

Gradient descent: implementálás

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2021. november 17.

Legmeredekebb ereszkedés gépi tanulásban¹

Stratégia

- iteratív algoritmus
- válasszunk egy tetszőleges kezdőértéket a a_0 , a_1 -nak
- repeat until convergence {
 $a_j := a_j - \alpha \frac{\partial}{\partial a_j} J(\mathbf{a}; \mathbf{x}^{(i)}, y^{(i)})$
}

Néhány gyakorlati kérdés az implemetációval kapcsolatban

- hogyan frissítsük a a_j értékeket?
- hogyan számoljuk a költségfüggvény deriváltjait?
- hogyan határozzuk meg az α learning rate-t?
- hogyan figyeljük a konvergenciát?

¹Andrew Ng, Coursera alapján

Egyidejű frissítés (simultaneous update)

Példa: két paraméter, a_0 , a_1 . Hogyan frissítsük $\frac{\partial}{\partial a_j} J(a_0, a_1)$ értékét?

Előkészület:

- a_0 és a_1 egy $a[i]$ vektor elemeiként tárolhatjuk
- hasonlóan $\frac{\partial}{\partial a_0} J(a_0, a_1)$ és $\frac{\partial}{\partial a_1} J(a_0, a_1)$ értékét egy $\text{derivJ}[i]$ vektor elemeiként

Egyidejű frissítés (simultaneous update)

Példa: két paraméter, a_0 , a_1 . Hogyan frissítsük $\frac{\partial}{\partial a_j} J(a_0, a_1)$ értékét?

Előkészület:

- a_0 és a_1 egy $a[i]$ vektor elemeiként tárolhatjuk
- hasonlóan $\frac{\partial}{\partial a_0} J(a_0, a_1)$ és $\frac{\partial}{\partial a_1} J(a_0, a_1)$ értékét egy $\text{derivJ}[i]$ vektor elemeiként

Egyidejű frissítés

```
1 do
2 {
3 deriv_calc(derivJ, a_vec, data, params);
4 //calculate the derivative
5
6 a_vec[0]=a_vec[0]-alpha*derivJ[0];
7 a_vec[1]=a_vec[1]-alpha*derivJ[1];
8
9 J=fun_calc(a_vec, data, params);
10 //calculate J to monitor convergence
11 }
12 while (...) //convergence achieved
```

Egyidejű frissítés (simultaneous update)

Példa: két paraméter, a_0 , a_1 . Hogyan frissítsük $\frac{\partial}{\partial a_j} J(a_0, a_1)$ értékét?

Előkészület:

- a_0 és a_1 egy $a[i]$ vektor elemeiként tárolhatjuk
- hasonlóan $\frac{\partial}{\partial a_0} J(a_0, a_1)$ és $\frac{\partial}{\partial a_1} J(a_0, a_1)$ értékét egy $\text{derivJ}[i]$ vektor elemeiként

Egyidejű frissítés

```
1 do
2 {
3 deriv_calc(derivJ, a_vec, data, params);
4 //calculate the derivative
5
6 a_vec[0]=a_vec[0]-alpha*derivJ[0];
7 a_vec[1]=a_vec[1]-alpha*derivJ[1];
8
9 J=fun_calc(a_vec, data, params);
10 //calculate J to monitor convergence
11 }
12 while (...) //convergence achieved
```

Nem egyidejű frissítés

```
1 do
2 {
3 derivJ[0]=deriv0_calc(a_vec, data, params);
4 a_vec[0]=a_vec[0]-alpha*derivJ[0];
5 //a_0 updated
6
7 derivJ[1]=deriv1_calc(a_vec, data, params);
8 //uses the updated a_vec[0]
9 // and the old a_vec[1]
10 a_vec[1]=a_vec[1]-alpha*derivJ[1];
11
12 J=fun_calc(a_vec, data, params);
13 //calculate J to minitor convergence
14 }
15 while (...) //convergence achieved
```

Egyidejű frissítés (simultaneous update)

Példa: két paraméter, a_0 , a_1 . Hogyan frissítsük $\frac{\partial}{\partial a_j} J(a_0, a_1)$ értékét?

Előkészület:

- a_0 és a_1 egy $a[i]$ vektor elemeiként tárolhatjuk
- hasonlóan $\frac{\partial}{\partial a_0} J(a_0, a_1)$ és $\frac{\partial}{\partial a_1} J(a_0, a_1)$ értékét egy $\text{derivJ}[i]$ vektor elemeiként

Egyidejű frissítés

```
1 do
2 {
3 deriv_calc(derivJ, a_vec, data, params);
4 //calculate the derivative
5
6 a_vec[0]=a_vec[0]-alpha*derivJ[0];
7 a_vec[1]=a_vec[1]-alpha*derivJ[1];
8
9 J=fun_calc(a_vec, data, params);
10 //calculate J to monitor convergence
11 }
12 while (...) //convergence achieved
```

Nem egyidejű frissítés

```
1 do
2 {
3     derivJ[0]=deriv0_calc(a_vec, data, params);
4     a_vec[0]=a_vec[0]-alpha*derivJ[0];
5     //a_0 updated
6
7     derivJ[1]=deriv1_calc(a_vec, data, params);
8     //uses the updated a_vec[0]
9     // and the old a_vec[1]
10    a_vec[1]=a_vec[1]-alpha*derivJ[1];
11
12    J=fun_calc(a_vec, data, params);
13    //calculate J to minitor convergence
14 }
15 while (...) //convergence achieved
```

Az **egyidejű frissítést** (simultaneous update) használjuk!

Hogyan számoljuk a költségfüggvény deriváltjait?

Két széleskörben alkalmazott eljárás:

- batch gradient descent
- stochastic gradient descent

Batch gradient descent

Példa:

- legkisebb négyzetek módszere
- költségfüggvény $J(a_0, a_1; \mathbf{x}^{(i)}, y^{(i)}) = \frac{1}{2N} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{a}))^2$
- egy $h(x; \mathbf{a}) = a_0 + a_1x$ függvényt szeretnénk illeszteni
- jelölés: $h(\mathbf{x}; \mathbf{a}) = a_0x_0 + a_1x_1$, $x_0 = 1.0$
- parciális deriváltak (egzakt eredmény):

$$\frac{\partial}{\partial a_j} J(\mathbf{a}; \mathbf{x}^{(i)}, y^{(i)}) = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{a})) x_j^{(i)}$$

Batch gradient descent

Példa:

- legkisebb négyzetek módszere
- költségfüggvény $J(a_0, a_1; \mathbf{x}^{(i)}, y^{(i)}) = \frac{1}{2N} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{a}))^2$
- egy $h(x; \mathbf{a}) = a_0 + a_1x$ függvényt szeretnénk illeszteni
- jelölés: $h(\mathbf{x}; \mathbf{a}) = a_0x_0 + a_1x_1$, $x_0 = 1.0$
- parciális deriváltak (egzakt eredmény):

$$\frac{\partial}{\partial a_j} J(\mathbf{a}; \mathbf{x}^{(i)}, y^{(i)}) = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{a})) x_j^{(i)}$$

- a fenti képlet közvetlenül beprogramozható
- az összes N adatot használjuk a parciális deriváltak kiszámolására
- ez a **batch gradient descent**

Stochastic gradient descent

Parciális deriváltak (egzakt):

$$\frac{\partial}{\partial a_j} J(\mathbf{a}; \mathbf{x}^{(i)}, y^{(i)}) = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{a})) x_j^{(i)}$$

- ha N nagyon nagy (akár $\sim 10^6 - 10^7$), akkor az összegek kiszámítása minden iterációs lépésben sok időt vesz el
- minden iterációs lépésben válasszunk véletlenszerűen egy $(\mathbf{x}^{(k)}, y^{(k)})$ adatot
- az összeget egyetlen taggal “közelítjük”

$$\frac{\partial}{\partial a_j} J(\mathbf{a}; \mathbf{x}^{(i)}, y^{(i)}) \approx (y^{(k)} - h(\mathbf{x}^{(k)}; \mathbf{a})) x_j^{(k)}$$

- ez a **stochastic gradient descent**

Learning rate, konvergencia figyelése

Paraméterek iteratív meghatározása:

$$a_j := a_j - \alpha \frac{\partial}{\partial a_j} J(\mathbf{a}; \mathbf{x}^{(i)}, y^{(i)})$$

Hogyan válasszuk az α learning rate-t ?

Learning rate, konvergencia figyelése

Paraméterek iteratív meghatározása:

$$a_j := a_j - \alpha \frac{\partial}{\partial a_j} J(\mathbf{a}; \mathbf{x}^{(i)}, y^{(i)})$$

Hogyan válasszuk az α learning rate-t ?

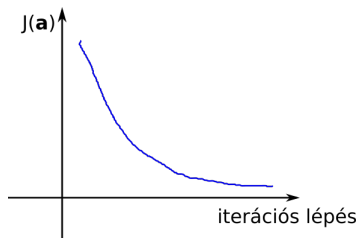
batch gradient descent

- jelöljük $\mathbf{a}_{(p)}$ -vel a p -ik iteráció eredményét
- ha α elég kicsi, akkor $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ minden iteráció során csökken
- ha α nem elég kicsi, akkor $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ növekszik vagy esetleg oszcillál
- ha α túl kicsi, akkor lassú lesz a konvergencia

Learning rate, konvergencia figyelése

Eljárás **batch gradient descent**-re

- válasszunk egy α -t
- minden iterációs lépésben értékeljük ki $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ -t
- ábrázoljuk $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ -t iterációs lépésszám függvényében

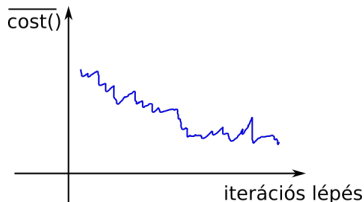


- ha $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ csökken, akkor α nem túl nagy
- ha $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ nem csökken, akkor válasszunk egy kisebb α -t
- ha $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ nem csökken bizonyos iterációs szám fölött \Rightarrow konvergencia

Learning rate, konvergencia figyelése

Eljárás **stochastic gradient descent**-re

- minden iterációs lépésben, mielőtt még frissítenénk $\mathbf{a}_{(p)}$, számítsuk ki a $\text{cost}(y^{(k)}, h(\mathbf{a}_{(p)}; \mathbf{x}^{(k)}))$
- pl legkisebb négyzetek egyenes illesztés esetén $\text{cost}(y^{(k)}, h(\mathbf{a}_{(p)}; \mathbf{x}^{(k)})) = \frac{1}{2}(y^{(k)} - (a_{0,(p)} + a_{1,(p)}x^{(k)}))^2$ -t
- minden iterációs lépésben jegyezzük fel $\text{cost}(y^{(k)}, h(\mathbf{a}; \mathbf{x}^{(k)}))$ -t
- 100 v 1000 stb lépésenként írjuk ki a $\text{cost}(y^{(k)}, h(\mathbf{a}; \mathbf{x}^{(k)}))$ -k **átlagát**



- ha α nagysága megfelelő, akkor az átlag csökkenő trendet mutat