Varijante gradijentnog spusta u strojnom učenju s primjerima

Bošnjak, Dorotea

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Dorotea Bošnjak

VARIJANTE GRADIJENTNOG SPUSTA U STROJNOM UČENJU S PRIMJERIMA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Nikola Sandrić

Zagreb, rujan 2023.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Obitelji

Sadržaj

Uvod

Gradijentni spust predstavlja temeljnu tehniku optimizacije u strojnom učenju. Cilj strojnog učenja je pronaći najbolje vrijednosti parametara u modelu koje minimiziraju funkciju gubitka. Taj proces možemo usporediti s navigacijom kroz neravan krajolik potencijalnih rješenja, gdje je cilj stići do najniže doline koja predstavlja optimalno rješenje. Kako su problemi strojnog učenja postajali sve složeniji, pojavila se potreba za optimizacijama osnovnog algoritma gradijentnog spusta poput momentuma ili Adam metode. Algoritmi optimizacije su sve popularniji i korišteniji. Sa željom izbjegavanja njihovog korištenja bez razumijevanja, u ovom ćemo radu proći osnovne pojmove o gradijentnom spustu, pokazati zašto se on tako zove, dati intuiciju o optimizacijskim metodama te proučiti njihove konvergencije.
Dodatno, na primjerima iz prakse ćemo ilustrirati optimizacije gradijentnog spusta. Primjeri se tiču podataka iz medicine i genetike. Vidjet ćemo kakve rezultate daje koja metoda s obzirom na prirodu našeg problema.

Poglavlje 1

Gradijentni spust

Gradijentni spust je optimizacijski algoritam koji se koristi u strojnom i dubokom učenju. To je optimizacijski algoritam prvog reda, što znači da pri modifikaciji parametara uzima u obzir samo prvu derivaciju funkcije gubitka [6]. U svakoj iteraciji modificira parametre u smjeru okomitom na gradijent funkcije gubitka s ciljem pronalaska lokalnog minimuma. Veličinu koraka u svakoj iteraciji zovemo *stopa učenja* i označavamo ju s α. Ukratko, slijedimo smjer padajućeg nagiba dok ne dođemo do lokalnog minimuma.

1.1 Opis algoritma

Za početak, objasnimo ugrubo korake gradijentnog spusta za funkciju gubitka $f : \mathbb{R}^d \to \mathbb{R}$, gdje je $d \in \mathbb{N}$.

Inicijalizacija parametara Na početku određujemo početne vrijednosti parametara $\boldsymbol{w}_0$, koje ćemo ažurirati u svakoj iteraciji. Vrijednosti parametara mogu biti slučajni brojevi.

Određivanje vrijednosti stope učenja U ovom koraku moramo pripaziti da α ne bude premali, jer to uzrokuje predugo vrijeme izvršavanja, niti velik, jer se može dogoditi da metoda ne konvergira ili da preskoči minimum. Stopa učenja se može i ne mora modificirati kroz iteracije. Najčešće vrijednosti u praksi su $\alpha \in \{0.001, 0.003, 0.01, 0.03, 0.1, 0.3\}$.

Normalizacija podataka Ukoliko se skale parametara jako razlikuju, normalizacijom se ubrzava kovergencija.

Računanje vrijednosti gradijenta Parcijalno deriviramo funkciju gubitka u svakoj iteraciji i:

$$\frac{\partial f}{\partial \boldsymbol{w}_i} = \nabla_{\boldsymbol{w}_i} f$$

Gradijent sadržava informacije o smjeru i veličini najstrmijeg porasta funkcije gubitka.

Modifikacija parametara Prilagođavamo parametre u suprotnom smjeru od gradijenta kako bi se funkcija gubitka smanjila. Točnije, oduzimamo α udio gradijenta od svakog parametra. Ažurirane parametre u iteraciji i označimo s $\boldsymbol{w}_{i+1}$:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla_{\boldsymbol{w}_i} f$$

Ponavljanje procesa U svakoj iteraciji računamo vrijednost gradijenta i modificiramo parametare. To činimo za određeni broj iteracija ili dok funkcija gubitka ne konvergira.

Algoritam možemo zapisati u obliku:

Algoritam 1 Algoritam gradijentnog spusta

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\alpha_0 > 0$
for $i = 0, 1, 2, ...$ **do**
1. Izračunamo gradijent funkcije f u točki $\boldsymbol{w}_i$: $\nabla f(\boldsymbol{w}_i)$
2. Ažuriramo stopu učenja $\alpha_i > 0$ (opcionalno)
3. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha_i \cdot \nabla f(\boldsymbol{w}_i)$

end for

1.2 Analiza konvergencije

Postavlja se pitanje zašto vrijedi konvergencija ovog algoritma.

Definicija 1.2.1. *(Lipschitz neprekidnost)*
Funkcija $g : \mathbb{R}^d \to \mathbb{R}^m$ je L-Lipschitz neprekidna na $\mathbb{R}^d$ ako

$$\forall (\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^d \times \mathbb{R}^d \quad ||f(\boldsymbol{u}) - f(\boldsymbol{v})|| \leq L \cdot ||\boldsymbol{u} - \boldsymbol{v}||,$$

gdje se $L > 0$ naziva Lipschitzova konstanta.

Definicija 1.2.2. *(Glatke funkcije s Lipschitz neprekidnim derivacijama)*
Familija $C_L^{1,1}(\mathbb{R}^d)$ je skup svih funkcija $g : \mathbb{R}^d \to \mathbb{R}$ koje pripadaju skupu $C^1(\mathbb{R}^d)$ čiji je gradijent ∇g L-Lipschitz neprekidan.

Pretpostavka 1.2.3. Funkcija gubitka $f : \mathbb{R}^d \to \mathbb{R}$ je iz skupa $C_L^{1,1}(\mathbb{R}^d)$ za neki $L > 0$ i postoji $f_{min} \in \mathbb{R}$ takav da

$$(\forall \boldsymbol{w} \in \mathbb{R}^d) \quad f(\boldsymbol{w}) \geq f_{min}$$

tj. f je ograničena odozdo u $\mathbb{R}^d$.

Pretpostavka 1.2.4. Funkcija gubitka f je konveksna funkcija koja postiže minimum f^* u točki $\boldsymbol{w}^*$. Dakle, grafički prikaz funkcije f je parabola u $\mathbb{R}^d$ s tjemenom u točki $(\boldsymbol{w}^*, f^*)$.

Dokazat ćemo sljedeći teorem:

Teorem 1.2.5. *(Teorem o konvergenciji gradijentnog spusta za konveksne funkcije)*
Neka je f funkcija koja zadovoljava Pretpostavke 1.2.3 i 1.2.4. Pretpostavimo da primjenjujemo Algoritam 1 za $\alpha = \frac{1}{L}$.
Tada za svaki $K \geq 1$, iteracija $\boldsymbol{w}_K$ zadovoljava sljedeće:

$$f(\boldsymbol{w}_K) - f^* \leq O\left(\frac{1}{K}\right),$$

gdje je f^* minimalna vrijednost od f te $O\left(\frac{1}{K}\right)$ označava složenost algoritma gradijentnog spusta u najgorem slučaju.

Prvo pokažimo da funkcija gubitka zaista pada u svakoj iteraciji algoritma gradijentnog spusta.

Propozicija 1.2.6. *Pretpostavimo da smo u k-toj iteraciji Algoritma 1 primjenjenog na funkciju gubitka $f \in C_L^{1,1}(\mathbb{R}^d)$ te neka je $\nabla f(\boldsymbol{w}_k) \neq 0$.*
Tada vrijedi

$$0 < \alpha_k < \frac{2}{L} \implies f(\boldsymbol{w}_k - \alpha_k \nabla f(\boldsymbol{w}_k)) < f(\boldsymbol{w}_k).$$

Specijalno, za $\alpha_k = \frac{1}{L}$ vrijedi

$$f(\boldsymbol{w}_k - \frac{1}{L}\nabla f(\boldsymbol{w}_k)) < f(\boldsymbol{w}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{w}_k)\|^2.$$

Dokaz. U samom početku iskoristimo Taylorov razvoj funkcije, a zatim računamo:

$$\begin{aligned} f(\boldsymbol{w}_k - \alpha_k \nabla f(\boldsymbol{w}_k)) &< f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^T \cdot (-\alpha_k \nabla f(\boldsymbol{w}_k)) + \frac{L}{2} \cdot \| - \alpha_k \nabla f(\boldsymbol{w}_k)\|^2 \\ &= f(\boldsymbol{w}_k) - \alpha_k \cdot \nabla f(\boldsymbol{w}_k)^T \cdot \nabla f(\boldsymbol{w}_k) + \frac{L}{2} \cdot \alpha_k^2 \cdot \|\nabla f(\boldsymbol{w}_k)\|^2 \\ &= f(\boldsymbol{w}_k) + (-\alpha_k + \frac{L}{2} \cdot \alpha_k^2) \cdot \|\nabla f(\boldsymbol{w}_k)\|^2. \end{aligned}$$

Ako želimo da je izraz $f(\boldsymbol{w}_k) + (-\alpha_k + \frac{L}{2} \cdot \alpha_k^2) \cdot \|\nabla f(\boldsymbol{w}_k)\|^2 < f(\boldsymbol{w}_k)$, dovoljno je pokazati sljedeće:

$$\begin{aligned} -\alpha_k + \frac{L}{2} \cdot \alpha_k^2 < 0 \quad &\Leftrightarrow \quad \frac{L}{2} \cdot \alpha_k^2 < \alpha_k \quad / : \alpha_k \quad (\alpha_k > 0) \\ &\Leftrightarrow \quad \frac{L}{2} \cdot \alpha_k < 1 \quad / \cdot \frac{2}{L} \\ &\Leftrightarrow \quad \alpha_k < \frac{2}{L}, \end{aligned}$$

što je pretpostavka propozicije.
Za specijalan slučaj dokaz ide analogno uz uvrštavanje $\alpha_k = \frac{1}{L}$. □

Dokažimo sada Teorem 1.2.5.

Dokaz. Neka je $K \geq 1$ takav da za $k = 0, 1, ..., K-1$ vrijedi $f(\boldsymbol{w}_k) - f(\boldsymbol{w}^*) > \epsilon$. Iz karakterizacije konveksnosti funkcije f dobivamo sljedeći rezultat:

$$f(\boldsymbol{w}^*) \geq f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^T \cdot (\boldsymbol{w}^* - \boldsymbol{w}_k)$$

Iz prethodne nejednakosti i Propozicije 1.2.6. za $\alpha = \frac{1}{L}$ računamo:

$$\begin{aligned} f(\boldsymbol{w}_{k+1}) &= f(\boldsymbol{w}_k - \frac{1}{L}\nabla f(\boldsymbol{w}_k)) \\ &\leq f(\boldsymbol{w}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{w}_k)\|^2 \\ &\leq f(\boldsymbol{w}^*) - \nabla f(\boldsymbol{w}_k)^T \cdot (\boldsymbol{w}^* - \boldsymbol{w}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{w}_k)\|^2 \end{aligned}$$

Primjetimo da vrijedi sljedeće:

$$\begin{aligned} \frac{L}{2}(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_k - \boldsymbol{w}^* - \frac{1}{L}\nabla f(\boldsymbol{w}_k)\|^2) &= \frac{L}{2}(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2 - \|(\boldsymbol{w}_k - \boldsymbol{w}^*) - \frac{1}{L} \cdot \nabla f(\boldsymbol{w}_k)\|^2) \\ &= \frac{L}{2}(\cancel{\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2} - \cancel{\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2} + \frac{2}{L} \cdot \nabla f(\boldsymbol{w}_k)^T(\boldsymbol{w}_k - \boldsymbol{w}^*) - \frac{1}{L^2} \cdot \|\nabla f(\boldsymbol{w}_k)\|^2) \\ &= \nabla f(\boldsymbol{w}_k)^T(\boldsymbol{w}_k - \boldsymbol{w}^*) - \frac{1}{2L} \cdot \|\nabla f(\boldsymbol{w}_k)\|^2, \end{aligned}$$

što je upravo pribrojnik u zadnjoj gornjoj nejednakosti.
Sada imamo

$$\begin{aligned} f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}^*) &\leq \frac{L}{2}(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_k - \boldsymbol{w}^* - \frac{1}{L}\nabla f(\boldsymbol{w}_k)\|^2) \\ &= \frac{L}{2}(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\|^2). \end{aligned}$$

Sumirajmo izraz $f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}^*)$ po $k = 0, 1, ..., K-1$ te omeđimo tu sumu:

$$\begin{aligned}\sum_{k=0}^{K-1}(f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}^*)) \leq \frac{L}{2} \cdot (&\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2 - \cancel{\|\boldsymbol{w}_1 - \boldsymbol{w}^*\|^2} + \cancel{\|\boldsymbol{w}_1 - \boldsymbol{w}^*\|^2} + \\ &- \cancel{\|\boldsymbol{w}_2 - \boldsymbol{w}^*\|^2} + ... + \cancel{\|\boldsymbol{w}_{K-1} - \boldsymbol{w}^*\|^2} - \|\boldsymbol{w}_K - \boldsymbol{w}^*\|^2) \\ &= \frac{L}{2} \cdot (\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_K - \boldsymbol{w}^*\|^2) \\ &\leq \frac{L}{2} \cdot \|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2\end{aligned}$$

S druge strane, iz Propozicije 1.2.6. je jasno da vrijedi niz nejednakosti

$$f(\boldsymbol{w}_0) \geq f(\boldsymbol{w}_1) \geq ... \geq f(\boldsymbol{w}_K).$$

Iz njih dobivamo sljedeći rezultat:

$$\sum_{k=0}^{K-1}(f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}^*)) \geq K \cdot (f(\boldsymbol{w}_K) - f(\boldsymbol{w}^*))$$

Dakle, imamo ogradu sume s lijeve i desne strane

$$K \cdot (f(\boldsymbol{w}_K) - f(\boldsymbol{w}^*)) \leq \sum_{k=0}^{K-1}(f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}^*)) \leq \frac{L}{2} \cdot \|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2$$

Konačno,

$$f(\boldsymbol{w}_K) - f(\boldsymbol{w}^*) \leq \frac{L \cdot \|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2}{2} \cdot \frac{1}{K}$$

iz čega slijedi tvrdnja teorema. □

1.3 Primjer

Pokažimo na jednostavnom primjeru funkcije kako gradijentni spust pronalazi minimum. Uzmimo funkciju $f : \mathbb{R} \to \mathbb{R}$

$$f(x) = 3x^2 - 5x + 8$$

Računamo parcijalne derivacije:

$$\frac{\partial f}{\partial x}(x) = 6x - 5, \quad \frac{\partial f^2}{\partial x}(x) = 6 > 0$$

Uočavamo da je funkcija konveksna. Izračunajmo ručno minumum ove funkcije.

$$x^* = \frac{-b}{2a} = 0.83, \quad y^* = \frac{4ac - b^2}{4a} \approx 5.92$$

Uzmimo $\alpha = 0.1$ i $x_0 = 10$ te izračunajmo ručno prvih par iteracija.
$x_1 = x_0 - \alpha \nabla f(x_0) = 10 - 0.1 \cdot (6 \cdot 10 - 5) = 4.5$
$x_2 = x_1 - \alpha \nabla f(x_1) = 4.5 - 0.1 \cdot (6 \cdot 4.5 - 5) = 2.3$
$x_3 = x_2 - \alpha \nabla f(x_2) = 2.3 - 0.1 \cdot (6 \cdot 2.3 - 5) = 1.42$
$x_4 = x_3 - \alpha \nabla f(x_3) = 1.42 - 0.1 \cdot (6 \cdot 1.42 - 5) = 1.068$
$x_5 = x_4 - \alpha \nabla f(x_4) = 1.068 - 0.1 \cdot (6 \cdot 1.068 - 5) = 0.9272$
Vidimo da je svakom iteracijom algoritam sve bliže točki minimuma $x^* = 0.83$.

Kako ne bismo ručno računali ostale iteracije, u Pythonu provedimo gradijentni spust s početnom točkom x_0, stopom učenja α, najvećim mogućim brojem iteracija 100 i tolerancijom 0.001. Python kod je dostupan u Dodatku A na kraju rada.
Algoritam daje sljedeće vrijednosti od x po iteracijama i:

i	0	1	2	3	4	5	6	7	8	9	10
x_i	10	4.5	2.3	1.42	1.068	0.927	0.871	0.848	0.839	0.836	0.834

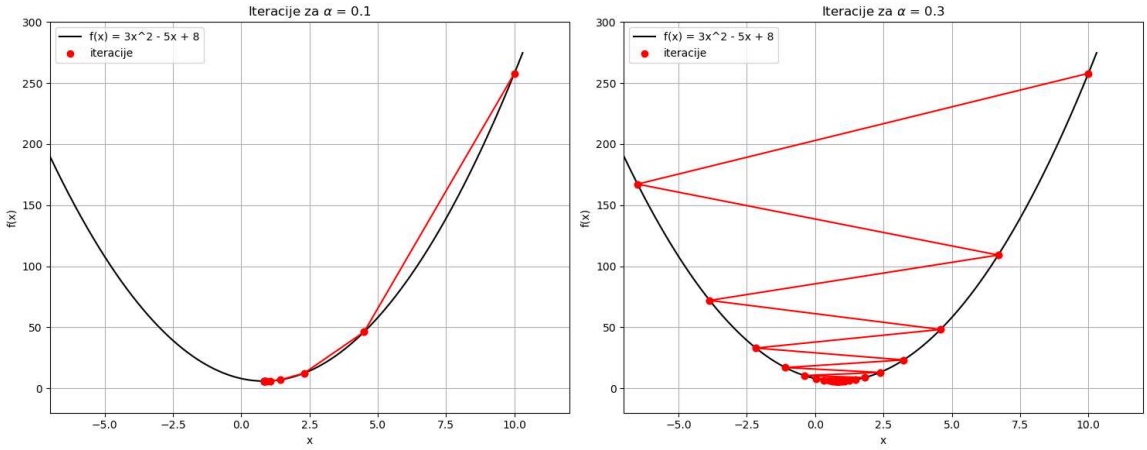
Output algoritma nam potvrđuje točnost ručno izračunatih iteracija te uočavamo da se zaista radi o gradijentnom *spustu*. Primjetimo da je uz ove vrijednosti početnih parametara bilo potrebno 10 iteracija da se algoritam izvrši.

Nadalje, kako bismo bolje razumjeli utjecaj stope učenja na algoritam, probajmo sljedeće: sve parametre ostavimo istima kao u prvom slučaju, osim stope učenja koju postavljamo na $\boldsymbol{\alpha = 0.3}$.
Output poziva ove varijante gradijentnog spusta daje rezultate:

i	0	1	2	3	4	5	6	...	42	43	44
x_i	10	-6.5	6.68	-3.86	4.588	-2.170	3.236	...	0.834	0.833	0.834

Bile su potrebne 44 iteracije za izvršenje ovog algoritma. Ako pogledamo vrijednosti x-eva, primjećujemo da se događa gradijentni *spust*, ali da je za male i norma gradijenta od f dosta veća nego za iste vrijednosti i u prvom slučaju.

Pogledajmo kako grafički izgledaju koraci ovih dvaju algoritama. Crvene linije povezuju točke dobivene iz dvije uzastopne iteracije.



Slika 1.1: Utjecaj stope učenja na gradijentni spust

Dakle, uočavamo da je u slučaju manjeg $\alpha = 0.1$, algoritam pronašao zadovoljavajući minimum u 10 iteracija, dok je za $\alpha = 0.3$ u svakoj iteraciji preskočio minimum te su mu je bile potrebne 44 iteracije da dođe do minimuma. S druge strane, minimum pronađen u drugom slučaju je nešto točniji.
Naslućujemo da se događa kompromis između računalne složenosti algoritma i točnosti pronađenog minimuma.

1.4 Dodavanje akceleracije

Pokazani rezultat o brzini konvergencije gradijentnog spusta konveksnih funkcija se naziva *gornjom granicom složenosti*. Pokazalo se da je moguće ubrzati algoritam. Osnovna ideja ubrzanja je da ako se u trenutnoj iteraciji uzmu u obzir dostupne informacije o prethodnima, može se napraviti korak koji je bolji od onoga koji nudi trenutni gradijent. Neke od ubrzanih metoda su momentum i Nesterov ubrzani gradijent. Kako točno Nesterova metoda radi, možemo vidjeti u Algoritmu 2. Slično kao u gradijentnom spustu, potrebno je odrediti jedan gradijent po iteraciji. Modifikacija se događa u računanju koraka iteracije, gdje se uzima u obzir prethodni.

Nesterov algoritam možemo opisati kao:

Algoritam 2 Algoritam Nesterovog ubrzanog gradijenta

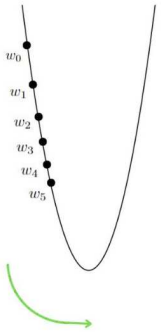
Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$

for $i = 0, 1, 2, ...$ **do**

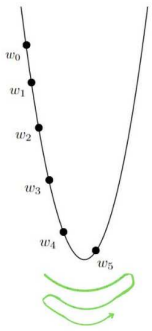
1. Izračunamo stopu učenja $\alpha_i > 0$ i parametar $\beta_i > 0$.
2. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha_i \cdot \nabla f(\boldsymbol{w}_i + \beta_i \cdot (\boldsymbol{w}_i - \boldsymbol{w}_{i-1})) + \beta_i \cdot (\boldsymbol{w}_i - \boldsymbol{w}_{i-1})$.

end for

Intuitivno shvaćanje dodavanja akceleracije mogu dočarati sljedeće slike i "*heavy ball*" metoda kao drugi naziv za momentum metodu. Može se reći da gradijentni spust ima pristup s lijeva, dok se akcelerirani algoritam kreće oko minumuma kao kada bismo pustili tešku kuglu na nizbrdici u dolinu. Kugli tada gravitacija daje ubrzanje te ona preskače najnižu točku nekoliko puta dok ju ne pronađe. Zelene strijelice dočaravaju putanju pojedine metode.



Slika 1.2: Metoda gradijentnog spusta



Slika 1.3: "Heavy ball" metoda

Iskažimo sada teorem o brzini konvergencije za akcelerirani gradijentni spust u slučaju kada je $f \in C_L^{1,1}(\mathbb{R}^d)$ za neki $L > 0$ i ograničena odozdo u $\mathbb{R}^d$.

Teorem 1.4.1. *Neka je Algoritam 2 primjenjen na funkciju f s gornjim pretpostavkama uz $\alpha = \frac{1}{L}$ i $\epsilon > 0$.*
Tada za svaki $K \geq 1$, parametar $\boldsymbol{w}_K$ zadovoljava

$$f(\boldsymbol{w}_K) - f^* \leq O\left(\frac{1}{K^2}\right).$$

Dodatno, ako je f konveksna, tada vrijedi $\beta_k = \frac{t_k - 1}{t_{k+1}}$, $t_{k+1} = \frac{1}{2} \cdot \left(1 + \sqrt{1 + 4t_k^2}\right)$ te $t_0 = 0$.

Možemo primijetiti da je brzina konvergencije uvelike poboljšana u ubrzanom gradijentnom spustu, tj. $O\left(\frac{1}{K^2}\right)$ je znatno bolja složenost algortima od $O\left(\frac{1}{K}\right)$.
Više detalja o pojedinim varijantama ubrzanog gradijentnog spusta slijedi u nastavku rada.

Poglavlje 2

Varijante gradijentnog spusta

Postoje različite varijante gradijentnog spusta. Glavna razlika između sljedećih varijanti je količina podataka za trening koju koristimo pri računanju gradijenta u jednoj iteraciji. Uvodimo pojam **epohe** koja predstavlja jedan prolazak kroz cjelokupni skup podataka za treniranje modela. Dok postoji varijanta gradijentnog spusta koja u jednoj iteraciji prolazi kroz sve podatke za treniranje, vidjet ćemo da postoje i druge varijante koje u jednoj iteraciji uzimaju samo podskup podataka za treniranje. To doprinosi smanjenju računalne složenosti te takvi modeli upravo potvrđuju prethodnu slutnju o kompromisu između točnosti gradijenta i vremenske složenosti za izvođenje ažuriranja svakog parametra [7].

2.1 Gradijentni spust s paketima

U gradijentnom spustu s paketima se u svakom koraku uzimaju u obzir svi podaci iz skupa za treniranje. Uzimamo prosjek gradijenata te pomoću njega ažuriramo parametre za sljedeću iteraciju. Ovo je slučaj u kojem je iteracija ekvivalentna epohi.

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla_{\boldsymbol{w}_i} f$$

Algoritam za gradijentni spust s paketima izgleda ovako:

Algoritam 3 Algoritam gradijentnog spusta s paketima

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\alpha > 0$, $i_{max} \in \mathbb{N}$, $tol > 0$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$

for $i = 0, 1, 2, ...$ **do**

1. Provjerimo je li $i \leq i_{max}$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
2. Provjerimo je li $\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\| > tol$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
3. Izračunamo prosijek gradijenata funkcije f u svim točkama iz skupa za treniranje: $\nabla f(\boldsymbol{w}_i)$.
4. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \cdot \nabla f(\boldsymbol{w}_i)$.

end for

- *Prednosti:* Gradijentni spust s paketima ima nepristrani procjenitelj gradijenata. Što je veći skup podataka za trening, to je standardna greška manja. Također, pogodan je za konveksne funkcije ili funkcije sa stabilnom greškom. U tom se slučaju krećemo direktno prema optimalnom rješenju, što često znači brza konvergencija.
- *Mane:* Može se dogoditi da nemamo brzu konvergenciju u slučaju kada je skup podataka za treniranje jako velik jer moramo pohraniti puno podataka u memoriju. Također, skup podataka za treniranje može sadržavati točke koje su suvišne u procesu računanja gradijenta, što čini cijeli proces bespotrebno složenijim. Nadalje, može se dogoditi da točka konvergencije nije najbolja, tj. postiže se lokalni minimum umjesto globalni.

2.2 Stohastički gradijentni spust

Umjesto prolaženja kroz sve točke u jednoj iteraciji, stohastički gradijentni spust (SGD) ažurira parametre s obzirom na jednu slučajnu točku $(\hat{\boldsymbol{x}}_i, \hat{y}_i)$. Dakle, učenje se događa za svaku točku iz skupa za treniranje.

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla f(\hat{\boldsymbol{x}}_i, \hat{y}_i)$$

Algoritam za stohastički gradijentni spust ima sljedeći oblik:

Algoritam 4 Algoritam za stohastički gradijentni spust

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\alpha > 0$, $i_{max} \in \mathbb{N}$, $tol > 0$, $i = 0$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$
while 1 **do**
1. Izmiješamo redosljed podataka za trening τ te dobijemo skup $\hat{\tau}$.
while $\hat{\tau} \neq \emptyset$ **do**
2. Provjerimo je li $i \leq i_{max}$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
3. Provjerimo je li $\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\| > tol$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
4. Na slučajan način izaberemo točku $(\hat{\boldsymbol{x}}_i, \hat{y}_i) \in \hat{\tau}$.
5. Izračunamo gradijent funkcije f u toj točki: $\nabla f(\hat{\boldsymbol{x}}_i, \hat{y}_i)$.
6. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \cdot \nabla f(\hat{\boldsymbol{x}}_i, \hat{y}_i)$.
7. Izbacimo točku $(\hat{\boldsymbol{x}}_i, \hat{y}_i)$ iz skupa za treniranje $\hat{\tau}$.
8. Povećamo broj napravljenih iteracija $i = i + 1$.
end while
end while

- *Prednosti:* Stohastički gradijentni spust zahtjeva manje memorije od gradijentnog spusta s paketima, jer je u iteraciji potrebno zadržati samo jednu točku treniranja. Nadalje, slučajan odabir točke u iteraciji pomaže pri izbjegavanju računanja gradijenta u sličnim točkama koje u paru nemaju velik utjecaj na učenje.

- *Mane:* U ovom slučaju imamo sporu konvergenciju. Gradijent se temelji na jednoj točki što je uzrok velike varijance gradijenta. Dakle, gradijent kroz iteracije varira pa se to događa i oko minimuma, ali nam to s druge strane može pomoći u izlasku iz lokalnog minimuma i pronalaženju globalnog. Iako češća ažuriranja mogu pružiti brže informacije, to može rezultirati gubicima u računalnoj učinkovitosti u usporedbi s gradijentnim spustom s paketima.

2.3 Gradijentni spust s mini-paketima

Gradijentni spust s mini-paketima je kombinacija spusta s paketima i stohastičkog spusta. U ovom slučaju se gradijenti računaju na slučajnom podskupu točaka za trening koje nazivamo mini-paketima. Korak pomaka u iteraciji je prosjek gradijenata točaka iz jednog mini-paketa skaliran s α. Ako skup za treniranje podijelimo na m mini-paketa veličine b, učenje se opisuje kao:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla f(x_i^{(1:b)}, y_i^{(1:b)}).$$

Broj i veličinu mini-paketa možemo prilagoditi podacima. Najčešće je veličina potencija broja 2 jer neka računala brže procesuiraju takve veličine paketa.
Koraci gradijentnog spusta s mini-paketima izgledaju ovako:

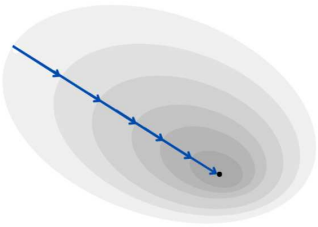
Algoritam 5 Algoritam gradijentnog spusta s mini-paketima

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\alpha > 0$, $i_{max} \in \mathbb{N}$, $tol > 0$, $i = 0$, $m \in \mathbb{N}$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$
while 1 **do**
1. Izmiješamo redosljed podataka za trening τ te dobijemo skup $\hat{\tau}$.
2. Skup $\hat{\tau}$ podijelimo na m mini-paketa $\hat{\mu}_j$, $j = 1, 2, ..., m$.
for $j = 1, 2, ..., m$ **do**
3. Provjerimo je li $i \leq i_{max}$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
4. Provjerimo je li $\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\| > tol$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
5. Na slučajan način izaberemo mini-paket $\hat{\mu}_i$.
6. Izračunamo prosjek gradijenata funkcije f u svim točkama iz mini-paketa $\hat{\mu}_j$: $\nabla f(\hat{\mu}_i)$.
7. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \cdot \nabla f(\hat{\mu}_i)$.
8. Izbacimo skup $\hat{\mu}_i$ iz familije mini-paketa za treniranje $\hat{\tau}$.
9. Povećamo broj napravljenih iteracija $i = i + 1$.
end for
end while

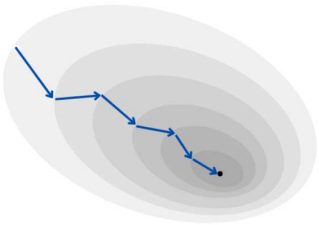
- *Prednosti:* Gradijentni spust s mini-paketima povećava računalnu učinkovitost u odnosu na gradijentni spust s paketima, dok smanjuje varijancu ažuriranja u usporedbi sa SGD-om.
- *Mane:* Kao i kod SGD-a, gradijentni spust lokalno varira jer u iteraciji uzimamo u obzir manji broj točaka.

2.4 Usporedba varijanti

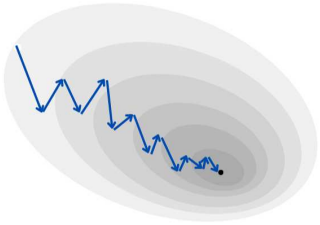
Sljedeće tri slike predstavljaju graf konveksne funkcije te predočavaju kako izgleda gradijentni spust za svaku od ove tri varijante. Plavi vektori predstavljaju smjer kretanja kroz iteracije u cilju pronalaska minimuma funkcije označenog crnom točkom.



Slika 2.1: Gradijenti spust s paketima



Slika 2.2: Gradijenti spust s mini-paketima



Slika 2.3: Stohastički gradijenti spust

Možemo primjetiti kako gradijentni spust s paketima ima najbržu konvergenciju te je ona sve sporija što smo bliže SGD-u. Sada je i vizualno jasno da bi smjerovi vektora sve više varirali kada bismo povećavali broj mini-paketa na koje dijelimo skup za treniranje, dok ne bismo došli do stohastičkog gradijentnog spusta.

Ne možemo reći da je jedna varijanta bolja od druge niti da se u primjeni češće koristi. Ovisno o podacima i prirodi problema, odabiremo koja je varijanta najprikladnija.

2.5 Izazovi

Gradijentni spust s mini-paketima je kompromis dvije varijacije te posjeduje njihova pozitivna svojsta, ali i dalje postoje problemi pri provođenju algoritma. Među njima je već spomenuti osjetljivi odabir stope učenja α koja ne mora nužno biti fiksna. Treba pažljivo odrediti stope učenja u iteracijama, što može biti izazovno u praksi.
Nadalje, problem mogu predstavljati spora konvergencija ili velike oscilacije u algoritmu kada se smjer gradijenta brzo mijenja.
Još jedan izazov predstavljaju nekonveksne funkcije, tj. izbjegavanje upadanja u lokalne minimume. Postoje istraživanja koja tvrde da su sedlaste točke još veći problem. One su obično okružene ravninom, što predstavlja problem za stohastički gradijentni spust jer je gradijent tada blizu nule u svim dimenzijama.

Poglavlje 3

Optimizacijski algoritmi gradijentnog spusta

U nastavku ćemo opisati neke algoritme koji se koriste u strojnom i dubokom učenju kao rješenja za spomenute izazove.
Navedimo ukratko ideje metoda u nastavku:

- dodati akceleraciju u algoritam akumulirajući prošle gradijente kako bismo napravili veće korake u pravom smjeru
- dovesti do stabilnijih putanja konvergencije
- ublažiti potrebu za ručnim prilagodbama stope učenja u iteracijama
- pomoći pri učinkovitijem izlasku iz lokalnih minimuma ili sedlastih točaka
- smanjiti šum konzistentnijim ažuriranjima parametara

3.1 Momentum gradijentni spust

Momentum gradijentni spust je varijacija standardnog gradijentnog spusta koja daje ubrzanje algoritmu kako bi poboljšala brzinu i stabilnost konvergencije. To postiže dodavanjem *momentum parametra* γ koji određuje koliko će prošla iteracija utjecati na trenutnu. To se radi kako bi se zadržala brzina ažuriranja iz prethodnih koraka.

$$\boldsymbol{v}_{i+1} = \gamma \cdot \boldsymbol{v}_i + \alpha \nabla_{\boldsymbol{w}_i} f$$

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \boldsymbol{v}_{i+1},$$

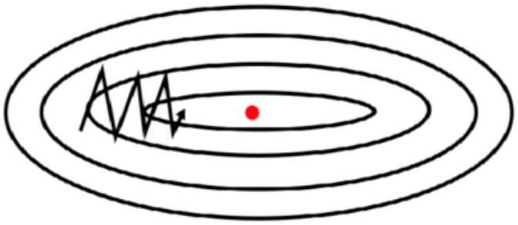
gdje je $\boldsymbol{v}_i$ brzina u iteraciji i. Momentum γ je najčešće vrijednosti 0.9 ili slično.

Pogledajmo kako izgleda algoritam SGD-a s metodom momentuma.

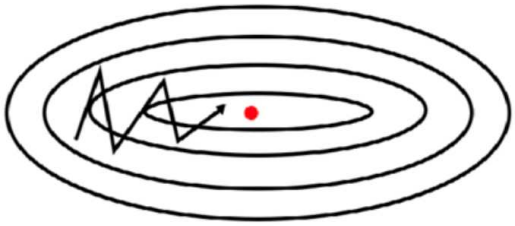
Algoritam 6 Algoritam za SGD s momentumom

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $v_0 = 0$, $\alpha > 0$, $\gamma \in [0, 1]$, $i_{max} \in \mathbb{N}$, $tol > 0$, $i = 0$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$
while 1 **do**
1. Izmiješamo redosljed podataka za trening τ te dobijemo skup $\hat{\tau}$.
while $\hat{\tau} \neq \emptyset$ **do**
2. Provjerimo je li $i \leq i_{max}$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
3. Provjerimo je li $\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\| > tol$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
4. Na slučajan način izaberemo točku $(\hat{\boldsymbol{x}}_i, \hat{y}_i) \in \hat{\tau}$.
5. Izračunamo gradijent funkcije f u toj točki: $\nabla f(\hat{\boldsymbol{x}}_i, \hat{y}_i)$.
6. Izračunamo momentum vektor brzine $\boldsymbol{v}_{i+1} = \gamma \cdot v_i + \alpha \cdot \nabla f(\hat{\boldsymbol{x}}_i, \hat{y}_i)$.
7. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \boldsymbol{v}_{i+1}$.
8. Izbacimo točku $(\hat{\boldsymbol{x}}_i, \hat{y}_i)$ iz skupa za treniranje $\hat{\tau}$.
9. Povećamo broj napravljenih iteracija $i = i + 1$.
end while
end while

Primjetimo da je razlika između Algoritma 6 u odnosu na Algoritam 4 u 6. koraku gdje uvodimo vektor brzine kao linearnu kombinaciju vektora brzine iz prethodne iteracije i gradijenta od f.
Na sljedećim grafovima možemo vidjeti što momentum zaista napravi gradijentnom spustu. Primjećujemo da je konvergencija uistinu brža s momentumom.



Slika 3.1: SGD bez momentuma



Slika 3.2: SGD s momentumom

3.2 Nesterov ubrzani gradijent

Nesterov ubrzani gradijent (NAG) je razvio slovački matematičar Yurii Nesterov. Glavni cilj algoritma je smanjenje oscilacija u brzini konvergencije, posebno blizu minimuma funkcije gubitka. U svakom koraku ažuriranja, algoritam prvo primjenjuje korak unaprijed, odnosno privremeno ažurira parametare u smjeru prethodnog momentum ažuriranja. Zatim se računa gradijent funkcije troška na temelju tog privremenog ažuriranja.

$$\boldsymbol{v}_{i+1} = \gamma \cdot \boldsymbol{v}_i + \alpha \nabla f(\boldsymbol{w}_i - \gamma \cdot \boldsymbol{v}_i)$$

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \boldsymbol{v}_{i+1}$$

Pogledajmo kako NAG algoritam izgleda:

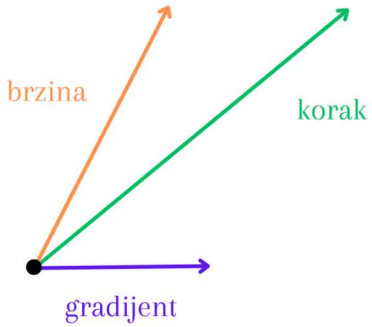
Algoritam 7 Algoritam za Nesterov ubrzani gradijentni spust

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $v_0 = 0$, $\alpha > 0$, $\gamma \in [0, 1]$, $i_{max} \in \mathbb{N}$, $tol > 0$, $i = 0$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$
while 1 **do**
1. Provjerimo je li $i \leq i_{max}$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
2. Provjerimo je li $\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\| > tol$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
3. Izračunamo momentum vektor brzine $\boldsymbol{v}_{i+1} = \gamma \cdot \boldsymbol{v}_i + \alpha \nabla f(\boldsymbol{w}_i - \gamma \cdot \boldsymbol{v}_i)$.
4. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \boldsymbol{v}_{i+1}$.
5. Povećamo broj napravljenih iteracija $i = i + 1$.

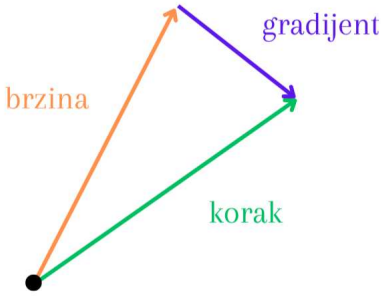
end while

3.3 Usporedba momentuma i Nesterova

Razlika između ove dvije metode je jasno prikazana na sljedećim slikama. Na njima se nalazi prikaz vektora pomaka za jednu iteraciju. Uočavamo da se razlika krije u gradijentu, odnosno u kojoj se točki on računa. Vidimo da je gradijent na slici 3.4 ipak drugačiji vektor od onog na slici 3.3, tj. $\nabla f(\boldsymbol{w}_i) \neq \nabla f(\boldsymbol{w}_i - \gamma \cdot \boldsymbol{v}_i)$. Nesterova metoda, osim ubrzanja, ipak i usmjerava algoritam u pravom smjeru.



Slika 3.3: Metoda momentum



Slika 3.4: Metoda Nesterov

3.4 Primjer (nastavak)

Kao što smo ranije pogledali algoritam gradijentnog spusta na primjeru funkcije

$$f(x) = 3x^2 + 5x + 8,$$

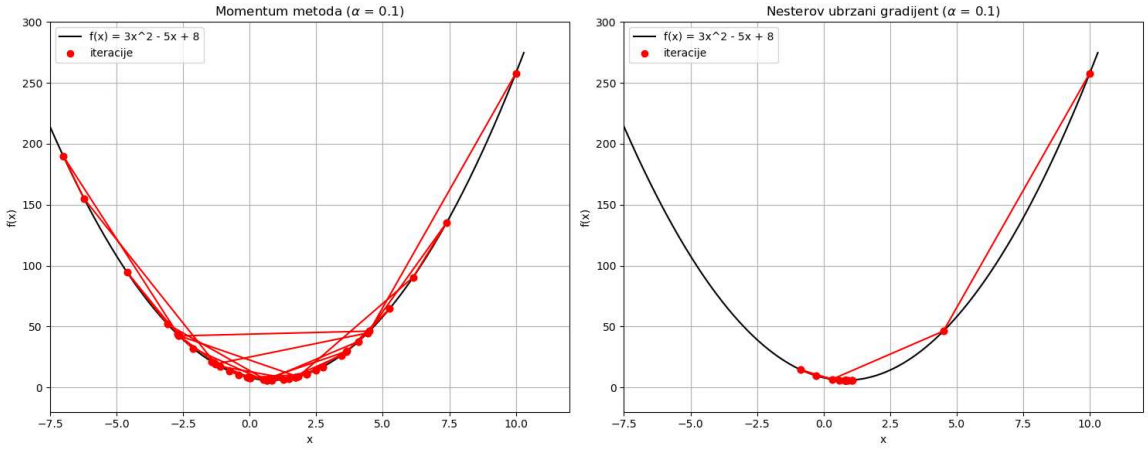
pogledajmo sada kako bi izgledale ubrzane metode na ovom primjeru.
Za početak, pogledajmo momentum metodu. Kod dosputan u Dodatku A daje rezultate:

i	0	1	2	3	4	5	6	...	38	39	40
x_i	10	4.5	-2.65	-6.995	-6.209	-1.276	4.429	...	2.148	1.744	0.834

Sada provedimo Nesterov ubrzani algoritam. Kod je dostupan u Dodatku A. On daje rezultate:

i	0	1	2	3	4	5	6	...	13	14	15
x_i	10	4.5	0.32	-0.877	-0.282	0.602	1.059	...	0.839	0.840	0.837

Analogno kao ranije, grafički usporedimo razlike ovih dviju metoda.



Slika 3.5: Usporedba dviju ubrzanih metoda za gradijentni spust

Uočavamo da je Nesterova metoda efikasnija od momentuma jer završi u 15 iteracija, dok momentum metoda zahtijeva 40 iteracija.
Nadalje, ako usporedimo ove dvije ubrzane metode s osnovnim gradijentnim spustom (slučaj kada je $\alpha = 0.1$ sa Slike 1.3.), vidimo da je za konkretno ovaj problem početna metoda ipak najbolja. Prednosti ubrzanih metoda dolaze do izražaja pri kompleksnijim i robusnijim problemima u više dimenzija.

Sada kada smo u mogućnosti prilagoditi naša ažuriranja nagibu naše funkcije gubitka i ubrzati SGD, željeli bismo također prilagoditi naša ažuriranja svakom pojedinom parametru ovisno o njihovoj važnosti.

3.5 Adagrad

Adagrad (adaptive gradient algorithm) je optimizacijski algoritam koji prilagođava stope učenja za svaki parametar pojedinačno. To čini na način da parametri koji imaju veće gradijente imaju manje stope učenja, dok parametri s manjim gradijentima imaju veće stope učenja. Ovaj pristup pomaže da algoritam automatski prilagodi brzine učenja ovisno o njihovoj važnosti i dinamici te je posebno koristan za probleme s raspršenim podacima.
Ranije smo u iteraciji ažurirali za sve parametre $\boldsymbol{w}_i$ istovremeno, jer je svaki parametar koristio istu stopu učenja α. Budući da Adagrad koristi različite stope učenja za svaki parametar u svakoj iteraciji i, prvo odredimo Adagradovo ažuriranje po parametru, koje potom vektoriziramo.

Prije nego počnemo uvoditi nove oznake, iskažimo formulu po kojoj se ažuriraju parametri Adagrad metodom:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \frac{\alpha}{\sqrt{\epsilon I + G_i}} \times g_i,$$

gdje $\sqrt{\epsilon I + G_i}$ označava matricu $\epsilon I + G_i$ kojoj korjenujemo svaki element posebno te "$\times$" označava matrično množenje.

Pokažimo kako doći do gornje formule.

Zbog lakšeg snalaženja, označimo s $g_{i,k}$ gradijent funkcije gubitka u odnosu na parametar w_k za $k \in 1, 2, ..., d$, u iteraciji i:

$$g_{i,k} = \nabla f(w_{i,k}).$$

Sada SGD ažuriranje za svaki parametar w_k za svaku iteraciju i možemo zapisati kao:

$$w_{i+1,k} = w_{i,k} - \alpha \cdot g_{i,k}$$

Adagrad algoritam modificira opću stopu učenja α u svakoj iteraciji i za svaki parametar w_k na temelju prošlih gradijenata koji su izračunati za w_k:

$$w_{i+1,k} = w_{i,k} - \frac{\alpha}{\sqrt{\epsilon + G_{i,kk}}} \cdot g_{i,k},$$

gdje je $G_i \in \mathbb{R}^{d \times d}$ dijagonalna matrica čiji je dijagonalni element na mjestu (k, k) zbroj kvadrata gradijenata u odnosu na w_k do iteracije i, dok pribrojnik $\epsilon > 0$ spriječava dijeljenje s nulom. Zanimljivo je da algoritam znatno gore radi bez operacije kvadratnog korijena. Pošto dijagonala od G_i sadržava zbroj kvadrata prošlih gradijenata u odnosu na sve parametre w_k, možemo vektorizirati našu implementaciju množenjem G_i i g_i, čime dobivamo konačnu formulu ažuriranja:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \frac{\alpha}{\sqrt{\epsilon I + G_i}} \times g_i.$$

Zapišimo Adagrad u obliku algoritma:

Algoritam 8 Adagrad algoritam

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\alpha > 0$, $\epsilon > 0$ mali, $i_{max} \in \mathbb{N}$, $tol > 0$, $i = 0$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$

while 1 **do**

1. Provjerimo je li $i \leq i_{max}$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
2. Provjerimo je li $\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\| > tol$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
3. Izračunamo $g_i = [g_{i,k}]_{k\in\{1,2,...,d\}}$.
4. Izračunamo dijagonalne elemente matrice $G_i \in \mathbb{R}^{d\times d}$:
   $G_{i,kk} = \sum_{j=0}^{i} g_{j,k}^2, \quad k \in \{1, 2, ..., d\}$
5. Izračunamo vrijednost $\frac{\alpha}{\sqrt{\epsilon I + G_i}}$, gdje je $\epsilon > 0$ t.d. $\sqrt{\epsilon I + G_i} \neq 0$
6. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \frac{\alpha}{\sqrt{\epsilon I + G_i}} \times g_i$.
7. Povećamo broj napravljenih iteracija $i = i + 1$.

end while

Prednost Adagrada je što ne postoji potreba za ručnim podešavanjem stope učenja α. Većina implementacija koristi zadane vrijednosti, npr. $0,01$, i ostavlja ih nepromijenjenima. Adagrad se prilagođava dinamici gradijenata za svaki parametar.

Glavna mana Adagrada je akumulacija kvadratnih gradijenata u nazivniku. Budući da je svaki dodani član pozitivan, akumulirani zbroj raste s iteracijama. To uzrokuje smanjenje stope učenja koja je u konačnici skoro nula, što predstavlja trenutak kada algoritam više nije u stanju steći dodatno znanje. U nastavku ćemo upoznati algoritme kao što su RMSProp i Adam, koji imaju cilj riješiti ovo usporavanje učenja.

3.6 Adadelta

Adadelta je proširenje Adagrada koje pokušava ublažiti njegovu opadajuću stopu učenja α. Ključna ideja Adadelta algoritma je da se umjesto korištenja apsolutnih kvadratnih akumulacija gradijenata kao u Adagradu, koriste relativne promjene između uzastopnih koraka. Ovo omogućava algoritmu da se prilagodi promjenjivoj dinamici optimizacije. Umjesto akumuliranja svih prethodnih kvadratnih gradijenata, Adadelta ograničava broj prethodnih gradijenata koji se akumuliraju. Označimo taj broj sa c.

Umjesto pohranjivanja c prethodnih kvadratnih gradijenata, suma gradijenata je rekurzivno definirana kao opadajući prosjek svih prethodnih kvadratnih gradijenata. Prosjek $E[g^2]_i$ u iteraciji i ovisi samo o prethodnom prosjeku i trenutnom gradijentu:

$$E[g^2]_i = \gamma E[g^2]_{i-1} + (1 - \gamma) g_i^2,$$

gdje je γ kao kod Momentum metode. Zapišimo ponovno ažuriranje SGD-a u terminima vektora ažuriranja parametara $\Delta \boldsymbol{w}_i := \boldsymbol{w}_{i+1} - \boldsymbol{w}_i$:

$$\Delta \boldsymbol{w}_i = -\alpha \cdot g_{i,k}$$

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \Delta \boldsymbol{w}_i$$

Vektor ažuriranja parametara Adagrada koji smo prethodno izveli sada ima sljedeći oblik:

$$\Delta \boldsymbol{w}_i = -\frac{\alpha}{\sqrt{\epsilon I + G_i}} \times g_i.$$

Sada zamjenjujemo dijagonalnu matricu G_i s opadajućim prosjekom prethodnih kvadratnih gradijenata $E[g^2]_i$:

$$\Delta \boldsymbol{w}_i = -\frac{\alpha}{\sqrt{\epsilon + E[g^2]_i}} g_i.$$

Budući da je nazivnik korijen srednje kvadratne pogreške gradijenta, možemo ga zamijeniti kraticom RMS:

$$\Delta \boldsymbol{w}_i = -\frac{\alpha}{RMS\,[g]_i} g_i.$$

Nadalje, kako bismo mogli povući paralelu Adadelte s drugim varijantama (kao što su SGD, Momentum, Adragrad), definirajmo još jedan eksponencijalno opadajući prosjek, ovaj put ne kvadratnih gradijenata, već kvadratnih ažuriranja parametara:

$$E[\Delta \boldsymbol{w}^2]_i = \gamma E[\Delta \boldsymbol{w}^2]_{i-1} + (1-\gamma)\Delta \boldsymbol{w}_i^2.$$

Korijen srednje kvadratne greške ažuriranja parametara je sada:

$$RMS\,[\Delta \boldsymbol{w}]_i = \sqrt{\epsilon + E[\Delta \boldsymbol{w}^2]_i}.$$

Budući da $RMS\,[\Delta \boldsymbol{w}]_i$ nije poznat, aproksimiramo ga RMS-om ažuriranja parametara do prethodne iteracije. Zamjena stope učenja α u prethodnom pravilu ažuriranja s $RMS\,[\Delta \boldsymbol{w}]_{i-1}$ konačno daje pravilo ažuriranja Adadelte:

$$\Delta \boldsymbol{w}_i = -\frac{RMS\,[\Delta \boldsymbol{w}]_{i-1}}{RMS\,[g]_i} g_i$$

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \Delta \boldsymbol{w}_i$$

Zapišimo kako izgleda Adadelta algoritam [8]:

Algoritam 9 Adadelta algoritam

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\gamma \in [0,1]$, $\epsilon > 0$ mali, $i_{max} \in \mathbb{N}$, $tol > 0$, $i = 0$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$

0. Postavimo da je $E[g^2]_0 = 0$, $E[\Delta \boldsymbol{w}^2]_0 = 0$.

while 1 **do**

1. Provjerimo je li $i \leq i_{max}$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
2. Provjerimo je li $\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\| > tol$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
3. Izračunamo gradijent $g_i = [g_{i,k}]_{k\in\{1,2,\ldots,d\}}$, gdje je $g_{i,k} = \nabla f(\boldsymbol{w}_{i,k})$.
4. Akumuliramo gradijent $E[g^2]_i = \gamma \cdot E[g^2]_{i-1} + (1-\gamma) g_i^2$.
5. Izračunamo ažuriranja $\Delta \boldsymbol{w}_i = -\frac{RMS[\Delta \boldsymbol{w}]_{i-1}}{RMS[g]_i} \cdot g_i$.
6. Akumuliramo ažuriranja $E[\Delta \boldsymbol{w}^2]_i = \gamma \cdot E[\Delta \boldsymbol{w}^2]_{i-1} + (1-\gamma)\Delta \boldsymbol{w}_i^2$.
7. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \Delta \boldsymbol{w}_i$.
8. Povećamo broj napravljenih iteracija $i = i + 1$.

end while

U Adadelta algoritmu ne moramo postaviti početnu stopu učenja jer je eliminirana iz pravila ažuriranja. Pokazao se kao efikasan alat za optimizaciju dubokih modela.

3.7 RMSprop

RMSprop (root mean square propagation) je optimizacijski algoritam koji je također razvijen kako bi se nosio s problemom opadanja stope učenja tijekom treniranja. Razvijen je neovisno od Adadelte, otprilike u isto vrijeme, kako bi riješio problem drastično opadajućih stopa učenja u Adagradu. Zapravo, RMSprop je identičan prvom vektoru ažuriranja Adadelte koji smo prethodno izveli.
Umjesto eksponencijalno opadajuće prosječne promjene, RMSprop koristi eksponencijalno opadajući prosjek kvadrata gradijenata.

$$E[g^2]_i = 0.9E[g^2]_{i-1} + 0.1g_i^2$$
$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \frac{\alpha}{\sqrt{\epsilon + E[g^2]_i}} g_i,$$

gdje je $\gamma = 0.9$, što je i preporuka. Ovaj je algoritam popularan u optimizaciji dubokih modela, posebno u kombinaciji s neuronskim mrežama.

3.8 Adam

Adam (adaptive moment estimation) je još jedna metoda koja računa prilagođene stope učenja za svaki parametar. On kombinira prednosti različitih optimizacijskih tehnika kao

što su momentum, Adadelta i RMSprop. Osim što pohranjuje u memoriju eksponencijalno opadajući prosjek prošlih kvadratnih gradijenata $\boldsymbol{v}_i$, Adam također čuva eksponencijalno opadajući prosjek prošlih gradijenata $\boldsymbol{m}_i$, slično momentumu:

$$\boldsymbol{m}_i = \beta_1 \boldsymbol{m}_{i-1} + (1 - \beta_1) g_i$$

$$\boldsymbol{v}_i = \beta_2 \boldsymbol{v}_{i-1} + (1 - \beta_2) g_i^2,$$

gdje su $\boldsymbol{m}_i$ i $\boldsymbol{v}_i$ procjenitelji prvog momenta (aritmetičke sredine) i drugog momenta (necentrirane varijance) gradijenata, odakle i dolazi naziv metode. Adam također uključuje korekciju pomaka prvog i drugog momenta kako bi se nadoknadila inicijalizacija vektora $\boldsymbol{m}_i$ i $\boldsymbol{v}_i$ kao nul-vektora. Usmjereni su prema nuli tijekom početnih iteracija i kada su stope opadanja male (tj. kada su β_1 i β_2 blizu 1). Suprotstavljamo se ovim pristranostima tako da računamo procjene prvog i drugog momenta na sljedeći način:

$$\hat{\boldsymbol{m}}_i = \frac{\boldsymbol{m}_i}{1 - \beta_1^i}$$

$$\hat{\boldsymbol{v}}_i = \frac{\boldsymbol{v}_i}{1 - \beta_2^i}.$$

Konačno, Adamovo pravilom ažuriranja:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \frac{\alpha}{\sqrt{\hat{\boldsymbol{v}}_i} + \epsilon} \hat{\boldsymbol{m}}_i$$

Adam algoritam je sljedećeg oblika:

Algoritam 10 Adam algoritam

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\alpha > 0$, $\beta_1, \beta_2 \in [0, 1)$, $\epsilon > 0$, $i_{max} \in \mathbb{N}$, $tol > 0$, $i = 0$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$

0. Inicijaliziramo vektor prvog i drugog momenta: $\boldsymbol{m}_0 = 0$, $\boldsymbol{v}_0 = 0$.

while 1 **do**

1. Provjerimo je li $i \leq i_{max}$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
2. Provjerimo je li $\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\| > tol$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
3. Izračunamo gradijent $g_i = [g_{i,k}]_{k\in\{1,2,\ldots,d\}}$, gdje je $g_{i,k} = \nabla f(\boldsymbol{w}_{i,k})$.
4. Ažuriramo pristrani procjenitelj prvog momenta:
   $\boldsymbol{m}_i = \beta_1 \cdot \boldsymbol{m}_{i-1} + (1 - \beta_1) \cdot g_i$.
5. Ažuriramo pristrani procjenitelj drugog momenta:
   $\boldsymbol{v}_i = \beta_2 \cdot \boldsymbol{v}_{i-1} + (1 - \beta_2) \cdot g_i^2$.
6. Ispravljamo pristranost procjenitelja prvog momenta:
   $\hat{\boldsymbol{m}}_i = \boldsymbol{m}_i / (1 - \beta_1^i)$.
7. Ispravljamo pristranost procjenitelja drugog momenta:
   $\hat{\boldsymbol{v}}_i = \boldsymbol{v}_i / (1 - \beta_2^i)$.
8. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \cdot \hat{\boldsymbol{m}}_i \Big/ \left(\sqrt{\hat{\boldsymbol{v}}_i} + \epsilon \right)$.
9. Povećamo broj napravljenih iteracija $i = i + 1$.

end while

Adam se često koristi kao zadani algoritam optimizacije u mnogim bibliotekama koje se koriste za duboko učenje zbog svoje sposobnosti rada u različitim scenarijima i dobre ravnoteže između brzine konvergencije i stabilnosti.

3.9 AdaMax

AdaMax je proširenje optimizacijskog algoritma Adam čiji je cilj riješiti probleme vezane uz nestajanje stope učenja koja se može pojaviti u Adamu. Uvodi beskonačnu normu kako bi zamijenio uobičajenu l_2 normu koja se koristi u ažuriranju težina u Adamu:

$$\boldsymbol{v}_i = \beta_2 \boldsymbol{v}_{i-1} + (1 - \beta_2)|g_i|^2.$$

Možemo generalizirati u l_p normi s parametriziranim β_2:

$$\boldsymbol{v}_i = \beta_2^p \boldsymbol{v}_{i-1} + (1 - \beta_2^p)|g_i|^p.$$

Norme l_p za velike vrijednosti p obično postaju numerički nestabilne, zbog čega su norme l_1 i l_2 najčešće korištene u praksi. Međutim, norma l_∞ je obično stabilna. Kako ne bismo

miješali procjenitelje drugog momenta u AdaMaxu s onima u Adamu, korist ćemo oznaku $\boldsymbol{u}_i$ za $\boldsymbol{v}_i$ ograničenog beskonačnom normom:

$$\boldsymbol{u}_i = \beta_2^{\infty} \cdot \boldsymbol{v}_{i-1} + (1 - \beta_2^{\infty}) \cdot |g_i|^{\infty} = max\{\beta_2 \cdot \boldsymbol{v}_{i-1}, |g_i|\}$$

Konačno, pravilo ažuriranja za AdaMax glasi:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \frac{\alpha}{\boldsymbol{u}_i} \cdot \hat{\boldsymbol{m}}_i$$

Pogledajmo kako izgleda AdaMax algoritam:

Algoritam 11 AdaMax algoritam

Inicijalizacija: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\alpha > 0, \beta_1, \beta_2 \in [0, 1)$, $\epsilon > 0$, $i_{max} \in \mathbb{N}$, $tol > 0$, $i = 0$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$

0. Inicijaliziramo vektor prvog i drugog momenta u normi ∞: $\boldsymbol{m}_0 = 0, \boldsymbol{u}_0 = 0$.

while 1 **do**

1. Provjerimo je li $i \leq i_{max}$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
2. Provjerimo je li $\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\| > tol$. Ako nije, algoritam vraća $\boldsymbol{w}_{i-1}$.
3. Izračunamo gradijent $g_i = [g_{i,k}]_{k \in \{1,2,\ldots,d\}}$, gdje je $g_{i,k} = \nabla f(\boldsymbol{w}_{i,k})$.
4. Ažuriramo pristrani procjenitelj prvog momenta:
   $\boldsymbol{m}_i = \beta_1 \cdot \boldsymbol{m}_{i-1} + (1 - \beta_1) \cdot g_i$.
5. Ažuriramo pristrani procjenitelj drugog momenta u normi ∞:
   $\boldsymbol{u}_i = max\{\beta_2 \cdot \boldsymbol{u}_{i-1}, |g_i| + \epsilon\}$.
6. Ažuriramo parametre: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \cdot \hat{\boldsymbol{m}}_i \Big/ \Big(\sqrt{\hat{\boldsymbol{v}}_i} + \epsilon \Big)$.
7. Povećamo broj napravljenih iteracija $i = i + 1$.

end while

AdaMax ima prednosti koje i Adam, dok rješava neke od njegovih mana, kao što je poboljšana stabilnost kod velikih ažuriranja gradijenata. Ovaj je algoritam pokazao dobre rezultate u treniranju dubokih neuronskih mreža i postao još jedan optimizacijski algoritam koji se koristi u strojnom i dubokom učenju.

Poglavlje 4

Primjena

Primjenu optimizacija gradijentnog spusta ćemo napraviti na skupovima podataka iz medicine i genetike. Prvi primjer je iz medicine i tiče se podataka o pacijentima s rakom pluća. Drugi primjer su podaci iz genetike koje čine dijelovi DNA koje ćemo klasificirati kojem živom biću pripadaju.

4.1 Primjer iz medicine

Rak pluća predstavlja globalnu zdravstvenu zabrinutost i odgovoran je za značajan broj smrtnih slučajeva povezanih s rakom svake godine. Iako je pušenje dugo prepoznato kao vodeći uzrok raka pluća, nova istraživanja sugeriraju da okolišni faktori poput onečišćenja zraka, genetska predispozicija i profesionalni rizici također mogu imati ključnu ulogu u razvoju ove smrtonosne bolesti. Ovaj skup podataka [1] obuhvaća informacije o prisutnosti ovih čimbenika kod pacijenata oboljelih od raka pluća. Podaci posjeduju 24 svojstava koja su kategorijalne varijable. Pogledajmo dostupne informacije o pacijentima u ovom istraživanju.

- Dob
- Spol
- Izloženost onečišćenju zraka
- Konzumacija alkohola
- Alergija na prašinu
- Profesionalni rizici
- Genetska predispozicija
- Kronične bolesti pluća
- Izloženost pasivnom pušenju
- Bol u prsima
- Kašalj s krvlju
- Umor
- Gubitak tjelesne težine
- Otežano disanje
- Zviždanje pri disanju
- Poteškoće pri gutanju
- Promjena oblika noktiju
- Učestalost prehlada

- Uravnotežena prehrana
- Pretilost
- Pušenje
- Suhi kašalj
- Hrkanje
- Level raka pluća

Primjer podataka za neka svojstva:

index	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	Dry Cough	Snoring	Level
0	33	1	2	4	5	4	3	Low
1	17	1	3	1	5	3	7	Medium
2	35	1	4	5	6	8	9	High
3	37	2	7	7	7	4	5	High
4	46	1	6	8	7	3	4	High

Opišimo svojstvo **level raka pluća** pomoću vrijednosti ostalih svojstava.

Za početak moramo definirati model kojim ćemo opisati varijablu level raka pluća. Kako nam je u fokusu gradijentni spust i njegove optimizacije, uzet ćemo jednostavan model te promotriti kako se ponaša u pojedinom slučaju. Opišimo ovisnost level raka pluća o ostalim svojstvima *softmax regresijom* [3].

Softmax regresija, također poznata kao logistička regresija za više klasa, je statistička metoda koja se koristi za klasifikaciju više od dvije kategorije. Ova tehnika je posebno korisna u strojnom učenju za zadatke višeklasne klasifikacije, kao što su prepoznavanje rukom pisanih slova ili klasifikacija slika u više kategorija.

Model softmax regresije koristi softmax funkciju za modeliranje vjerojatnosti pripadnosti svakoj od n klasa. Formulacija za vjerojatnost da ulazni primjer pripada klasi i može se izraziti kao:

$$P(Y = i \mid X) = \frac{e^{\beta_i^T X}}{\sum_{j=1}^{n} e^{\beta_j^T X}} \tag{4.1}$$

Ovdje su:

$P(Y = i \mid X)$ - vjerojatnost da ulazni primjer pripada klasi i,

X - ulazne značajke,

β_i - težine (parametri) za klasu i,

e - eksponencijalna funkcija,

$\sum_{j=1}^{n} e^{\beta_j^T X}$ - suma svih eksponencijalnih funkcija za sve klase.

Cilj treninga u softmax regresiji je naučiti odgovarajuće težine β_i koje minimiziraju funkciju gubitka, često korištenu unakrsnu entropiju:

$$L(Y, \hat{Y}) = -\sum_{i=1}^{n} Y_i \log(\hat{Y}_i) \tag{4.2}$$

Ovdje su:

$L(Y, \hat{Y})$ - funkcija gubitka,

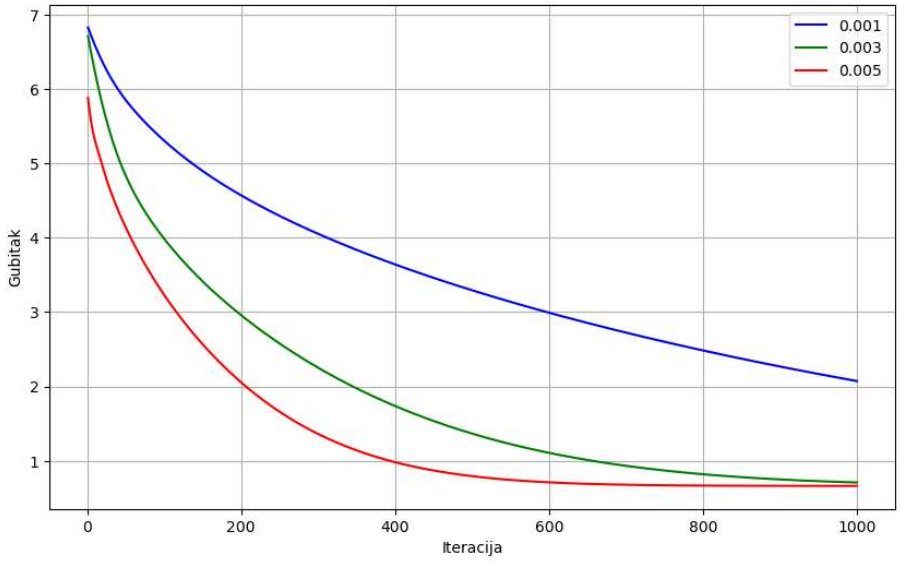
Y - stvarna distribucija klasa za ulazni primjer (one-hot kodiranje),

$\hat{Y}$ - predviđena distribucija klasa (izlaz modela).

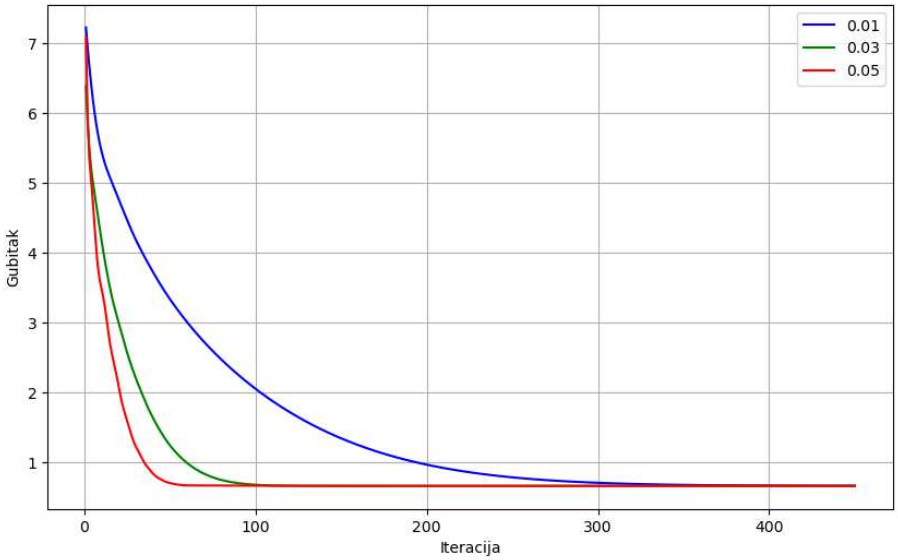
Prijeđimo na rezultate o gradijentnom spustu koje daju ovaj model softmax regresije i naši podaci.

Stopa učenja

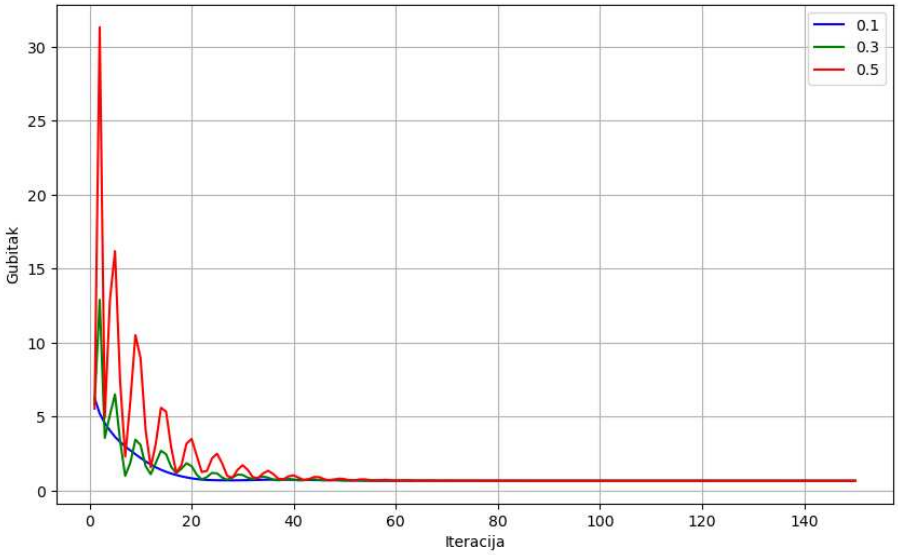
Promotrimo kako različite vrijednosti stope učenja utječu na brzinu konvergencije gradijentnog spusta. Gledamo vrijednosti gubitka po iteracijama za Adam optimizaciju.



Slika 4.1: Usporedba konvergencije za manje stope učenja



Slika 4.2: Usporedba konvergencije za srednje stope učenja



Slika 4.3: Usporedba konvergencije za veće stope učenja

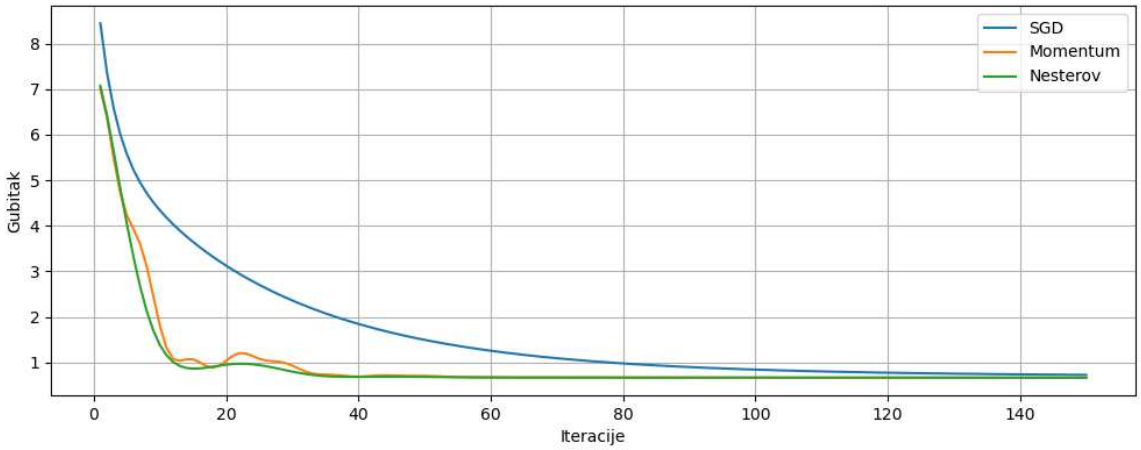
Iz prvog grafa, gdje je $\alpha \in \{0.001, 0.003, 0.005\}$, vidimo da premala stopa učenja uzrokuje sporu konvergenciju te uočavamo da je ona brža za $\alpha = 0.005$ nego za $\alpha = 0.001$. Zbog $0.005 > 0.001$, naslućujemo da će za veće vrijednosti stope učenja konvergencija biti brža.

Našu slutnju potvrđuje zadnji graf. Na njemu je $\alpha \in \{0.1, 0.3, 0.5\}$ te iz njega vidimo da gubitak vrlo brzo padne na malu vrijednost, već za 40-ak iteracija. S druge strane, taj je gubitak dosta nestabilan. Primijetimo kako se oscilacije gubitka smanjuju iteracijama, što je rezultat Adam optimizacije gradijentnog spusta.

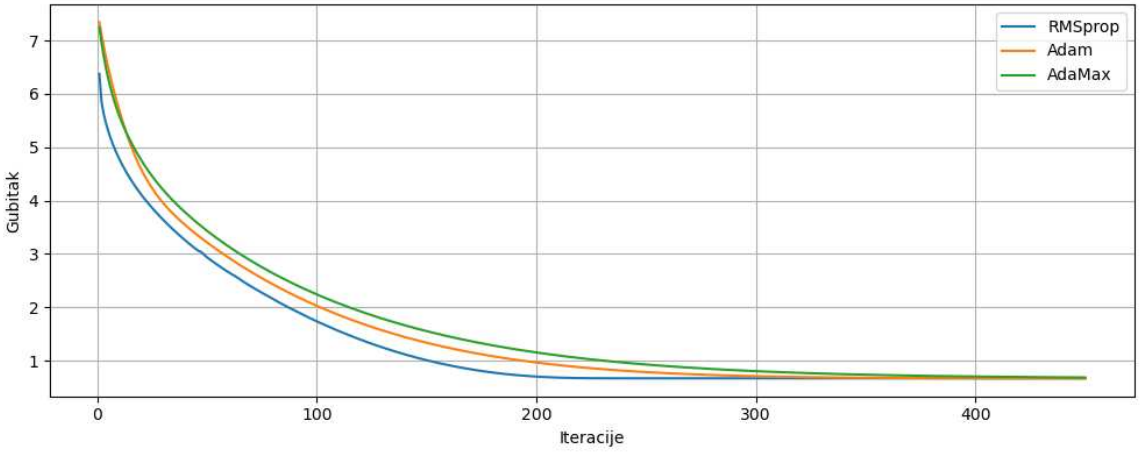
Konačno, srednji graf, sa stopama učenja $\alpha \in \{0.01, 0.03, 0.05\}$, se čini najprikladnijim za rješavanje ovog problema, jer je kompromis između relativno brze konvergencije i stabilnosti gubitka. U nastavku primjera ćemo koristiti $\boldsymbol{\alpha = 0.01}$.

Optimizacije gradijentnog spusta

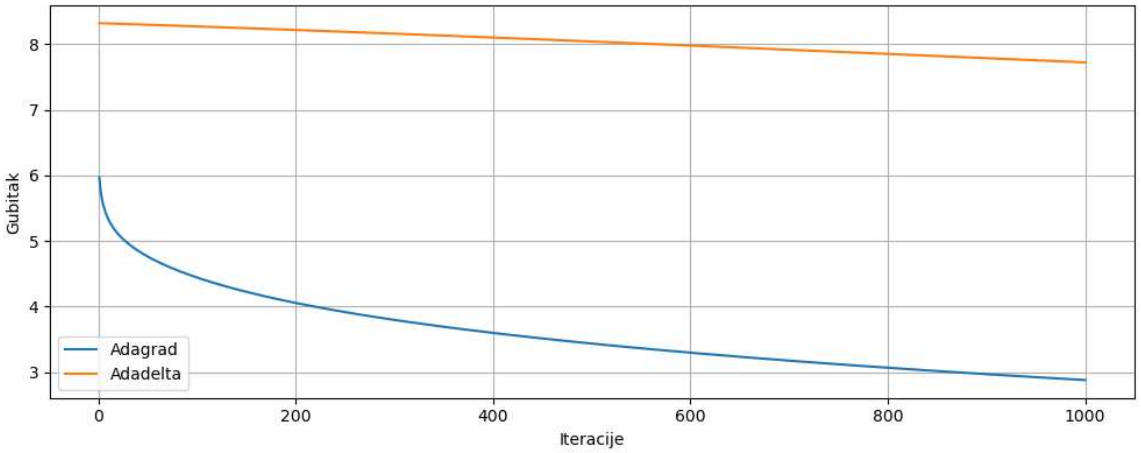
Pogledajmo u nastavku kako optimizacijski algoritmi gradijentnog spusta utječu na konvergenciju funkcije gubitka. Provest ćemo metode koje smo upoznali u Poglavlju 3. Primjenjujemo ih na naš model uz stopu učenja $\alpha = 0.01$. Python kod koji koristimo u ovom procesu je dostupan u Dodatku B.



Slika 4.4: Gubitak SGD-a sa i bez ubrzanih metoda



Slika 4.5: Gubitak uz optimizacijske algoritme RMSprop, Adam i AdaMax



Slika 4.6: Gubitak uz optimizacijske algoritme Adagrad i Adadelta

Usporedbom ova tri grafa uočavamo da su se ubrzane metode (Momentum i Nesterova) pokazale najučinkovitije za ovaj problem. Gubitak brzo padne, već za 15-ak iteracija, te se brzo ustali. Uočavamo da je konvergencija brža nego u slučaju SGD-a bez ubrzanja, što je i očekivano. Ipak, Nesterova metoda ubrzanog gradijenta je nešto stablinija od momentuma.

Nadalje, analizom drugog grafa vidimo da su metode RMSprop, Adam i AdaMax slične u kontekstu konvergencije gradijentnog spusta. One stabilno konvergiraju nakon 400 iteracija, što je ipak značajno lošije od ubrzanih metoda. Naposljetku, Adagrad i Adadelta nisu prikladne za ovaj problem zbog vrlo spore konvergencije u odnosu na ostale metode.

Zaključak

Analizom modela softmax regresije kojom opisujemo level raka pluća, dolazimo do rezultata da je pri računanju gradijentnog spusta dobar odabir stope učenja $\alpha = 0.01$ te primjena metode Nesterovog ubrzanog gradijenta.

Promotrimo drugu primjenu gradijentnog spusta na primjeru iz genetike. On je složeniji od prošlog primjera jer je model neuronska mreža sa 14 slojeva te su podaci većeg obujma. U složenijim je primjerima lakše vidjeti razliku u konvergenciji optimizacijskih metoda.

4.2 Primjer iz genetike

U području genetike, gdje je razumijevanje funkcija gena i njihovih dijelova koji reguliraju druge procese od velike važnosti, modeli strojnog učenja postali su vrlo bitni alati. Oni omogućuju identifikaciju i karakterizaciju ključnih genomskih komponenti. Modeli također doprinose predviđanju ključnih koncepata genomske analize. Npr. dostupnost histona, tj. bjelančevina u jezgri stanice koje služe kako bi se lanac DNA mogao uspješno spiralizirati u kromosome ili vezanje mRNA koja izlazi iz jezgre u citoplazmu na ribosome s uputom za sintezu proteina.

Skup podataka obuhvaća različite dijelove gena kao što su pojačivači, promotori te otvorene regije kromatina. Oni imaju ključnu ulogu u regulaciji kako i kada geni u genomima postaju aktivni. Pojačivači pomažu u pojačavanju ekspresije, promotori započinju transkripciju gena, a stanje kromatina utječe na dostupnost gena za ekspresiju. Svi oni čine osnovu za kontrolu genomske ekspresije i regulaciju bioloških procesa unutar organizama.

Podaci [4] predstavljaju sekvence koje su 200 uzastopnih baza iz DNA. Baze su iz skupa {A, T, C, G} te su izabrane na slučajan način iz genoma čovjeka ili crva. Cilj nam je istrenirati model da primi niz baza i binarno ih klasificira u ovisnosti pripadaju li čovjeku ili crvu. Pogledajmo primjere sekvenci:

index	sample 1	sample 2	sample 3	sample 4	sample 5	sample 6
1	A	T	C	G	T	A
2	C	T	A	G	C	A
3	T	A	G	C	T	A
⋮	⋮	⋮	⋮	⋮	⋮	⋮
199	T	A	G	C	T	A
200	G	C	A	T	C	G

Definirajmo model kao konvolucijsku neuronsku mrežu (CNN) sa 14 slojeva na kojoj ćemo usporediti konvergencije optimizacija gradijentnog spusta.

Konvolucijske neuronske mreže [2] su posebna klasa dubokih neuronskih mreža dizajnirana za obradu podataka u obliku mreže, poput slika ili u našem slučaju sekvenci. Opremljene su slojevima posebno prilagođenim za obradu prostornih hijerarhija prisutnih u ulaznim podacima.

Model koji koristimo [4] sastoji se od sljedećih dijelova:

Pretvorba tekstualnih ulaznih podataka u numeričke Konvolucijske neuronske mreže upravljaju numeričkim podacima, stoga model priprema tekstualne sekvence za prolaz kroz mrežu pomoću vektorizacije i "*one-hot*" kodiranja koji pretvara znakove u vektore s jednim aktivnim bitom koji označava prisutnost znaka.

Prolaz kroz slojeve Pripremljeni numerički podaci prolaze kroz *konvolucijske slojeve* koji primjenjuju filtere na ulazne sekvence kako bi unutar njih otkrili uzorke ili strukture. Postoje i *slojevi sažimanja* koji smanjuju dimenzionalnost izlaza konvolucijskih slojeva zadržavajući bitne značajke. Nadalje, *dropout sloj* smanjuje rizik od prenaučenosti modela tako da na slučajan način odbacuje neurone tijekom treniranja.

ReLU funkcija Aktivacijska funkcija koja se najčešće koristi u našem modelu je ReLU funkcija. Ona je definirana na sljedeći način:

$$f(x) := max\{0, x\} = \begin{cases} x & , x > 0 \\ 0 & , \text{inače.} \end{cases}$$

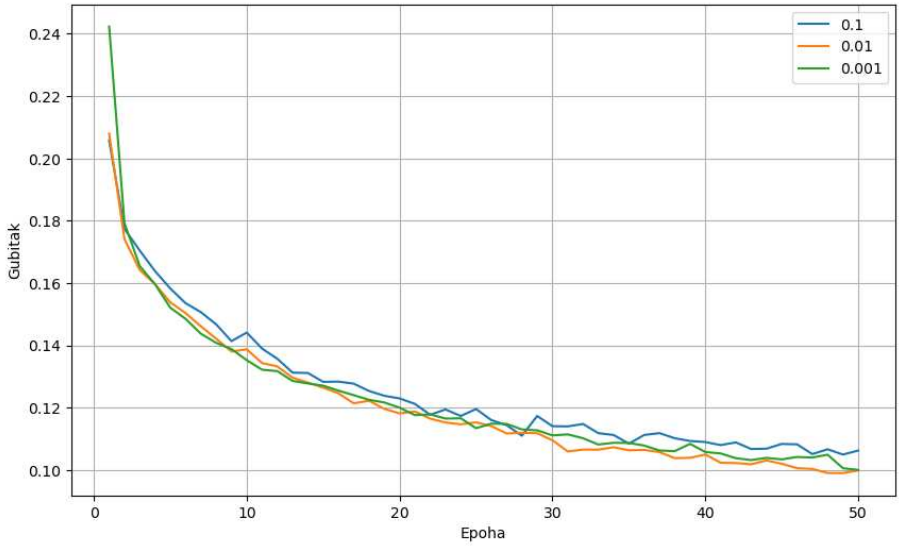
Ona se koristi kako bi se unijeli nelinearni elementi u mrežu. To pomaže u modeliranju kompleksnih značajki.

Završni sloj Posljednji se sloj ponaša kao linearni sloj. On je potpuno povezan s neuronima iz prethodnog sloja te se sastoji od jednog čvora koji se koristi za binarnu klasifikaciju (čovjek ili crv).

Prijeđimo na gradijentni spust. Pri usporedbi stopa učenja i optimizacijskih metoda koristit ćemo epohe umjesto iteracija. Prisjetimo se, epoha predstavlja jedan prolazak kroz cjelokupni skup podataka za treniranje modela.

Stopa učenja

Pogledajmo kako stopa učenja djeluje na ovaj složeni model i puno ulaznih podataka. Primjenjujemo Adam optimizaciju te razne stope učenja.

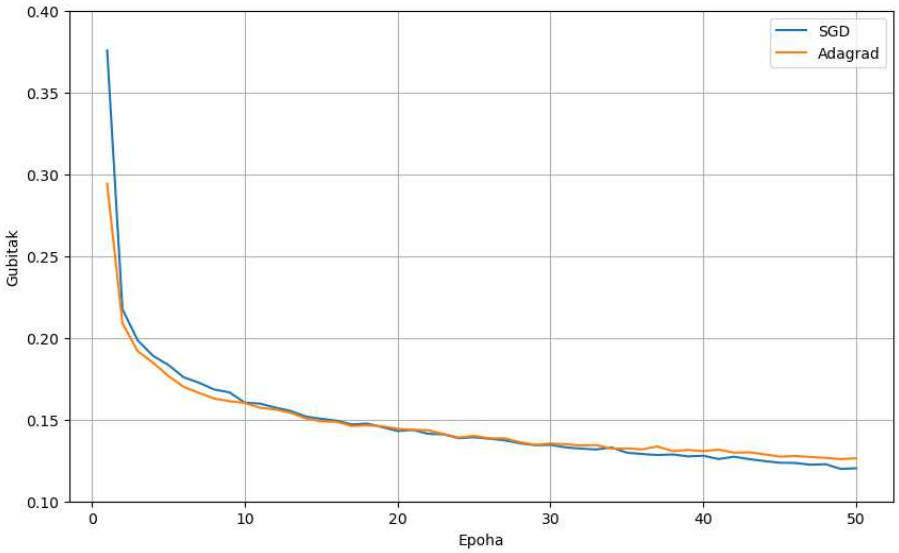


Slika 4.7: Uspredba konvergencije za razne stope učenja

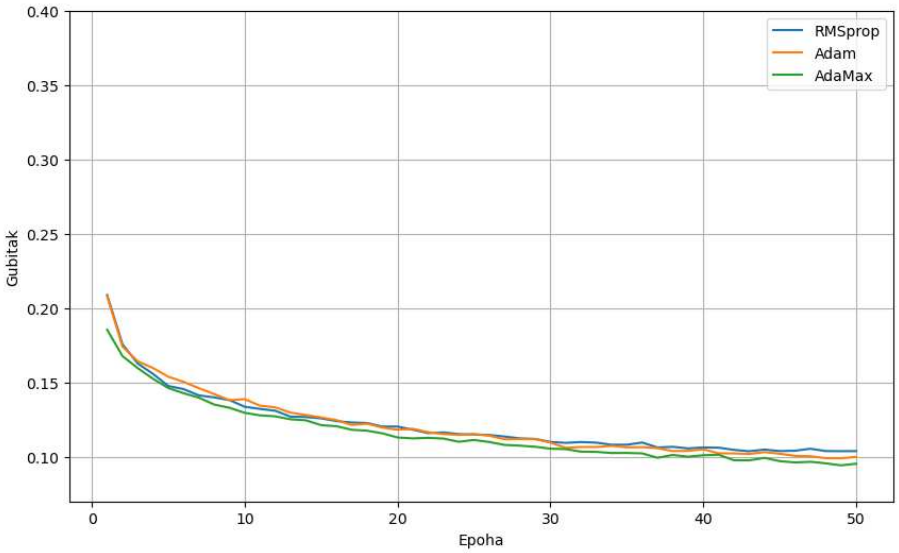
Iz priloženog grafa možemo vidjeti da se gubitak ponaša vrlo slično za sve vrijednosti stope učenja iz skupa $\alpha \in \{0.001, 0.01, 0.1\}$. Ipak, u zadnjih 20-ak iteracija, slučaj $\alpha = 0.01$ postiže najmanji gubitak. Stoga, u nastavku koristimo tu vrijednost stope učenja.

Optimizacije gradijentnog spusta

Nadalje, pogledajmo kako različite optimizacijske metode djeluju na ovaj problem. Na sljedećim je grafovima prikazan gubitak po epohama za razne algoritme. Primjenjujemo metode na naš model uz stopu učenja $\alpha = 0.01$. Kostur koda koji koristimo pri računanju gubitaka za pojedinu optimizacijsku metodu kroz epohe je dostupan u Dodatku B.



Slika 4.8: Uspredba konvergencije SGD-a i Adagrad optimizacijske metode



Slika 4.9: Uspredba konvergencije metoda RMSprop, Adam i AdaMax

Sa gornjih grafova vidimo da su obični SGD i Adagrad dali slične rezultate konvergencije. Isto vrijedi i za drugu skupinu metoda RMSprop, Adam i AdaMax.

SGD i Adagrad pokazuju brzi pad gubitka, već u prvih 10-ak epoha, te se onda nastavlja nešto blaži padajući trend. S druge strane, RMSprop, Adam i AdaMax kreću s manjim gubitkom i u konačnici postižu manju vrijednost od druge dvije metode. Nadalje, pokazuju jednaku sklonost varijabilnosti oko dosegnutog minimuma u kasnijim iteracijama kao i SGD i Adagrad. Stoga, sveukupno gledajući, možemo reći da se AdaMax u ovom slučaju pokazao kao dobar odabir optimizatora gradijentnog spusta.

Zaključak

Analizirajući razne stope učenja i optimizacijske alogritme gradijentnog spusta, došli smo do zaključka da je optimalno uzeti $\alpha = 0.01$ i AdaMax optimizator. Važno je napomenuti da se gubitak ponašao vrlo slično za razne stope učenja i pri analizi najboljeg optimizatora smo došli do više metoda sličnih po efektivnosti.

Poglavlje 5

Zaključak

U ovom radu smo napravili pregled optimizacijskih algoritama za pronalaženje globalnog minimuma funkcije. Počevši od najjednostavnijih, ističući njihove mane, stigli smo do najnovijih algoritama koji se svakodenvno koriste u mnogim primjenama, ponajviše u strojnom učenju. Za svaku od metoda smo objasnili njihove prednosti i nedostatke te priložili nacrt algoritma koji se jednostavno može implementirati u većinu programskih jezika. Uz prezentaciju svih dostupnih metoda, uz dodatne pretpostavke, prezentirali smo dokaz da gradijenti spust uistinu konvergira, te smo odredili brzinu te konvergencije.

U drugom dijelu rada, vodeći se prezentiranim teorijskim idejama, testirali smo dane algoritme na dva statistička modela. U prvom, jednostavnijem primjeru, logističkom regresijom smo modelirali ovisnost stadija raka pluća o brojnim čimbenicima. Zbog jednostavnosti modela, čak i manje napredni algoritmi brzo rješe problem optimizacije, ali je vidljivo da novije metode ipak značajno ubrzavaju konvergenciju ka minimumu funkcije gubitka. Pravo testiranje smo napravili u drugom primjeru, gdje smo konvolucijskim neuronskim mrežama predviđali pripada li uzorak genoma crvu ili čovjeku. Kako se radi o složenom modelu, s velikim brojem parametara koje treba procijeniti, jasno je uočljivo kako naprednije metode imaju bržu konvergenciju od standardnih.

Pronalaženje novih metoda optimizacije i dalje je aktivno područje, i mali napredak može značajno smanjiti vremena izvršavanja učenja velikih statističkih modela (npr. velikih jezičnih modela), koji se znaju trenirati tjednima, ako ne i mjesecima. Kao posljedni napredak u ovom području, treba spomenuti Sophia-u (A Scalable Stochastic Second-order Optimizer for Language Model Pre-training), za koju se tvrdi da je i do dva puta brža od Adama [5]. Kao nastavak na ovaj rad, bilo bi zanimljivo isprobati dani algoritam na našem modelu prepoznavanja gena, te ispitati koliko bržu konvergenciju možemo postići u odnosu na metodu Adam.

Poglavlje 6

Dodaci

6.1 Dodatak A

Python kod za računanje gradijentnog spusta za odlomak 1.3 Primjer:

Gradijentni spust za različite stope učenja

```
import numpy as np

def gradient_descent_alg(start, gradient, learn_rate, max_iter,
tol=0.001):
    steps = [start]
    x = start
    count = 0

    for _ in range(max_iter):
        diff = learn_rate * gradient(x)
        if np.abs(diff) < tol:
            break
        x = x - diff
        steps.append(x)
        count += 1

    return steps, count, x

def f(x):
    return 3*x**2-5*x+8

def gradient_f(x):
    return 6*x-5
```

```
history, num_steps, result = gradient_descent_alg
                                    (10, gradient_f, 0.1, 100)
history, num_steps, result = gradient_descent_alg
                                    (10, gradient_f, 0.3, 100)
```

Python kod za računanje ubrzanog gradijentnog spusta za odlomak 3.4 Primjer(nastavak):

Momentum metoda

```
def gradient_descent_alg(start, gradient, learn_rate, max_iter,
momentum=0, tol=0.001):
    steps = [start]
    x = start
    v = 0
    num_steps = 0

    for _ in range(max_iter):
        diff = learn_rate * gradient(x)
        v = momentum * v + diff
        if np.abs(diff) < tol:
            break
        x = x - v
        steps.append(x)
        num_steps += 1

    return steps, x, num_steps

momentum = 0.9
history_momentum, result_momentum, num_steps_momentum =
        gradient_descent_alg(10, gradient_f, 0.1, 100, momentum)
```

Nesterov ubrzani gradijent

```
def nesterov_gradient_descent_alg(start, gradient, learn_rate, max_iter,
momentum=0, tol=0.001):
    steps = [start]
    x = start
    v = 0
    num_steps = 0

    for _ in range(max_iter):
        x_ahead = x - momentum * v
        diff = learn_rate * gradient(x_ahead)
        v = momentum * v + diff
        if np.abs(diff) < tol:
```

```
            break
        x = x - v
        steps.append(x)
        num_steps += 1

    return steps, x, num_steps

nesterov_momentum = 0.9
history_nesterov, result_nesterov, num_steps_nesterov =
    nesterov_gradient_descent_alg(10, gradient_f, 0.1, 100,
                                                nesterov_momentum)
```

6.2 Dodatak B

Ovdje su dostupni dijelovi Python koda za Poglavlje 4.

Primjer iz medicine

```
import tensorflow as tf
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Ucitavanje podataka
df = pd.read_csv('cancer_patient_data.csv')
df['Level'] = df['Level'].map({'Low': 1, 'Medium': 2, 'High': 3})
data = df.drop(columns=['Level', 'index', 'Patient Id'])
target = df['Level']
data_scaled = StandardScaler().fit_transform(data)

optimizers = ['SGD', 'Momentum', 'Nesterov', 'Adagrad',
                        'Adadelta', 'RMSprop', 'Adam', 'AdaMax']
final_losses = []

for optimizer_name in optimizers:

   # Napravimo model softomax regresiju s 3 klase
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(23,)),
        tf.keras.layers.Dense(3, activation='softmax')
    ])

    optimizer = tf.keras.optimizers.get(optimizer_name.lower(),
                learning_rate=0.01)
    model.compile(optimizer=optimizer, loss='mean_squared_error')
```

```
    target = tf.convert_to_tensor(target, dtype=tf.float32)

    # Petlja treniranja
    num_steps = 1000
    for step in range(num_steps):
        with tf.GradientTape() as tape:
            predictions = model(data_scaled)
            loss = tf.reduce_mean(tf.square(predictions - target))

        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients,
                                      model.trainable_variables))

    final_loss = loss.numpy()
    final_losses.append(final_loss)
```

Primjer iz genetike

```
import tensorflow as tf
from tensorflow.keras.layers import Conv1D, Dense, Dropout,
        GlobalAveragePooling1D, MaxPooling1D
from tensorflow.keras.layers.experimental.preprocessing
    import TextVectorization
import tensorflow_addons as tfa

# Definiramo konvolucijsku neuronsku mrezu
vocab_size = 5
onehot_layer = tf.keras.layers.Lambda
    (lambda x: tf.one_hot(tf.cast(x, "int64"), vocab_size))
last_layer = Dense(1)

model = tf.keras.Sequential([
    onehot_layer,
    Conv1D(32, kernel_size=8, activation="relu"),
    MaxPooling1D(),
    Conv1D(16, kernel_size=8, activation="relu"),
    MaxPooling1D(),
    Conv1D(4, kernel_size=8, activation="relu"),
    MaxPooling1D(),
    Dropout(0.3),
    GlobalAveragePooling1D(),
    last_layer,
])

model.compile(loss="binary_crossentropy", optimizer =
```

```
                tf.keras.optimizers.Adam(learning_rate=0.01),
            metrics=["accuracy", tfa.metrics.F1Score
              (num_classes=1, threshold=0.5, average="micro")])

# Petlja treniranja
EPOCHS = 50
history = model.fit(train_ds, epochs=EPOCHS)
```

Bibliografija

[1] *Lung Cancer Prediction: Air Pollution, Alcohol, Smoking Risk of Lung Cancer*, (2022), https://www.kaggle.com/datasets/thedevastator/cancer-patients-and-air-pollution-a-new-link.

[2] *Convolutional neural network*, https://en.wikipedia.org/wiki/Convolutional_neural_network.

[3] *Softmax Regression*, http://deeplearning.stanford.edu/tutorial/supervised/SoftmaxRegression/.

[4] Katarína Grešová, Vlastimil Martinek, David Čechák, Petr Šimeček i Panagiotis Alexiou, *Genomic benchmarks: a collection of datasets for genomic sequence classifcation*, (2023).

[5] Hong Liu, Zhiyuan Li, David Hall, Percy Liang i Tengyu Ma, *Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training*, 2023.

[6] Clement W. Royer, *Lecture notes on advanced gradient descent*, (2021/2022).

[7] Sebastian Ruder, *An overview of gradient descent optimization algorithms*, (2017).

[8] Matthew D. Zeiler, *Adadelta: An Adaptive Learning Rate Method*, (2017).

Sažetak

U ovom radu stječemo dublje razumijevanje gradijentnog spusta u strojnom i dubokom učenju, istraživajući različite varijante i optimizacije. U prvom poglavlju iznosimo ključne koncepte i teoreme vezane uz konvergenciju gradijentnog spusta i metoda ubrzanja. Nadalje, analiziramo varijante gradijentnog spusta koje se razlikuju po broju točaka iz skupa za treniranje koje se uzimaju u obzir u svakoj iteraciji. Iz njihove jednostavnosti proizlaze brojni izazovi i ograničenja, što nas potiče na istraživanje optimizacijskih algoritama.

Za svaku od optimizacijskih metoda proučavamo teorijske temelje i opis koraka algoritma s ciljem stjecanja intuicije i razumijevanja njihovog funkcioniranja. Nakon teorijskog objašnjenja slijedi primjena varijanti i optimizacijskih algoritama na dva različita modela. Jedan je iz područja medicine, a drugi je iz genetike. Na taj način dobivamo uvid u praktičnu primjenu ovih metoda i njihovu implementaciju u Pythonu. Istražujemo njihovu učinkovitost na konkretnim modelima.

Summary

In this thesis, we aim to better understand gradient descent in machine and deep learning by exploring its different versions and improvements. In the first chapter, we cover essential concepts and theorems related to how gradient descent works and how it can be made faster. We also look at different variations of gradient descent that involve considering different amounts of data in each step. While studying these variations, we encounter various challenges and computer limitations that lead us to optimization techniques.

For each of these optimization methods, we dive into the theory behind them and explain how they work step by step. Our goal is to help readers gain an intuitive understanding of these methods. After explaining the theory, we apply these optimization techniques to two different models. One in the medical field and the other in genetics. This allows us to see how these methods are used in practice and how they can be implemented using Python. We also evaluate how well they perform on real-world models.

Životopis

Zovem se Dorotea Bošnjak. Rođena sam 10. listopada 1999. godine u Osijeku. U osnovnoj i srednjoj školi sam sudjelovala na matematičkim natjecanjima i kampovima nakon čega odlučujem studirati matematiku.

Upisujem inžinjerski smjer na Prirodoslovno-matematičkom fakultetu u Zagrebu 2018. godine. Završetkom preddiplomskog studija dobivam titulu sveučilišnog prvostupnika matematike. Zatim 2021. godine, na istom fakultetu upisujem diplomski studij Matematičke statistike. Očekujem da ću diplomirati u rujnu 2023. godine.

Tijekom studiranja sam stekla radno iskustvo u *Nanobitu* kao analitičarka podataka te sam pohađala Data Science akademiju pod organizacijom *Sofascore*-a. Tijekom preddiplomskog sam studija primala državnu STEM stipendiju. Nadalje, volontirala sam u udruzi *Mladi nadareni matematičari Marin Getaldić* pripremajući učenike za matematička natjecanja kroz predavanja i projekte. Također, sudjelovala sam na ERASMUS+ projektu pod nazivom *"WAR - World Against Racism"* koji se održao u Helsinkiju i Osijeku.

U budućnosti bih se željela razvijati u smjeru analize podataka, strojnog učenja te uhvatiti korak s razvojem umjetne inteligencije.

rujan 2023.