Лабораторийн ажил: Градиент буулгалтын арга

Хичээлийн тодорхойлолт:

Градиент буулгалтын алгоритм (Gradient Descent Algorithm) нь алдааны функцын хамгийн бага утгыг (**Global Minimum**) олж авах давталтын алгоритм юм. GD(Gradient Descent) алгоритмын төрлүүд нарийвчлал (accuracy), хугацааны төвөгшилт (time) зориулагдсан бөгөөд үүнийг доор дэлгэрэнгүй авч үзэх болно. Градиент буулгалтын эхний алхамд параметрүүдийг тодорхойлох, утга тодорхойлох (initialize) ба глобал минимумд хүрч очтол тэдгээрийн утгуудыг өөрчилнө. Тус алгоритмд бид дараах томъёоны дагуу давталт бүрд алдааны функцыг (cost function) тооцоолох ба параметрийн утгыг нэгэн зэрэг шинэчилдэг (update).

$$\theta := \theta - \alpha \frac{\delta}{\delta\theta} J(\theta).$$

Лекцийн жишээнд авч үзсэнээр

$$w_i = w_i - \alpha * \frac{dL}{dw_i}$$

Энд α нь сургалтын хурд (step)

- Алдааг багасгахын тулд гарсан хариуг буюу сургалтын хурд (learning rate) гэсэн хувьсагчаар үржиж болно.
- Сургалтын хурд хэт их бол градиентийн арга хаа хамаагүй үсчих тул минимум рүү нийлэхгүй.
- Сургалтын хурд хэт бага бол алгоритм маань минимум цэг рүү яст мэлхий шиг удаанаар дөхнө.

Бид шугаман регрессийн аргыг авч үзэх ба хэрэгжүүлэлтийг пайтон дээр numpy сантай ажиллана.

Градиент буулгалтын алгоритмын төрлүүд:

Градиент буулгалтын төрлүүдийг градиент буулгалт алдааны функцийн дериативыг тооцоолохдоо өгөгдлийг хэрхэн авч ашиглаж байгаагаас үндэслэн тодорхойлдог. Ашигласан өгөгдлийн хэмжээнээс хамааран алгоритмуудын хугацааны төвөгшилт (time complexity), нарийвчлал (accuracy) нь өөр өөр байдаг. Градиент буулгалтын үндсэн төрлүүд:

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-Batch Gradient Descent

Batch gradient descent

Градиент буулгалтын эхний төрөл нь датасэт бүхэлд нь градиент буулгалтыг тооцоолно. Бид зөвхөн нэг шинэчлэлт (update) хийхийн тулд бүх өгөгдлийн датасэт дээрх градиентийг тооцоолох шаардлагатай байдаг тул Batch Gradient Descent нь тооцоолол маш удаан бөгөөд санах ойд багтахгүй нөхцөл байдал бий болдог.

Параметрийг дурын утгуудаар эхлүүлсний дараа бид дараах хамаарлыг ашиглан алдааны функцийн градиентийг тооцоолно.

- $L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- $\hat{y}_i = w_1 x_i + w_0$
- $L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - w_1 x - w_0)^2$

$$L_{w1} = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

$$L_{w0} = \frac{-2}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)$$

Энд: ‘n’ сургалтын (training) тоо.

Өөр жишээ :

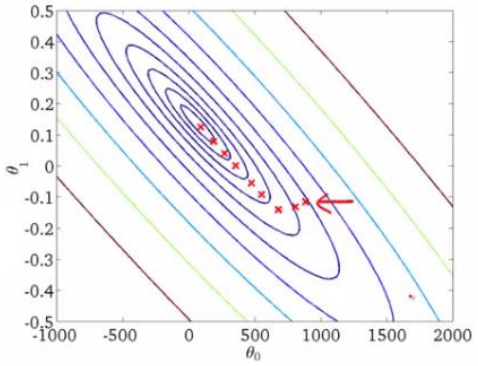
Repeat until convergence

{

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

}

- Хэрэв бидэнд 300,000,000 бичлэг байна гэж үзвэл бүгдийг санах ойд хадгалах боломжгүй тул бүх бичлэгийг дискнээс санах ойд унших шаардлагатай
- Нэг давталтын сигма-г тооцоолсны дараа бид нэг алхам шилжинэ.
- Дараа нь алхам бүрд давтана.
- Энэ нь нэгтгэхийн (integrate) тулд удаан хугацаа шаарддаг гэсэн үг юм.
- Ялангуяа дискний оролт/гаралт нь ихэвчлэн системд ачаалал үүсгдэг учраас асар их хэмжээний бичлэг унших шаардлагатай болдог.



Batch gradient descent асар том өгөгдлийн багцад тохиромжгүй.

Доорх код нь python дээр градиент descent-ийг хэрэгжүүлэх талаар тайлбарласан болно.

Жишээ №1: Энгийн жишээ

Алхам 1: Параметрүүд тодорхойлох.

```
cur_x = 3 # The algorithm starts at x=3
rate = 0.01 # Learning rate
precision = 0.000001 #This tells us when to stop the algorithm
previous_step_size = 1 #
max_iters = 10000 # maximum number of iterations
iters = 0 #iteration counter
df = lambda x: 2*(x+5) #Gradient of our function
```

Алхам 2: Градиент буулгалтын давталт ашиглах

```
while previous_step_size > precision and iters < max_iters:
    prev_x = cur_x #Store current x value in prev_x
    cur_x = cur_x - rate * df(prev_x) #Grad descent
    previous_step_size = abs(cur_x - prev_x) #Change in x
    iters = iters+1 #iteration count
    print("Iteration",iters,"\nX value is",cur_x) #Print iterations
```

print("The local minimum occurs at", cur_x)

Жишээ №2

```
import random

def gradient_descent(alpha, x, y, ep=0.0001, max_iter=10000):
    converged = False
    iter = 0
    m = x.shape[0] # number of samples

    # initial theta
    t0 = np.random.random(x.shape[1])
    t1 = np.random.random(x.shape[1])

    # total error, J(theta)
    J = sum([(t0 + t1*x[i] - y[i])**2 for i in range(m)])

    # Iterate Loop
    while not converged:
        # for each training sample, compute the gradient (d/d_theta j(theta))
        grad0 = 1.0/m * sum([(t0 + t1*x[i] - y[i]) for i in range(m)])
        grad1 = 1.0/m * sum([(t0 + t1*x[i] - y[i])*x[i] for i in range(m)])

        # update the theta_temp
        temp0 = t0 - alpha * grad0
        temp1 = t1 - alpha * grad1

        # update theta
        t0 = temp0
        t1 = temp1

        # mean squared error
        e = sum( [ (t0 + t1*x[i] - y[i])**2 for i in range(m)] )

        if abs(J-e) <= ep:
            print 'Converged, iterations: ', iter, '!!!'
            converged = True

        J = e   # update error
        iter += 1  # update iter

        if iter == max_iter:
            print 'Max interactions exceeded!'
            converged = True

    return t0,t1
```

Стохастик градиент бүүлгалт (stochastic gradient descent):

Batch Gradient Descent нь удаан алгоритм болж таарав. Иймд илүү хурдтай алгоритм сонгох шаардлагатай бөгөөд үүнд stochastic gradient descent алгоритм орно.

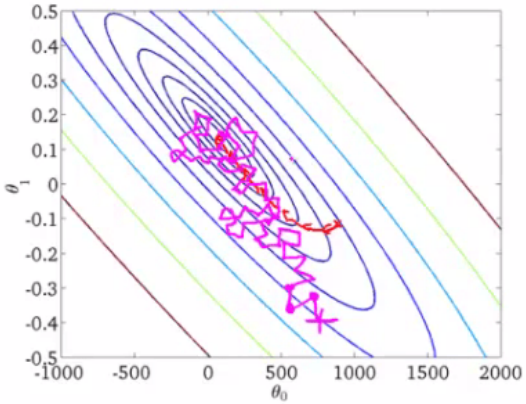
Алгоритмын эхний алхам бол бүх сургалтын багцыг (training set) санамсаргүй байдлаар хуваах (түүвэр) явдал юм. Дараа нь параметр бүрийг шинэчлэхийн (update) тулд бид давталт бүрт зөвхөн нэг сургалтын түүвэр ашиглан алдааны градиентийг тооцоолоход ашигладаг. Давталт бүрт нэг сургалтын түүвэр ашигладаг тул энэ алгоритм нь том өгөгдлийн багцад илүү хурдан байдаг. SGD-д нарийвчлалд (accuracy) хүрэхгүй байж болох ч үр дүнг тооцоолох нь илүү хурдан байдаг.

Параметрийг дурын утгуудаар эхлүүлсний дараа бид дараах хамаарлыг ашиглан алдааны градиентийг тооцоолно.

Randomly shuffle (reorder) training examples

Repeat {
  for $i := 1, \ldots, m$ {
    $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
      (for every $j = 0, \ldots, n$)
  }
}

'm' нь сургалтын түүвэр тоо.



Mini batch gradient descent

Mini batch алгоритм нь “m” сургалтын түүврийг ашиглан илүү өндөр нарийвчлалтай (accurcy), хурдан үр дүн гаргадаг тул нилээд өргөн ашиглагддаг. Тус алгоритм нь цогц өгөгдлийн багц ашиглахын оронд давталт бүрд “m” сургалтын түүврийг ашиглан алдааны функцын градиентыг тооцоолдог. Mini-batch нийтлэг хэмжээ нь 50-аас 256 хооронд боловч хэрэглээнээс хамаараад өөр өөр байж болно.

Say $b = 10, m = 1000.$
Repeat {
  for $i = 1, 11, 21, 31, \ldots, 991$ {
    $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$
        (for every $j = 0, \ldots, n$) } }

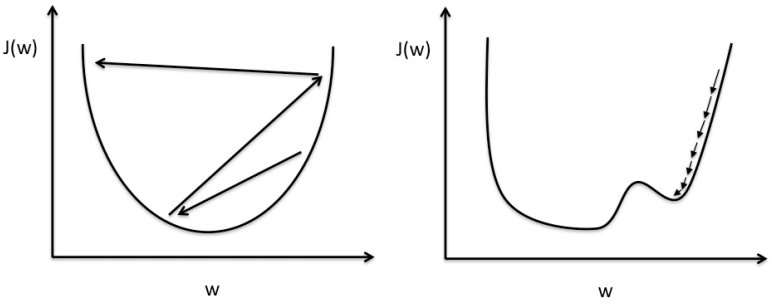
‘ b ‘ нь batches тоо ‘ m ‘ нь сургалтын түүврийн тоо.

Анхаарвал зохих зүйлс:

- Параметрүүдээ давтамжтайгаар шинэчлэнэ (update)
  Энэ нь параметрийн утгыг эхлээд зарим түр зуурын хувьсагчд хадгалж, дараа нь параметрүүдэд хуваарилах ёстой гэсэн үг юм.

Сургалтын хурд (Learning rate)

Алгоритм хэр хэмжээтэй алхам хийхийг хянадаг чухал параметр



Large learning rate: Overshooting.

Small learning rate: Many iterations until convergence and trapping in local minima.

Жишээ код:

Градиент буулгалтын псевдо код:

1. Параметр утгуудыг үүсгэх (Initialize)
2. Алдааны утгыг хамгийн бага хэмжээнд хүртэл давтах.

Python

```
def gradient_descent(gradient, start, learn_rate, n_iter):
    vector = start
    for _ in range(n_iter):
        diff = -learn_rate * gradient(vector)
        vector += diff
    return vector
```

Python

```
import numpy as np

def gradient_descent(
    gradient, start, learn_rate, n_iter=50, tolerance=1e-06
):
    vector = start
    for _ in range(n_iter):
        diff = -learn_rate * gradient(vector)
        if np.all(np.abs(diff) <= tolerance):
            break
        vector += diff
    return vector
```

Python >>>

```
>>> gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.2
... )
2.210739197207331e-06
```

Даалгавар: Дээрх 3 аргыг тодорхойлсон функц бичиж, үр дүнг харьцуулна уу.