PRIRODNO-MATEMATIČKI FAKULTET

DEPARTMAN ZA RAČUNARSKE NAUKE

SOFTVERSKI PRAKTIKUM

Algoritmi za optimizaciju gradijentnog spusta

Student:
Lazar Stojković

Profesor:
Dr Branimir Todorović

26.06.2016, Niš

Sadržaj

1 Uvod

Gradijentni spust je jedan od danas najpopularnijih algoritama kojim se vrši optimizacija neuronskih mreža. Trenutno sve poznatije *Deep learning* biblioteke, kao što su `lasagne`, `caffe` i `keras` dokumentacije, sadrže implementacije raznih algoritama koji optimizuju gradijentni spust. Ovi algoritmi se često koriste kao tzv. *black-box* optimizatori, što znači da je u praksi teško objasniti njihove prednosti i mane.

Ovaj rad sadrži opise nekoliko ovakvih algoritama. Najpre je predstavljen standardni stohastički gradijentni spust, koji ujedno predstavlja osnovu koja se ostalim algoritmima optimizuje. Zatim su opisani sledeći algoritmi: Momentum, Adagrad, Adadelta i Adam.

2 Stohastički gradijentni spust

Gradijentni spust ima za cilj da minimizuje objektivnu funkciju $J(\theta)$, parametrizovanu parametrima modela $\theta \in \mathbb{R}^d$, ažurirajući respektivno parametre u suprotnom smeru u odnosu na smer gradijenta objektivne funkcije $\nabla_\theta J(\theta)$. Stopa učenja (*learning rate*) η, određuje veličinu koraka koji se prave kako bi se dostigao minimum. Drugim rečima, ide se 'niz' nagib površine koju daje objektivna funkcija, sve dok se ne stigne (približno) do njenog dna (minimuma).

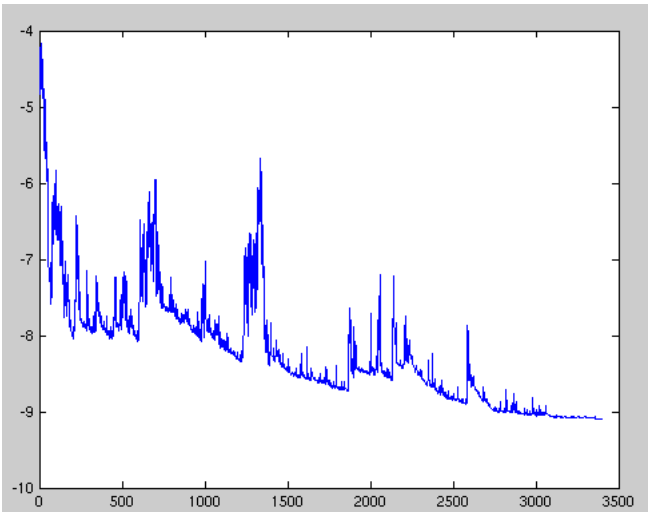
Stohastički gradijentni spust (SGD) vrši ažuriranje parametara nakon obrade svakog trening primerka $\left(x^{(i)}, y^{(i)}\right)$:

$$\theta = \theta \ominus \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}, y^{(i)}\right).$$

SGD često vrši ažuriranja sa visokom varijansom, što izaziva veliko kolebanje objektivne funkcije.

Velika fluktuacija kod SGD algoritma može omogućiti konvergenciju ka 'boljem' lokalnom minimumu. Sa druge strane, ta konvergencija neće biti precizna zbog stalnih (i eventualno velikih) kolebanja. U praksi se pokazalo da blagim smanjivanjem stope učenja[1] SGD skoro sigurno daje konvergenciju ka lokalnom ili globalnom minimumu, nekonveksne ili konveksne funkcije optimizacije respektivno.

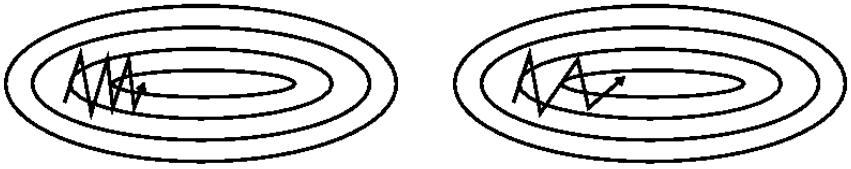
[1]Veličina stope učenja varira u zavisnosti od samog problema koji neuronska mreža rešava. Neki algoritmi (Momentum) favorizuju manje stope učenja ($\eta \approx 0.1$), dok konstantno visoke stope učenja ($\eta > 0.9$) mogu sabotirati konvergenciju.



Slika 1: Fluktuacija objektivne funkcije kod SGD

3 Momentum

SGD ima problema sa navigacijom u onim delovima gde se površ mnogo više 'krivi' u jednoj dimenziji u odnosu na ostale, što je karakteristično za samu okolinu optimalnog minimuma. U tom scenariju, SGD oscilira duž nagiba površi i neodlučno napreduje ka optimalnom minimumu.



Slika 2: SGD i SGD+Momentum

Momentum usmerava SGD u željenom pravcu i prigušuje oscilacije. To postiže dodavanjem udela γ[2] vektora za ažuriranje parametara iz prethodnog koraka (*time step*), vektoru za ažuriranje parametara u trenutnom koraku:

$$\begin{aligned} v_t &= \gamma v_{t-1} \oplus \eta \cdot \nabla_\theta J(\theta), \\ \theta_t &= \theta_{t-1} \ominus v_t. \end{aligned} \tag{1}$$

[2]Često se uzima $\gamma \approx 0.9$.

Efekat Momentuma se može porediti sa kotrljanjem lopte nizbrdo. Lopta takvim kotrljanjem 'sakuplja' momenat i postaje sve brža i brža, sve dok ne dostigne terminalnu brzinu ili cilj.

Sabirak u (1) koji se odnosi na momenat (γv_{t-1}) se povećava u onim dimenzijama u kojima su gradijenti isto usmereni i smanjuje u onim dimenzijama u kojima su gradijenti različito usmereni (posmatrajući sve prethodne i trenutni korak). Za rezultat imamo bržu konvergenciju i manje oscilacije.

4 Adagrad

Adagrad je optimizacioni algoritam koji jednostavno prilagođava stopu učenja, tako da se veća ažuriranja primenjuju na manje efektivne parametre i manja ažuriranja na uticajnije parametre. Iz tog razloga Adagrad daje dobre rezultate kada su podaci za treniranje razređeni (u susednim koracima na 'većem' rastojanju).

Dean [1] je primetio da Adagrad znatno poboljšava robusnost SGD alogrtma prilikom treniranja ogromnih neuronskih mreža. Ovaj algoritam je, uostalom, korišćen od strane kompanije Google za treniranje neuronske mreže koja je naučila da prepozna mačke u YouTube klipovima.

U prethodnim algoritmima smo imali konstantnu stopu učenja η, dok Adagrad u svakom koraku ima drugačiju stopu učenja. Radi jednostavnijeg zapisa, označimo sa $g_{t,i}$ gradijent objektivne funkcije J po parametru θ_i u koraku t tj:

$$g_{t,i} = \nabla_\theta J(\theta_i).$$

Sa ovim oznakama bi formula za ažuriranje parametra θ_i u koraku t kod SGD algoritma glasila:

$$\theta_{t,i} = \theta_{t-1,i} - \eta \cdot g_{t,i}.$$

Pravilo za ažuriranje parametra θ_i u koraku t kod algoritma Adagrad glasi:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \cdot g_{t,i},$$

gde je $G \in \mathbb{R}^d$ vektor istih dimenzija kao i vekotor svih parametara θ. Pritom, $G_{t,i}$ predstavlja sumu kvadrata svih gradijenata $g_{k,i}$ do koraka t ($k \leq t$), dok ϵ predstavlja veoma mali broj[3] koji se dodaje kako bi se izbeglo deljenje nulom. Zanimljiva je i

[3] Najčešće reda 10^{-8}.

činjenica da bez operacije kvadratni koren ovaj algoritam ima daleko gore performanse.

Jedan od glavnih aduta algoritma Adagrad jeste eliminisanje potrebe da se tokom procesa treniranja neuronske mreže ručno prilagođava stopa učenja[4]. Osnovna mana ovog algoritma jeste stalan rast vrednosti u vektoru G. To uzrokuje da efektivna stopa učenja kod ovog algoritma nakon izvesnog broja koraka (neminovno kod ogromnih *training set*-ova) postane toliko mala da mreža u tom trenutku prestaje da uči.

5 Adadelta

Adadelta predstavlja ekstenziju Adagrad algoritma i za cilj ima da smanji monotono opadanje stope učenja. Umesto akumulacije kvadrata svih prethodnih gradijenata, Adadelta se ograničava na neki fiksirani broj, w, kvadrata prethodnih gradijenata.

Umesto čuvanja kvadrata prethodnih w gradijenata (što je neefikasno), suma kvadrata gradijenata se rekurzivno definiše kao opadajući prosek sume prethodnih kvadrata gradijenata i kvadrata trenutnog gradijenta. Slično kao kod Momentuma, definiše udeo γ kvadrata svih prethodnih gradijenata:

$$E[g^2]_t = \gamma E[g^2]_{t-1} \oplus (1-\gamma) g_t^2.$$

Sada se jednostavno u algoritmu Adagrad vektor G zameni vektorom $E[g^2]$, pa su vektor ažuriranja i pravilo ažuriranja u koraku t:

$$\begin{aligned} \Delta\theta_t &= -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \odot g_t, \\ \theta_t &= \theta_{t-1} \oplus \Delta\theta_t. \end{aligned}$$

Delilac u efektivnoj stopi učenja predstavlja *root mean squared* (RMS) *error* kriterijum za gradijente, stoga možemo kraće pisati:

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} \odot g_t.$$

Zeiler [2] tvrdi da se jedinice prilikom ovog ažuriranja ne poklapaju i da vektor ažuriranja treba imati iste hipotetičke jedinice kao i vektor parametara. Kako bi se to realizovalo, najpre definiše još jedan opadajući prosek, ovog puta ne kvadrata prethodnih gradijenata već kvadrata prethodnih vektora ažuriranja:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} \oplus (1-\gamma) \Delta\theta_t^2.$$

[4]Za ovaj algoritam se predlaže veća inicijalna vrednost stope učenja ($\eta \approx 0.9$).

Sada, kao i kod gradijenata, imamo:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}.$$

Kako je $RMS[\Delta\theta]_t$ u trenutku t nepoznato, ovu vrednost aproksimiramo verdnošću $RMS[\Delta\theta]_{t-1}$. Konačno, smenom inicijalne stope učenja η sa $RMS[\Delta\theta]_{t-1}$ dobijamo pravilo za ažuriranje parametara kod algoritma Adadelta:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} \odot g_t,$$
$$\theta_t = \theta_{t-1} \oplus \Delta\theta_t.$$

Primetimo i to da Adadelta algoritam eleminiše potrebu za definisanjem inicijalne stope učenja.

6 Adam

Adaptive Moment Estimation (ADAM) je takođe jedan od optimizacionih algoritama koji računa adaptivnu stopu učenja za svaki parametar. Pored toga što pamti opadajući prosek kvadrata prethodnih gradijenata (ovde u oznaci v_t) kao Adagrad i Adadelta, Adam takođe pamti i opadajući prosek prethodnih (nekvadriranih) gradijenata (ovde u oznaci m_t), slično kao i Momentum:

$$m_t = \beta_1 m_{t-1} \oplus (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} \oplus (1-\beta_2) g_t^2.$$

m_t i v_t su procene prvog momenta (sredine) i drugog momenta (necentrirane varijanse), respektivno. Kako su m_t i v_t inicijalizovani kao $\mathbf{0}$ vektori, autori [3] su primetili da su oni pristrasni ka nuli, naročito u prvim koracima treniranja, i neminovno ukoliko su stope opadanja (*decay rates*) β_1 i β_2 veoma male[5].

Ove pristrasnosti se ispravljaju sledećim procenama prvog i drugog momenta:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t},$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

Konačno, analogno prethodnim algoritmima, pravilo za ažuriranje parametara u koraku t kod algoritma Adam je:

$$\theta_t = \theta_{t-1} \ominus \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \odot \hat{m}_t.$$

[5]Autori predlažu $\beta_1 = 0.9$ i $\beta_2 = 0.999$.

Reference

[1] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V, Ng, A. Y. (2012). Large Scale Distributed Deep Networks. NIPS 2012: Neural Information Processing Systems, 111. https://doi.org/10.1109/ICDAR.2011.95

[2] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. https://arxiv.org/abs/1212.5701

[3] Diederik Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. https://arxiv.org/abs/1412.6980