Gradient descent: implementálás

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2020. október 19.

Legmeredekebb ereszkedés gépi tanulásban[1]

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?

[1] Andrew 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 `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 `derivJ[i]` vektor elemeiként

Egyidejű frissítés

```
  do
  {
  deriv_calc(derivJ, a_vec, data, params);
  //calculate the derivative

  a_vec[0]=a_vec[0]-alpha*derivJ[0];
  a_vec[1]=a_vec[1]-alpha*derivJ[1];

  J=fun_calc(a_vec, data, params);
  //calculate J to monitor convergence
  }
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 `derivJ[i]` vektor elemeiként

Egyidejű frissítés

```
do
{
deriv_calc(derivJ, a_vec, data, params);
//calculate the derivative

a_vec[0]=a_vec[0]-alpha*derivJ[0];
a_vec[1]=a_vec[1]-alpha*derivJ[1];

J=fun_calc(a_vec, data, params);
//calculate J to monitor convergence
}
while (...) //convergence achieved
```

Nem egyidejű frissítés

```
do
  {
    derivJ[0]=deriv0_calc(a_vec, data, params);
    a_vec[0]=a_vec[0]-alpha*derivJ[0];
    //a_0 updated

    derivJ[1]=deriv1_calc(a_vec, data, params);
    //uses the updated a_vec[0]
    // and the old a_vec[1]
    a_vec[1]=a_vec[1]-alpha*derivJ[1];

  J=fun_calc(a_vec, data, params);
  //calculate J to minitor convergence
  }
  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 `derivJ[i]` vektor elemeiként

Egyidejű frissítés

```
  do
  {
  deriv_calc(derivJ, a_vec, data, params);
  //calculate the derivative

  a_vec[0]=a_vec[0]-alpha*derivJ[0];
  a_vec[1]=a_vec[1]-alpha*derivJ[1];

  J=fun_calc(a_vec, data, params);
  //calculate J to monitor convergence
  }
 while (...) //convergence achieved
```

Nem egyidejű frissítés

```
do
   {
    derivJ[0]=deriv0_calc(a_vec, data, params);
    a_vec[0]=a_vec[0]-alpha*derivJ[0];
    //a_0 updated

    derivJ[1]=deriv1_calc(a_vec, data, params);
    //uses the updated a_vec[0]
    // and the old a_vec[1]
    a_vec[1]=a_vec[1]-alpha*derivJ[1];

  J=fun_calc(a_vec, data, params);
  //calculate J to minitor convergence
  }
  while (...) //convergence achieved
```

Az egyidejű frissítést (simultaneous update) használjuk!

Hogyan számoljuk a költségfüggvény deriváljait?

Két széleskörben alkalmazott eljárás:

- batch gradient descent
- stochastic gradient descent

Batch gradient descent

Példa:

- legkisebb négyzetek módszere
- költségfüggvény $J(a_0, a_1; x^{(i)}, y^{(i)}) = \frac{1}{2N} \sum_{i=1}^{N} \left(y^{(i)} - h(x^{(i)}; \mathbf{a})\right)^2$
- egy $h(x; \mathbf{a}) = a_0 + a_1 x$ függvényt szeretnénk illeszteni
- jelölés: $h(\mathbf{x}; \mathbf{a}) = a_0 x_0 + a_1 x_1$, $x_0 = 1.0$
- parciális deriváltak (egzakt eredmény):

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

Batch gradient descent

Példa:

- legkisebb négyzetek módszere
- költségfüggvény $J(a_0, a_1; x^{(i)}, y^{(i)}) = \frac{1}{2N} \sum_{i=1}^{N} \left(y^{(i)} - h(x^{(i)}; \mathbf{a})\right)^2$
- egy $h(x; \mathbf{a}) = a_0 + a_1 x$ függvényt szeretnénk illeszteni
- jelölés: $h(\mathbf{x}; \mathbf{a}) = a_0 x_0 + a_1 x_1$, $x_0 = 1.0$
- parciális deriváltak (egzakt eredmény):

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

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

- 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"

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

- 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 ?

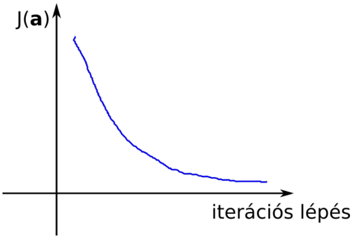
batch gradient descent

- 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

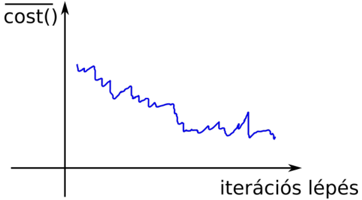


- ha $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ csökken, akkor α nem túl nagy
- ha $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ nem csökken, akkor válasszunk egy kisebb α-t
- ha $J(\mathbf{a}_{(p)}; \mathbf{x}^{(i)}, y^{(i)})$ nem csökken bizonyos iterációs szám fölött $\Rightarrow$ konvergencia

Learning rate, konvergencia figyelése

Eljárás stochastic gradient descent-re

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



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