# Օպտիմիզացման ալգորիթմներ

Հայկ Կարապետյան

Օպտիմիզացման ալգորիթմերից հիշում ենք stochastic gradient descent-ը և mini-batch gradient descent-ը։ Ծանոթանանք ևս մի քանի օպտիմիզացման ալգորիթմների հետ։

## 1 Գրադիենտային վայրէջք թափով (momentum)

Եկեք վերհիշենք mini-batch gradient descent-ը։ Այդ ալգորիթմի դեպքում մենք վերցնում էինք առաջին batch-size հատ տվյալները, ստանում նրանց գրադիենտների ուղղությունները և միջինացնում։ Այսինքն ամեն քայլի մենք ոչ թե օգտագործում էինք բոլոր տվյալները, այլ նրանց մի մասը։

$$L_i = \frac{1}{100} \sum_{k=100(i-1)+1}^{100i} L_k,\ batch_size = 100$$

Ամեն քայլի հաշվի ենք առնում 100 տվյալ (ռեսուրսներ և ժամանակ խնայելու համար) և շարժվում դրանց ուղղությամբ

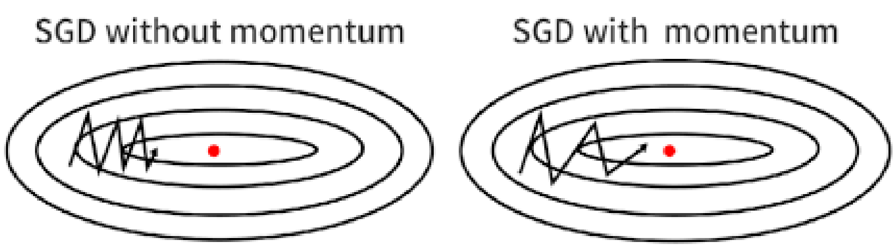
Եթե մենք վերցնենք մի կետում հաշվենք $\{L_1, L_2, \ldots L_10\}$ գրադիենտները և դրանք միջինացնենք, ապա կստացվի նույնը, եթե գրադիենտային վայրէջք կատարեինք 1000 batch-size-ով։

$$\frac{1}{10} L_{1,\ldots,10} = \frac{1}{10} \left( \frac{1}{100} \sum_{k=1}^{100} L_k + \frac{1}{100} \sum_{k=101}^{200} L_k + \cdots + \frac{1}{100} \sum_{k=901}^{1000} L_k \right) = \frac{1}{1000} \sum_{k=1}^{1000} L_k$$

Եթե միջինացնենք առաջին 10 batch-երը և նոր մեկ քայլ կատարենք, դա նույնն է, որ մեկ քայլ կատարենք, երբ batch-size=1000։ Մենք այսպես չենք անում, ժամանակ խնայելու համար։ Այսինքն առաջարկն այն է, որ 10 գրադիենտի ուղղություն հաշվենք, նրանց միջինացնենք և նոր մեկ անգամ թարմացնենք կշիռները, փոխարենը կարող էինք 10 անգամ թարմացնել կշիռները։ Հետևյալ եզրակացությունից առաջանում է Gradient descent with momentum ալգորիթմը։

$$\begin{gathered} v_0 = 0 \\ v_t = \beta v_{t-1} + (1-\beta) x \nabla L(w_T), \beta \in [0,1] \\ w_{t+1} = w_t - \alpha v_t \end{gathered}$$

Այս դեպքում, բացի նոր տվյալի վրա հաշված գրադիենտի ուղղությունից, օգտագործում ենք նաև նախորդ ուղղությունները։ Բայց ամեն քայլի պետք չէ բոլոր նախորդ ուղղությունները նորից հաշվել, մենք կարող ենք կիրառել էքսպոնենցիալ շարժվող միջին և հիշել միայն նախորդ արժեքը։ Նախորդ ուղղությունները հաշվի առնելը կարող է օգնել outlier-ներին շատ ուշադրություն չդարձնել և ավելի արագ հասնել մինիմումի կետին (Նկար 1)։




Նկար 1: Առանց շարժվող միջինի և շարժվող միջինով գրադիենտային վայրէջքի քայլերը

## 2 RMSProp

Gradient descent with momentum ալգորիթմը հաշվի էր առնում նախորդ քայլերում գրադիենտի ուղղությունը: Հիմա տեսնենք, թե ինչպես կարող ենք հաշվի առնել նաև գրադիենտի արժեքը: Մեր նպատակն է փոքրացնել ուսուցման արագությունը, երբ գրադիենտի արժեքը մեծ է և մեծացնել ուսուցման արագությունը, երբ գրադիենտի արժեքը փոքր է: Դիտարկենք հետևյալ օպտիմիզացման ալգորիթմը.

$$w = w - \alpha \frac{\nabla L}{\sqrt{\nabla L^2}}$$

$$\nabla L^2 = \left[ \left( \frac{\partial L}{\partial w_1} \right)^2, \ldots, \left( \frac{\partial L}{\partial w_n} \right)^2 \right]$$

$$\frac{\nabla L^2}{\sqrt{\nabla L^2}} = \left[ sign \left( \frac{\partial L}{\partial w_1} \right), \ldots, sign \left( \frac{\partial L}{\partial w_n} \right) \right]$$

Տեսանք որ կորստի գրադիենտը բաժանելով իր մոդուլի վրա ստանում ենք սիգնում, այսինքն 1 կամ -1: Կշիռները թարմացնելիս ամեն քայլի w-ից կա՛մ հանելու ենք $\alpha$, կա՛մ գումարելու: Եթե w-ի արժեքը փոփոխենք $\alpha$-ով կամ -$\alpha$-ով, դրանք կարող են իրար ոչնչացնել և w-ի արժեքը կմնա նույնը: Նաև այս դեպքում գրադիենտի արժեքի նշանն ենք միայն փոփոխում: Այդ պատճառով, ինչպես նախորդ դեպքում օգտագործենք շարժվող միջին:

$$v_0 = 0$$

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla L(w_t)^2)$$

$$w_{t+1} = w_t - \alpha \frac{\nabla L(w_t)}{\sqrt{v_t} + \varepsilon}$$

Այժմ բացի $\alpha$-ի նշանը փոփոխելուց, փոխում ենք նաև նրա արժեքը:

Հայտարարում էպսիլոնը գումարում ենք 0 չստանալու համար:

## 3 ADAM

Այսպիսով և՛ առաջին (1), և՛ երկրորդ (2) օպտիմիզացման ալգորիթմների ժամանակ օգտագործում ենք էքսպոնենցիալ շարժվող միջին: Առաջինի դեպքում հաշվի ենք առնում նախորդ գրադիենտների ուղղությունները, իսկ երկրորդի դեպքում գրադիենտների արժեքները: Հիմա փորձենք մեկ օպտիմիզացման ալգորիթմի ժամանակ հաշվի առնել և՛ գրադիենտների ուղղությունները, և՛ գրադիենտների արժեքները:

$$m_0 = 0, v_0 = 0$$

Հաշվի առնենք գրադիենտների ուղղությունները

$$m_t = \beta m_{t-1} + (1 - \beta)\nabla(w_{t-1})$$

Հաշվի առնենք գրադիենտների արժեքները

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L(w_{t-1}))^2$$

Բաժանման մասին կխոսենք

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

Թարմացնենք կշիռները

$$w_t = w_{t-1} - \alpha \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon}$$

Հետևյալ օպտիմիզացման ալգորիթմը կոչվում է ADAM (Adaptive Moment Estimation)

Հիմա նայենք, թե ինչի համար ենք կատարում բաժանում.

$$m_t = \alpha m_{t-1} + (1-\alpha)\nabla L(w_{t-1}) =$$
$$= \alpha^2 m_{t-2} + \alpha(1-\alpha)\nabla L(w_{t-2}) + (1-\alpha)\nabla L(w_{t-1}) =$$
$$= \alpha^t m_0 + \alpha^{t-1}(1-\alpha)\nabla L(w_0) + \alpha^{t-2}(1-\alpha)\nabla L(w_1) + \cdots + \alpha^0(1-\alpha)\nabla L(w_{t-1})$$

Դիտարկենք գրադիենտների գործակիցների գումարը

$$\alpha^t m_0 + \alpha^{t-1}(1-\alpha) + \alpha^{t-2}(1-\alpha) + \cdots + \alpha^0(1-\alpha) = 0 + (1-\alpha)(\alpha^{t-1+\alpha^{t-2}+\cdots+\alpha^0})$$

Հետևյալ արտադրյալը ըստ նյուտոնի բինոմի կստանանք

$$1-\alpha^t$$

Ստացանք, որ գործակիցների գումարը հավասար է $1-\alpha^t$, այն պատճառով, որ $m_0 = 0$: Դրա համար բոլոր գործակիցները բաժանում ենք $1-\alpha^t$-ի, որպեսզի գումարը ստանանք 1: Ինչքան քայլերը շատ անենք t-ն ավելի մեծանալու է և $\alpha$-ն ձգտելու է 0-ի, այսինքն գործակիցների բաժանումը նրանց գումարի մեծ նշանակություն չի ունենալու, բայց այդ դեպքում էլ բաժանումը կատարում ենք:

Հետևյալ օպտիմիզացման ալգորիթմում $\beta_1, \beta_2$ և $\alpha$-ն հիպերպարամետրեր են: $\beta_1$ և $\beta_2$ հիպերպարամետրերը tensorflow և pytorch գրադարաններում ունեն լավ արժեքներ, որոնք ստացվել են տարբեր տարբերակներ փորձարկելուց և լավագույնը ընտրելուց հետո: $\alpha$ հիպերպարամետրը նույնպես ունի լավ արժեք, որը շատ ցանցերի դեպքում հարմար է, բայց այս հիպերպարամետրի ընտրությունը կատարվում է կախված մեր ցանցից, իսկ $\beta_1$ և $\beta_2$ հիպերպարամետրերը թողնում ենք նույնը, քանի որ դրանց արժեքները, գրեթե միշտ լավ են աշխատում և դրան զուգահեռ քչացնում ենք հիպերպարամետրերի քանակը, որոնք անհրաժեշտ է փոփոխել validation տվյալների վրա: