Mini-Batch Gradient Descent

- This balances the Batch and Stochastic GD
- You define a batch size - eg. 30
  - It gets updated when it goes through 30 rows
  - If there are 300 rows, **Mini batch Descent** it will update **10 times**
  - **Stochastics GD** would update → **300 times**
  - **Batch GD** would update → **1 Time**
- **Used in Deep Learning**
- In **LR** → We generally use **Stochastic GD**

Batch = Group of Rows

Batch size = 1 → Batch GD

Batch size = N → Stochastic GD

Mini-Batch Gradient Descent from scratch

```
import random

class MBGDRegressor:

    def __init__(self,batch_size,learning_rate=0.01,epochs=100):
```

```
    self.coef_ = None
    self.intercept_ = None
    self.lr = learning_rate
    self.epochs = epochs
    self.batch_size = batch_size

def fit(self,X_train,y_train):
    # init your coefs
    self.intercept_ = 0
    self.coef_ = np.ones(X_train.shape[1])

    for i in range(self.epochs):

        for j in range(int(X_train.shape[0]/self.batch_size)):

            idx = random.sample(range(X_train.shape[0]),self.batch_size)

            y_hat = np.dot(X_train[idx],self.coef_) + self.intercept_
            #print("Shape of y_hat",y_hat.shape)
            intercept_der = -2 * np.mean(y_train[idx] - y_hat)
            self.intercept_ = self.intercept_ - (self.lr * intercept_der)

            coef_der = -2 * np.dot((y_train[idx] - y_hat),X_train[idx])
            self.coef_ = self.coef_ - (self.lr * coef_der)

    print(self.intercept_,self.coef_)

def predict(self,X_test):
    return np.dot(X_test,self.coef_) + self.intercept_
```

Mini-Batch Gradient Descent using SKLearn

- We don’t have a batch_size parameter in SGDRegressor
- So, we have to use method

```
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

# Sample Data
np.random.seed(42)
X = np.random.rand(100, 1) * 10  # 100 samples, 1 feature
y = 5 * X.squeeze() + np.random.randn(100) * 2  # y = 5x + noise

# Splitting Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

# Scaling (SGD works better with standardized data)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

sgd = SGDRegressor(max_iter=1000, eta0=0.01, learning_rate='constant')

# Define batch size
batch_size = 10

# Implement Mini-Batch GD by manually looping
for _ in range(1000):  # Run for 1000 iterations
```

```
    for i in range(0, len(X_train_scaled), batch_size):
        X_mini = X_train_scaled[i:i+batch_size]  # Take mini-batch
        y_mini = y_train[i:i+batch_size]
        sgd.partial_fit(X_mini, y_mini)  # Train only on this batch
# Predictions
y_pred = sgd.predict(X_test_scaled)

# Print model parameters
print("Slope (m):", round(sgd.coef_[0], 4))
print("Intercept (b):", round(sgd.intercept_[0], 4))
```

Output:

Slope (m): 14.3558
Intercept (b): 23.3043

What This Code Does

This code trains a machine learning model using **mini-batch gradient descent**, a method where the model learns by looking at small chunks of data at a time (instead of all the data at once). Think of it like studying for a test by reviewing small sections of your notes repeatedly until you understand everything.

---

1. `for _ in range(1000):`

- **What it does**: Repeats the code inside this block **1,000 times**.
- **Analogy**: You're going to study your entire set of notes **1,000 times** to make sure you learn the material thoroughly.
- **Why?**: The model needs to see the data multiple times to improve its predictions.

---

2. `for i in range(0, len(X_train_scaled), batch_size):`

- **What it does**: Splits the training data (`X_train_scaled`) into **mini-batches**.

- `batch_size`: The size of each small chunk (e.g., 32 data points at a time).
- Example: If you have 1,000 data points and `batch_size=32`, this loop will create chunks of 32 data points each.

- **Analogy**: Instead of reading all your notes at once, you split them into smaller sections (like one chapter at a time).

---

3. `X_mini = X_train_scaled[i:i+batch_size]`

- **Starts** at the item at index `i`.
- **Goes up to, but does not include**, the item at index `i+batch_size`.

`y_mini = y_train[i:i+batch_size]`

- **What it does**:
  - `X_mini`: A small chunk of input data (features like house size, bedrooms, etc.).
  - `y_mini`: The corresponding correct answers (e.g., house prices).
- **Analogy**: You take a small section of your notes (e.g., pages 1–10) and the answers to the questions on those pages.

---

4. `sgd.partial_fit(X_mini, y_mini)`

- **What it does**: Updates the model (`sgd`) using the mini-batch of data.
- **Analogy**: After studying a small section of your notes, you take a mini-quiz to test your understanding. Based on your mistakes, you adjust how you study the next section.
- **Why `partial_fit`?**: This method lets the model learn incrementally (little by little) instead of all at once.

Key Concepts Simplified

1. **Epoch**: One full pass through **all the training data**.
   - The outer loop (`for _ in range(1000)`) runs 1,000 epochs.
2. **Mini-Batch**: A small subset of the training data.

- Example: If you have 1,000 data points and `batch_size=32`, you'll have 32 mini-batches per epoch.

3. **Stochastic Gradient Descent (SGD)**: The algorithm that updates the model's parameters using mini-batches.

Why Use Mini-Batches?

- **Efficiency**: It's faster to process small chunks of data than the entire dataset at once.
- **Memory-Friendly**: Works for large datasets that can't fit in your computer's memory.
- **Better Learning**: Small updates reduce the risk of getting stuck in bad solutions.

Real-World Analogy

Imagine you're learning to bake cookies:

1. You bake a small batch (**mini-batch**) of cookies, taste them, and adjust the recipe.
2. You repeat this process 1,000 times (**epochs**), improving the recipe each time.
3. Eventually, you master the perfect cookie recipe (**trained model**)!

Summary

This code teaches a model to make predictions by:

1. Repeating the learning process 1,000 times.
2. Breaking the data into small, manageable chunks.
3. Updating the model's "knowledge" with each chunk.