Experiment No. 9

Title - Implement an Optimization Algorithm

Theory -

Optimization in machine learning refers to the process of finding the best set of parameters (weights and biases) for a model that minimizes or maximizes a specific objective function. The goal is to improve the performance of the machine learning model by adjusting its parameters based on the training data. Optimization plays a crucial role in training models across various machine learning algorithms, and it is essential for achieving better accuracy and generalization on unseen data.

1. **Objective Function (Cost Function or Loss Function):**

- The objective function represents the performance of the machine learning model.
- In supervised learning, it measures the difference between the predicted outputs and the actual outputs (labels) for a given set of input data.
- The goal is to minimize this function for regression tasks (e.g., mean squared error) or classification tasks (e.g., cross-entropy loss).

2. **Parameters:**

- Parameters are the internal variables of a machine learning model that are adjusted during the training process to minimize the objective function.
- In a neural network, parameters include weights and biases for each layer.

3. **Optimization Algorithm:**

- Optimization algorithms determine how the model's parameters are updated during training to minimize the objective function.

- Common optimization algorithms include Gradient Descent, Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent, Adam, RMSprop, and more.
- These algorithms differ in terms of how they use gradients and update parameters.

4. Gradient Descent:

- Gradient Descent is a widely used optimization algorithm.
- It iteratively moves towards the minimum of the objective function by adjusting the parameters in the opposite direction of the gradient.
- The gradient represents the direction of the steepest increase in the objective function.

5. Learning Rate:

- The learning rate is a hyperparameter that controls the step size in the parameter space during optimization.
- A too small learning rate may result in slow convergence, while a too large learning rate may cause overshooting and divergence.
- It needs to be carefully tuned for optimal performance.

6. Batch, Stochastic, and Mini-Batch Gradient Descent:

- Batch Gradient Descent computes the gradient of the entire training dataset to update the parameters in each iteration.
- Stochastic Gradient Descent computes the gradient for each training example, updating the parameters one example at a time.
- Mini-Batch Gradient Descent combines aspects of both batch and stochastic by updating parameters using a small random subset (mini-batch) of the training data.

7. **Convergence and Stopping Criteria:**

- Convergence is reached when the optimization algorithm has sufficiently minimized the objective function.
- Stopping criteria, such as a maximum number of iterations or a threshold for the change in the objective function, are defined to avoid unnecessary computation.

8. **Regularization:**

- Regularization techniques, such as L1 and L2 regularization, can be applied to prevent overfitting and improve the generalization of the model.
- They add penalty terms to the objective function to discourage the model from fitting noise in the training data.

Optimization in machine learning is an iterative process, and the choice of optimization algorithm, learning rate, and other hyperparameters can significantly impact the performance of the model. It requires a balance between computational efficiency and the ability to find the global minimum of the objective function in the parameter space.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an optimization algorithm commonly used to train machine learning models, particularly in the context of large datasets. It is a variant of the gradient descent algorithm, but instead of computing the gradient of the entire dataset at each iteration, it randomly selects a single data point (or a small subset, often called a mini-batch) to update the model parameters. This randomness introduces noise into the optimization process but can significantly speed up the convergence of the algorithm.

How SGD works ?

1. Initialization:

Initialize the model parameters (weights and biases) with small random values. Choose a learning rate α, which determines the size of the steps taken during parameter updates.

2. Iterative Optimization:

Repeat the following steps until convergence or for a fixed number of iterations (epochs):

- Shuffle the dataset to randomize the order of data points.
- For each data point (or mini-batch) in the dataset:
- Compute the gradient of the cost function with respect to the model parameters using the current data point.
- Update the model parameters using the computed gradient and the learning rate.

The update step for the j-th parameter θ_j in a linear regression setting is given by:

$\theta_j := \theta_j - \alpha \cdot \text{gradient}$

where gradient is the gradient of the cost function with respect to θ_j.

3. Convergence:

Monitor the decrease in the cost function or other convergence criteria to decide when to stop the iterations.

The formula for updating the parameters (weights) in Stochastic Gradient Descent (SGD) for linear regression is derived from the gradient of the mean squared error with respect to the parameters. The mean squared error (MSE) for linear regression is given by:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Where:

- m is the number of training examples.
- θ is the parameter vector (weights).
- $h_\theta(x^{(i)})$ is the predicted value for the i-th example.

The gradient of the mean squared error with respect to the parameters is:

$$\nabla J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

In the case of SGD, we update the parameters for each training example individually. The update rule for the j-th parameter θ_j is given by:

$$\theta_j := \theta_j - \alpha \cdot (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Where:

- α is the learning rate.
- $x_j^{(i)}$ is the j-th feature of the i-th training example.

This update is performed for each training example, and the process is repeated for multiple epochs until convergence. The idea is to iteratively adjust the parameters in the direction that reduces the error.

Implementation :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Read data from csv file
data = pd.read_csv('SAT-GPA.csv')
data.head()
#remove null values
data=data.dropna()

#check if there are any null values
data.isnull().sum()
```

```
>       SAT   0
        GPA   0
        dtype: int64
```

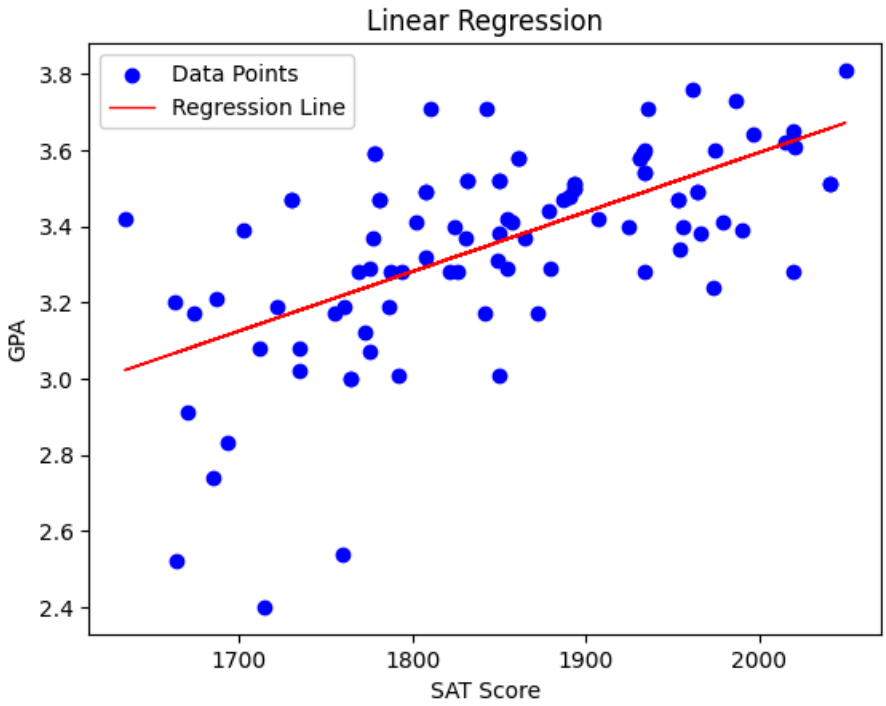
```
#prepare x and y
x=data['SAT'].values
y=data['GPA'].values

#apply linear regression
from sklearn.linear_model import LinearRegression

reg=LinearRegression()
x=x.reshape(-1,1)
reg.fit(x,y)
print('Coefficients:', reg.coef_)
print('Intercept:', reg.intercept_)
```

```
>       Coefficients: [0.00156111]
        Intercept: 0.47126246455052323
```

```
#plot the regression line
plt.scatter(x,y,color='blue',label='Data Points')
plt.plot(x,reg.predict(x),color='red',linewidth=1, label='Regression Line')
plt.title('Linear Regression')
plt.xlabel('SAT Score')
plt.ylabel('GPA')
plt.legend()
plt.show()
```



Apply Stochastic Gradient Descent

```
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# Feature scaling for SGD
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Apply stochastic gradient descent (SGD) for linear regression
sgd_reg = SGDRegressor(max_iter=1000, eta0=0.01, random_state=42)
sgd_reg.fit(x_train_scaled, y_train)

# Print the coefficients and intercept
print('Coefficients:', sgd_reg.coef_)
print('Intercept:', sgd_reg.intercept_)
```

```
# Plot the regression line
plt.scatter(x_train, y_train, color='black', label='Training Data')
plt.scatter(x_test, y_test, color='orange', label='Test Data')
plt.plot(x_train, sgd_reg.predict(x_train_scaled), color='red', linewidth=1, label='SGD Regression Line')
plt.plot(x,reg.predict(x),color='green',linewidth=1, label='Regression Line')
plt.title('Stochastic Gradient Descent Regression')
plt.xlabel('SAT Score')
plt.ylabel('GPA')
plt.legend()
plt.show()
```

> Coefficients: [0.14903386]
Intercept: [3.30557202]