Gradient Descent

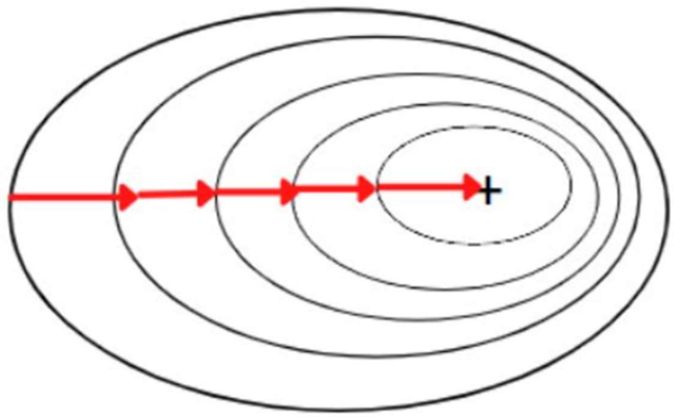
- Gradient Descent is an optimization algorithm that helps machine learning models converge at a minimum value through repeated steps. Essentially, gradient descent is used to minimize a function by finding the value that gives the lowest output of that function
- Often times, this function is usually a loss function. Loss functions measure how bad our model performs compared to actual occurrences. Hence, it only makes sense that we should reduce this loss. One way to do this is via Gradient Descent.
- There is a concept of learning schedule to smooth the fluctuation of stochastic GD to reach the solution.

A simple gradient Descent Algorithm is as follows:

- Obtain a function to minimize F(x)
- Initialize a value x from which to start the descent or optimization from
- Specify a learning rate that will determine how much of a step to descend by or how quickly you converge to the minimum value
- Obtain the derivative of that value x (the descent)
- Proceed to descend by the derivative of that value multiplied by the learning rate
- Update the value of x with the new value descended to
- Check your stop condition to see whether to stop
- If condition satisfied, stop. If not, proceed to step 4 with the new x value and keep repeating algorithm

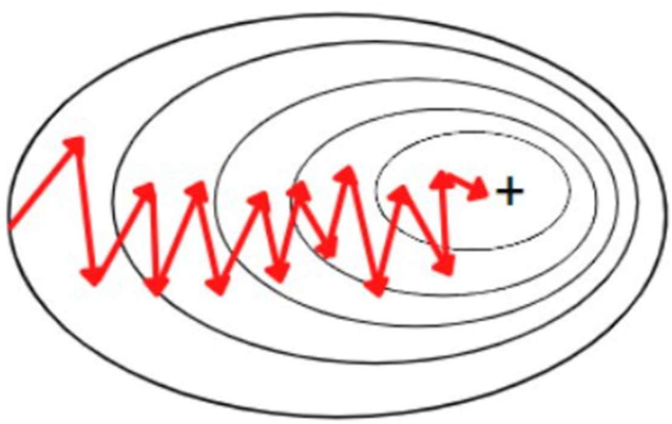
Types of Gradient Descent

1. **Batch Gradient Descent:**



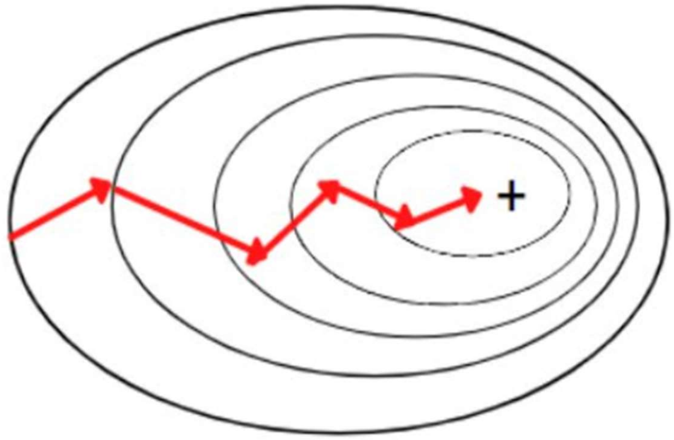
- **Description:** Uses the entire dataset to compute the gradient of the loss function for each iteration.
- **Pros:** Can converge smoothly to the minimum.
- **Cons:** Can be computationally expensive and slow for large datasets.

2. **Stochastic Gradient Descent (SGD):**



- **Description:** Uses a single data point (or a single example) to compute the gradient and update the parameters.
- **Pros:** Faster and more suitable for large datasets. Introduces more noise, which can help escape local minima.
- **Cons:** Convergence can be noisy and less stable. Requires careful tuning of the learning rate.

3. **Mini-Batch Gradient Descent:**



- **Description:** Uses a subset (mini-batch) of the data to compute the gradient. It combines the advantages of both batch and stochastic gradient descent.
- **Pros:** Balances the trade-offs between the stability of batch gradient descent and the efficiency of stochastic gradient descent.
- **Cons:** Requires choosing the size of the mini-batch, which can affect performance.

(1) Linear model

```
X,y = load_diabetes(return_X_y=True)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
reg = LinearRegression()
reg.fit(X_train,y_train)
```

```
print('intercepts :',reg.intercept_)
print('coef :',reg.coef_)
```

```
intercepts : 151.88331005254167
coef : [  -9.15865318 -205.45432163  516.69374454  340.61999905 -895.5520019
  561.22067904  153.89310954  126.73139688  861.12700152   52.42112238]
```

Compare all GD with linear model

(1) Batch GD

```python
#Batch Gradient Descent

class MyGDRegressor:
    def __init__(self,learning_rate=0.1,epochs=100):
        self.coef=None
        self.intercept=None
        self.lr=learning_rate
        self.e=epochs

    def fit(self,X,y):

        self.intercept=0
        self.coef=np.ones(X.shape[1])

        for i in range(self.e):
            y_hat=np.dot(X,self.coef)+self.intercept

            intercept_der=-2* np.mean(y-y_hat)
            self.intercept=self.intercept-(self.lr*intercept_der)

            coef_der=(-2*np.dot((y-y_hat),X))/(X.shape[0])
            self.coef=self.coef-(self.lr*coef_der)

        print('intercepts :',self.intercept,'\n','coef :',self.coef)
```

```python
gdr = MyGDRegressor(learning_rate=0.5,epochs=1000)
gdr.fit(X_train,y_train)
```

```
intercepts : 152.01351687661833
 coef : [  14.38990585 -173.7235727   491.54898524  323.91524824  -39.32648042
 -116.01061213 -194.04077415  103.38135565  451.63448787   97.57218278]
```

(2) Mini - batch

```python
# Mini-batch
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,y):
        self.intercept_ = 0
        self.coef_ = np.ones(X.shape[1])

        for i in range(self.epochs):
            for j in range(int(X.shape[0]/self.batch_size)):
                idx = random.sample(range(X.shape[0]),self.batch_size)
                y_hat = np.dot(X[idx],self.coef_) + self.intercept_
                #print("Shape of y_hat",y_hat.shape)
                intercept_der = -2 * np.mean(y[idx] - y_hat)
                self.intercept_ = self.intercept_ - (self.lr * intercept_der)

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

        print('intercepts :',self.intercept_,'\n','coef :',self.coef_)
```

```python
mbr = MBGDRegressor(batch_size=int(X_train.shape[0]/50),learning_rate=0.01,epochs=100)
mbr.fit(X_train,y_train)
```

```
intercepts : 152.89814913923837
 coef : [  28.42076017 -140.80852187  454.82844668  302.86621213  -24.62868749
  -91.36997095 -188.58317896  111.25902101  405.96116794  110.11405951]
```

(3) Stochastic

```python
# Stochastic gradient
class SGDRegressor:
    def __init__(self,learning_rate=0.01,epochs=100):

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

    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(X_train.shape[0]):
                idx = np.random.randint(0,X_train.shape[0])

                y_hat = np.dot(X_train[idx],self.coef_) + self.intercept_

                intercept_der = -2 * (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('intercepts :',self.intercept_,'\n','coef :',self.coef_)
```

```python
sgd = SGDRegressor(learning_rate=0.01,epochs=40)
sgd.fit(X_train,y_train)
```

```
intercepts : 151.38774327686912
 coef : [  69.87732552  -49.6438614   314.97645107  235.75450369   17.77718278
  -22.89424533 -162.76510881  125.67350101  290.26070524  137.4998188 ]
```