several ways to optimize a model. In this section we will discuss two important optimization algorithms: **Gradient Descent (GD)** and **Stochastic Gradient Descent (SGD)** algorithms.

2. Optimization algorithms



2.1. Gradient Descent

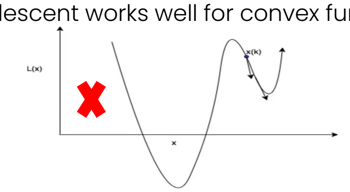
This optimization algorithm uses **computation** to **modify** the values in a coherent way and reach the **local minimum**.

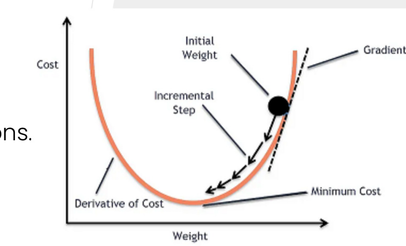
$w_{new} = w - \eta \cdot loss'(w)$

η: learning rate

$loss'(w)$ the derivative (gradient) of $loss(w)$

Gradient descent works well for convex functions.

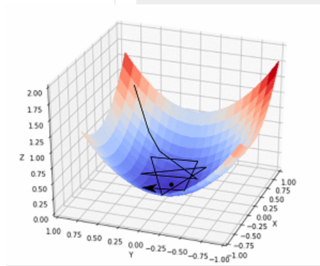




2. Optimization algorithms (next)

2.2. Stochastic Gradient Descent (SGD)

In this algorithm, rather than using the entire data set at each iteration, a **single random training example** (or a small batch) is selected to calculate the gradient and update the model parameters.



2. Optimization algorithms (next)

Exp.

Let's use a linear regression approach, with Mean Squared Error function (mse) as loss function.

The linear relationship between X and Y: $Y = a \cdot X + b$

$$mse = \frac{1}{N} \cdot \sum_1^N \left(Y_{actual} - Y_{pred}\right)^2$$

$$mse = \frac{1}{N} \cdot \sum_1^N \left(Y_{actual} - (a \cdot x_i + b)\right)^2$$

$$\frac{\partial mse}{\partial a} = \frac{-2}{N} \sum_1^N x_i \cdot \left(Y_{actual} - Y_{pred}\right)$$

in the same way we calculate:

$$\frac{\partial mse}{\partial b} = \frac{-2}{N} \sum_1^N \left(Y_{actual} - Y_{pred}\right)$$

2. Optimization algorithms (next)

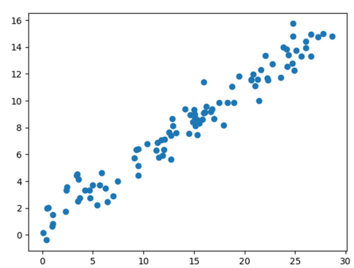
Exp. (next)

To update parameters:

$$a_{new} = a - lr * \frac{\partial mse}{\partial a}$$

$$b_{new} = b - lr * \frac{\partial mse}{\partial b}$$

```
# Generate Input data
num_points=100
X = 30 * np.random.random((num_points))
Y = 0.5 * X + 1.0 + np.random.normal(size=X.shape)
plt.scatter(X, Y)
plt.show()
```



2. Optimization algorithms (next)

```
a, b, lr = 0, 0, 0.0001  # a, b, learning Rate
epochs = 1000  # The number of iterations to perform gradient descent
N = float(len(X))  # Number of elements in X
# Performing Gradient Descent
for i in range(epochs):
    Y_pred = a * X + b
    dmse_da = (-2 / N) * sum(X * (Y - Y_pred))  # Derivative wrt a
    dmse_db = (-2 / N) * sum(Y - Y_pred)  # Derivative wrt b
    a = a - lr * dmse_da  # Update a
    b = b - lr * dmse_db  # Update b
print(a, b)
Y_pred = a * X + b
plt.scatter(X, Y)
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red')
plt.show()
```

Result:
[0.5508771] [0.0722681]