UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET
STUDIJSKI PROGRAM MATEMATIKA

OPTIMIZACIJA FUNKCIJE TROŠKOVA U NEURONSKIM MREŽAMA

MASTER RAD

Mentor:
Miljan Knežević

Student:
Adil Kolaković

Beograd, 2023.

Posvećujem ovaj master rad
mom ocu i majci,
supruzi, bratu i sestri,
koji su uvek bili tu za mene.

Zahvalnica

Veliku zahvalnost, u prvom redu, dugujem svom mentoru profesoru Miljanu Kneževiću, za njegovu nesebičnu podršku, usmeravanje i dragocene savete tokom izrade ovog master rada. Njegovo strpljenje i posvećenost doprineli su razvoju mog istraživačkog duha i usmerili me ka postizanju akademskih ciljeva.

Zahvalan sam i profesorima Zorici Stanimirović i Stefanu Miskoviću na vrednim savetima i sugestijama koje su unapredile kvalitet mog rada.

Takođe, želim da se zahvalim svojim kolegama iz Republičkog zavoda za statistiku koji su me podržavali i motivisali tokom celog procesa pisanja rada. Njihova pomoć, razumevanje i ohrabrenje bili su od neprocenjive vrednosti i doprineli su završetku ovog rada.

Sadržaj

1 Uvod

Neuronska mreža, inspirisana strukturom i funkcijom neurona, predstavlja jedan od pristupa implementaciji veštačke inteligencije. Ovaj sistem se sastoji od određenog broja međusobno povezanih procesora ili čvorova, poznatih kao veštački neuroni, koji simuliraju procese u neuronima. Matematički koreni neuronskih mreža možemo pronaći u linearnoj algebri, matematičkoj analizi i teoriji verovatnoće.

Optimizacija, se odnosi na proučavanje problema u kojima se traži maksimizacija ili minimizacija realne funkcije sistematičkim biranjem vrednosti realnih ili celobrojnih promenljivih iz određenog skupa. Optimizacija predstavlja proces pronalaženja najboljeg rešenja za zadati problem u okviru dozvoljenog skupa. U modernom svetu problemi optimizacije se najčešće rešavaju upotrebom računara, ali sve veća zastupljenost tehnologije u našim životima zahteva i razvoj optimizacionih tehnika za potrebe rešavanja novih problema. Problemi koji se javljaju u inžinjerstvu, ekonomiji, kompjuterskoj nauci sve više zahtevaju uključivanje matematičko programiranje prilikom rešavanja njihovih praktičnih problema.

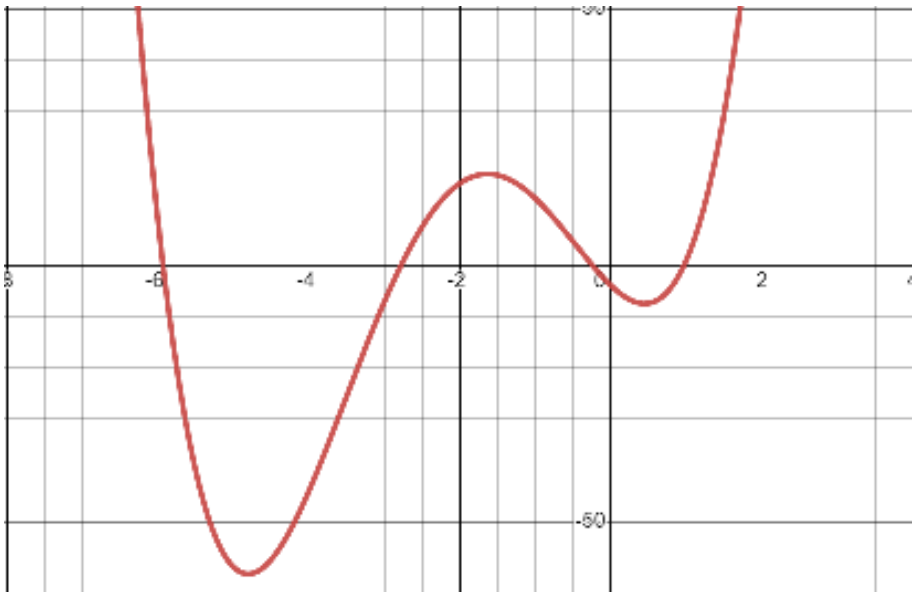
Široko polje koje optimizacija pokriva obuhvata i neuronske mreže, gde raznim algoritmima optimizuje funkciju troškova prilikom treniranja neuronske mreže. Funkcija troška („cost function“) se koristi za merenje performansi modela mašinskog učenja. Model mašinskog učenja bez funkcije troškova je uzaludan. Funkcija troškova nam pomaže da se analizira koliko dobro funkcioniše model mašinskog učenja. Funkcija troškova u osnovi upoređuje predviđene vrednosti sa stvarnim vrednostima. Odgovarajući izbor funkcije troškova doprinosi kredibilitetu i pouzdanosti modela.

Tema master rada se odnosi na pronalazak algoritma koji će dovoljno dobro optimizovati funkciju troškova, a potom ga implementirati u neuronskim mrežama i uporediti brzinu i resurse koje koristi pri radu sa podacima u odnosu na druge najčešće korišćene algoritme optimizacije.

2 Optimizacija

Optimizacija predstavlja granu primenjene matematike koja ima široku primenu u oblastima kao što su fizika, inžinjerstvo, ekonomija, te takođe ima ključnu ulogu u razvoju i treniranju neuronskih mreža.

Cilj optimizacije je minimizacija ili maksimizacija određene funkcije. Na primer, proizvođači nastoje smanjiti troškove dok teže povećanju profita. Stoga, cilj optimizacije je pronalaženje optimalne vrednosti za $x \in \mathbb{X}$, koje označavamo sa $\hat{x}$ (gde je x skup tačaka), koje ispunjavaju zadani kriterijum. Ovaj kriterijum se naziva ciljna funkcija. Uzmimo funkciju $f(x) = x^4 + 8x^3 + 10x^2 - 14x - 4$ kao primer, koju možemo prikazati na koordinatnom sistemu:



Slika 1: Grafik funkcije $f(x) = x^4 + 8x^3 + 10x^2 - 14x - 4$. Kreirano koristeći Desmos [1].

Možemo izračunati gradijent funkcije tako što ćemo odrediti prvi izvod funkcije u odnosu na x i postaviti ga jednakim nuli, a zatim rešiti jednačinu za x. Time dobijamo tačke funkcije koje predstavljaju minimum ili maksimum. U ovom slučaju, imamo:

$$\frac{df}{dx} = 4x^3 + 24x^2 + 20x - 14.$$

Rešavanjem ove jednačine, identifikujemo tri tačke, ali nije poznato koja od njih predstavlja minimum ili maksimum. Kako bismo odredili minimum i maksimum ovih tri tačke, potrebno je izračunati drugi izvod funkcije:

$$\frac{d^2f}{dx^2} = 12x^2 + 48x + 20,$$

nakon čega treba proveriti koja od stacionarnih tačaka ima pozitivnu ili negativnu vrednost drugog izvoda.

Vizuelno, lako možemo odredeti pogledom na grafik funkcije. Ali ukoliko se računski određuje lokalni i globalni minimum, to je zahtevan zadatak. Zato je potrebno početi od jedne tačke i pratiti gradijent do minimuma (nadajući se da ćemo pronaći globalni minimum).

Definicija optimizacije bez ograničenja može se dati kroz problem oblika:

$$\min_{x \in \mathbb{R}^n} f(x),$$

gde je $f : \mathbb{R}^n \to \mathbb{R}$ funkcija cilja definisana na prostoru $\mathbb{R}^n$. Uvek je moguće problem minimizacije prevesti u problem maksimizacije, i obrnuto, tako što se koristi relacija:

$$\max_{x \in \mathbb{R}^n} f(x) = \min_{x \in \mathbb{R}^n} -f(x).$$

U svakodnevnom životu se često susrećemo sa problemima optimizacije. To mogu biti jednostavne situacije kao što je pronalaženje najkraćeg puta do određene destinacije, do složenih problema kao što je optimizacija rasporeda letova u avio kompanijama.[1]

2.1 Globalni i lokalni optimum

U procesu optimizacije, ideja je pronaći najbolje rešenje za određeni problem. Ova "najbolja"rešenja su poznata kao optimalna rešenja, a tačka u kojoj se dostiže optimalna vrednost je poznata kao optimum. Međutim, neke funkcije imaju više od jednog optimalnog rešenja, a ovi optimumi mogu biti klasifikovani kao globalni ili lokalni, u zavisnosti od toga kako su pozicionirani u kontekstu čitave funkcije.

Globalni optimum predstavlja tačku u kojoj funkcija dostiže svoju najvišu (u slučaju maksimizacije) ili najnižu (u slučaju minimizacije) vrednost nad celim definisanim domenom.

S druge strane, lokalni optimum je tačka u kojoj funkcija dostiže najvišu ili najnižu vrednost u odnosu na tačke u neposrednoj blizini. Ova tačka ne mora nužno predstavljati optimalno rešenje za celokupnu funkciju, ali je optimalna u svojoj neposrednoj okolini.

Razumeti razliku između globalnog i lokalnog optimuma je važno pri rešavanju problema optimizacije, jer različite metode optimizacije mogu konvergirati prema različitim tipovima optimuma. Na primer, neki algoritmi mogu brzo pronaći lokalni optimum, ali mogu propustiti globalni optimum, dok drugi algoritmi mogu biti sporiji, ali su sposobni da pronađu globalni optimum.

Kroz razumevanje i upotrebu ovih koncepta, možemo razviti efikasne strategije za rešavanje složenih problema u različitim oblastima - od mašinskog učenja do operacione analize, ekonomije, fizike i mnogih drugih.

[1] Za više detalja, pogledajte [30], [13], [3], [2], [29], [20], [12], [24], [23], [25], [31].

Definicija 2.1. (Optimalno rešenje) U kontekstu optimizacionog problema, rešenje koje uspeva da dostigne najvišu vrednost ciljne funkcije naziva se *optimalno rešenje*, i označava se sa $\hat{x}$.[13]

Definicija 2.2. (Globalni minimum i maksimum) Uzmimo da je $f : S \to \mathbb{R}$ funkcija koja je definisana na skupu $S \subseteq \mathbb{R}^n$.[13] U tom slučaju:

- $\hat{x} \in S$ predstavlja tačku globalnog minimuma funkcije f na skupu S, ukoliko važi da je $f(x) \geq f(\hat{x})$ za sve $x \in S$.
- $\hat{x} \in S$ predstavlja tačku strogog globalnog minimuma funkcije f na skupu S, ukoliko važi da je $f(x) > f(\hat{x})$ za sve $x \in S \setminus \hat{x}$.
- $\hat{x} \in S$ predstavlja tačku globalnog maksimuma funkcije f na skupu S, ukoliko važi da je $f(x) \leq f(\hat{x})$ za sve $x \in S$.
- $\hat{x} \in S$ predstavlja tačku strogog globalnog maksimuma funkcije f na skupu S, ukoliko važi da je $f(x) < f(\hat{x})$ za sve $x \in S \setminus \hat{x}$.

Definicija 2.3. (Lokalni minimum i maksimum) Neka je $f : S \to \mathbb{R}$ funkcija koja je definisana na skupu $S \subseteq \mathbb{R}$[13]. U tom kontekstu:

- $\hat{x} \in S$ označava tačku lokalnog minimuma funkcije f na S ukoliko postoji takvo $r > 0$ da je $f(x) \geq f(\hat{x})$ za sve $x \in S \cap B(\hat{x}, r)$
- $\hat{x} \in S$ označava tačku strogog lokalnog minimuma funkcije f na S ukoliko postoji takvo $r > 0$ da je $f(x) > f(\hat{x})$ za sve $x \in S \cap B(\hat{x}, r) \setminus \hat{x}$
- $\hat{x} \in S$ označava tačku lokalnog maksimuma funkcije f na S ukoliko postoji takvo $r > 0$ da je $f(x) \leq f(\hat{x})$ za sve $x \in S \cap B(\hat{x}, r)$
- $\hat{x} \in S$ označava tačku strogog lokalnog maksimuma funkcije f na S ukoliko postoji takvo $r > 0$ da je $f(x) < f(\hat{x})$ za sve $x \in S \cap B(\hat{x}, r) \setminus \hat{x}$

Globalni i lokalni optimumi su ključni elementi u oblasti optimizacije. Njihovo razumijevanje je ključno za dizajniranje efikasnih algoritama za rešavanje složenih problema. Iako su ovi koncepti matematički u prirodi, njihova primena se proteže daleko van matematike, omogućavajući bolje razumijevanje i optimizaciju različitih sistema u širokom spektru disciplina.

2.2 Uslov optimalnosti prvog reda

Definicija 2.4. (Operator nabla) Nabla operator, poznat i kao del, je vektorski diferencijalni operator u matematici, obično predstavljen simbolom nabla ∇. Koristi se za definisanje niza drugih diferencijalnih operatora u vektorskom računu, uključujući gradient, divergenciju i rotaciju.

Nabla operator može biti definisan u bilo kojem broju dimenzija. U n dimenzija, nabla operator je definisan na sledeći način:

$$\nabla = \frac{\partial}{\partial x_1}\mathbf{i} + \frac{\partial}{\partial x_2}\mathbf{j} + \frac{\partial}{\partial x_3}\mathbf{k} + \cdots + \frac{\partial}{\partial x_n}\mathbf{n}$$

gde su **i**, **j**, **k**, ..., **n** jedinični vektori u pravcima x_1, x_2, x_3, ..., x_n, redom.

Definicija 2.5. (Gradijent funkcije) Gradijent funkcije f sa n promenljivih može se definisati kao vektorsko polje ∇f čija vrednost u tački x predstavlja "pravac i stopu najbržeg rasta".

Drugim rečima, gradient funkcije pokazuje u pravcu najstrmijeg uspona funkcije, a veličina gradienta predstavlja stopu promene funkcije u tom pravcu.

Gradijent funkcije može se izračunati koristeći nabla operator. Gradijent skalarnog polja f u n dimenzija tada se može izračunati kao:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}\right)\mathbf{i} + \left(\frac{\partial f}{\partial x_2}\right)\mathbf{j} + \left(\frac{\partial f}{\partial x_3}\right)\mathbf{k} + \cdots + \left(\frac{\partial f}{\partial x_n}\right)\mathbf{n}$$

gde su $\frac{\partial f}{\partial x_1}$, $\frac{\partial f}{\partial x_2}$, $\frac{\partial f}{\partial x_3}$, ..., $\frac{\partial f}{\partial x_n}$ parcijalni izvodi f po x_1, x_2, x_3, ..., x_n, redom.

Definicija 2.6. (Hesijan matrica) Neka svi parcijalni izvodi drugog reda funkcije $f : \mathbb{R}^n \to \mathbb{R}$ postoje u tački $x \in \mathbb{R}^n$. Matricu reda $n \times n$

$$\nabla^2(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_3}(x) & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_2^2}(x) & \frac{\partial^2 f}{\partial x_2 \partial x_3}(x) & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) \\ \frac{\partial^2 f}{\partial x_3 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_3 \partial x_2}(x) & \frac{\partial^2 f}{\partial x_3^2}(x) & \dots & \frac{\partial^2 f}{\partial x_3 \partial x_n}(x) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \frac{\partial^2 f}{\partial x_n \partial x_2}(x) & \frac{\partial^2 f}{\partial x_n \partial x_3}(x) & \dots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{bmatrix}$$

nazivamo Hesseova matrica ili Hessijan funkcije f u tački x.

Teorema 2.1. (Uslovi optimalnosti prvog reda) Razmotrimo funkciju $f : U \to \mathbb{R}$ koja je definisana na $U \subseteq \mathbb{R}^n$. Neka je $\hat{x} \in int(U)$ tačka lokalnog optimuma i da svi parcijalni izvodi funkcije f postoje u $\hat{x}$. Tada važi $\nabla f(\hat{x}) = 0$

Definicija 2.7. (Stacionarne tačke) Ukoliko je $f : U \to \mathbb{R}$ funkcija definisana na $U \subseteq \mathbb{R}^n$, pretpostavimo da $\hat{x} \in int(U)$ i da je funkcija f diferencijabilna u nekoj okolini tačke $\hat{x}$. Tada je $\hat{x}$ stacionarna tačka funkcije f ako je $\nabla f(\hat{x}) = 0$[13]

2.3 Uslov optimalnosti drugog reda

Teorema 2.2. (Potrebni uslovi optimalnosti drugog reda) Uzmimo $f : U \to \mathbb{R}$ funkciju definisanu na otvorenom skupu $U \subseteq \mathbb{R}$. Pretpostavimo da je funkcija f dva puta neprekidno diferencijabilna na skupu U i da je $\hat{x}$ stacionarna tačka funkcije f.[13] Tada sledi:

- Ako je $\hat{x}$ tačka lokalnog minimuma funkcije f na U, tada $\nabla^2 f(\hat{x}) \succeq 0$
- Ako je $\hat{x}$ tačka lokalnog maksimuma funkcije f na U, tada $\nabla^2 f(\hat{x}) \preceq 0$

Teorema 2.3. (Dovoljan uslov optimalnosti drugog reda) Za funkciju $f : U \to \mathbb{R}$ definisanu na $U \subseteq \mathbb{R}^n$, pretpostavimo da je funkcija f dva puta neprekidno diferencijabilna na skupu U i neka je $\hat{x} \in int(U)$ stacionarna tačka.[13] Tada sledi:

- Ako je $\nabla^2 f(\hat{x}) \succ 0$ tada je $\hat{x}$ tačka strogog lokalnog minimuma funkcije f na U.
- Ako je $\nabla^2 f(\hat{x}) \prec 0$ tada je $\hat{x}$ tačka strogog lokalnog maksimuma funkcije f na U.

Definicija 2.8. (Sedlasta tačka) Za funkciju $f : U \to \mathbb{R}$ definisanu na otvorenom skupu $U \subseteq \mathbb{R}^n$, pretpostavimo da je funkcija f neprekidno diferencijabilna na skupu U. Stacionarna tačka $\hat{x}$ se naziva sedlasta tačka funkcije f na skupu U ukoliko nije tačka ni lokalnog minimuma ni lokalnog maksimuma funkcije f na skupu U.[13]

2.4 Globalni uslovi optimalnosti

Uslovi koji obezbeđuju globalnu konvergenciju moraju da koriste globalne informacije. Kao na primer da je Hesijan funkcije uvek pozitivno semi-definitan, da su sve stacionarne tačke takođe globalne tačke minimuma.

Teorema 2.4. Razmotrimo funkciju f koja je dva puta neprekidno diferencijabilna i definisana na $\mathbb{R}^n$. Neka je $\nabla^2 f(x) \succeq 0$ za svako $x \in \mathbb{R}^n$. Ukoliko je $\hat{x}$ stacionarna tačka funkcije f , tada je tačka $\hat{x}$ tačka globalnog minimuma funkcije f.[13]

2.5 Kvadratne funkcije

Kvadratne funkcije su bitna klasa funkcija koje se koriste u modeliranju mnogih optimizacionih problema.

Definicija 2.9. Kvadratna funkcija na $\mathbb{R}^n$ predstavlja funkciju u formi

$$f(x) = x^T Ax + 2b^T x + c,$$

gde je $A \in \mathbb{R}^{n \times m}$ matrica, $b \in \mathbb{R}^n$ i $c \in \mathbb{R}$. Gradijent i Hesijan kvadratne funkcije imaju jednostavne analitičke formule:

$$\nabla f(x) = 2Ax + 2b,$$

$$\nabla^2 f(x) = 2A$$

Lema 2.1. Za $f(x) = x^T Ax + 2b^T x + c$, gde je $A \in \mathbb{R}^{n\times n}$ simetrična matrica, $b \in \mathbb{R}$ i $c \in \mathbb{R}$, sledi:

- x je stacionarna tačka funkcije f ako i samo ako $Ax = -b$,
- Ako je $A \succeq 0$, tada je x tačka globalnog minimuma funkcije f ako i samo ako je $Ax = -b$,
- Ako je $A \succ 0$, tada je $x = -A^{-1}b$ tačka strogog globalnog minimuma funkcije f.[13]

Dokaz 2.1. Pretpostavimo da je $f(x) = x^T Ax + 2b^T x + c$, gde je $A \in \mathbb{R}^{n\times n}$ simetrična matrica, $b \in \mathbb{R}^n$ i $c \in \mathbb{R}$.

(1) Gradijent f je vektor čije su komponente parcijalni izvodi funkcije f u odnosu na svaku od komponenti x. Prva tačka se može pokazati izračunavanjem gradijenta funkcije f i postavljanjem ga na nulu. Izračunavanje gradijenta daje:

$$\nabla f(x) = 2Ax + 2b.$$

Postavljanje ovog izraza na nulu, dobijamo $Ax = -b$.

(2) Ako je $A \succ 0$, tada je matrica A pozitivno definitna. Za pozitivno definitne matrice, kvadratna forma $x^T Ax$ je uvek pozitivna za sve $x \neq 0$. Dakle, $f(x)$ će dostići svoj minimum kada $x^T Ax$ bude najmanje, tj. kada je $x^T Ax = 0$. Ovo se događa kada $Ax = -b$, što dokazuje drugi deo leme.

(3) Ako je $A \succ 0$, onda postoji inverzna matrica A^{-1}. Tada možemo izraziti x kao $x = -A^{-1}b$. Umetanjem ove vrednosti u izraz za $f(x)$ i s obzirom da je A pozitivno definitna, dobijamo da je $f(x)$ minimalno kada je $x = -A^{-1}b$, što dokazuje treći deo leme.

3 Gradijentne metode

Gradijentne metode su kamen temeljac u mnogim oblastima nauke i tehnike, kao što su mašinsko učenje, veštačka inteligencija, fizika, hemija, matematika i mnogi drugi. Ove metode su u suštini algoritmi za rešavanje problema optimizacije, pri čemu se koriste gradijenti, odnosno vektori izvoda funkcija, kako bi se postiglo poboljšanje u odnosu na prethodne vrednosti.

Gradijent funkcije u određenoj tački predstavlja vektor koji pokazuje pravac najbržeg rasta funkcije u toj tački. Dakle, u kontekstu optimizacije, ideja je da sledimo suprotan pravac gradijenta kako bi se smanjila vrednost funkcije. Ovaj princip je u osnovi algoritama gradijentnog spusta, koji su veoma popularni u mašinskom učenju.

Gradijentne metode se klasifikuju na metode prvog i drugog reda, u zavisnosti od toga koliko informacija o funkciji koriste. Metode prvog reda koriste samo prvi izvod (gradijent) funkcije, dok metode drugog reda koriste i drugi izvod (Hesenovu matricu). Metode drugog reda obično konvergiraju brže, ali su računski zahtevnije, dok metode prvog reda mogu biti efikasnije za velike probleme, iako mogu zahtevati više iteracija da bi konvergirale.[29][20]

Treba naglasiti da su gradijentne metode samo jedan deo šire familije algoritama za optimizaciju. Postoje i negradijentne metode koje koriste samo vrednosti funkcije, bez informacija o gradijentima ili izvodima. Ove metode su obično manje efikasne, ali mogu biti korisne kada izvodi funkcije nisu dostupni ili su teško izračunljivi.[2]

3.1 Metod najbržeg spusta

Metod najbržeg spusta je fundamentalni gradijentni pristup, koji se uveliko primenjuje u rešavanju neograničenih optimizacionih problema. Metoda koristi informacije dobijene iz gradijenta funkcije da bi otkrila putanju najbržeg spusta, odnosno najefikasnijeg smanjenja vrednosti funkcije.

Osnovna zamisao metoda najbržeg spusta je određivanje koraka u smeru suprotnom od gradijenta funkcije u trenutnoj tački, za svaku iteraciju algoritma. Taj vektor, označen sa $d_k = -\nabla f(x_k)$, označava pravac i smer opadanja, jer gradijent funkcije je vektor koji pokazuje na putanju najbržeg rasta funkcije. Dakle, kretanje u smeru suprotnom od gradijenta vodi ka najbržem padu funkcije. Da bi dokazali da je ovo zaista pravac i smer opadanja, kad god je $\nabla f(x_k) \neq 0$, posmatrajmo izvod funkcije $f(x_k - \nabla f(x_k))$ duž pravca $-\nabla f(x_k)$:

$$f'(x_k - \nabla f(x_k)) = -\nabla f(x_k)^T \nabla f(x_k) = -\|\nabla f(x_k)\|^2 < 0$$

Uz to, kod opadajućih pravaca negativni gradijent takođe predstavlja metod

[2]Za više detalja, pogledajte [30], [13], [3], [2], [29], [20], [12], [24], [23], [25], [31].

najbržeg spusta, što znači da normalizovani nagib $\nabla f(x_k)/\|f(x_k)\|$ predstavlja minimalni izvod u smeru među svim normalizovanim pravcima. Sledeća lema pruža formalni dokaz.

Lema 3.1. Ukoliko je f neprekidno diferencijabilna funkcija i $x \in \mathbb{R}^n$ predstavlja nestacionarnu tačku ($\nabla f(x) \neq 0$), optimalno rešenje problema:

$$\min_{d \in \mathbb{R}^n} \{f'(x; d) : \|d\| = 1\}$$

je $d = -\frac{\nabla(f(x))}{\|\nabla f(x)\|}$.

Dokaz 3.1. Kako je $f'(x; d) = \nabla f(x)^T d$, problem (3.1) ekvivalentan je:

$$\min_{d \in \mathbb{R}^n} \{\nabla f(x)^T d : \|d\| = 1\}.$$

Primenom Koši-Švarc nejednakosti i koristeći $\|d\| = 1$, imamo:

$$\nabla f(x)^T d \geq -\|\nabla f(x)\|\|d\| = -\|\nabla f(x)\|.$$

Prema tome, $-\|\nabla f(x)\|$ je donja granica optimalne vrednosti problema (3.1). S druge strane, ako uzmemo da je $d = -\frac{\nabla f(x)}{\|\nabla f(x)\|}$ i zamenimo u funkciji cilja (3.1) dobijemo

$$f'\left(x, -\frac{\nabla f(x)}{\|\nabla f(x)\|}\right) = -\nabla f(x)^T \frac{\nabla f(x)}{\|\nabla f(x)\|} = -\|\nabla f(x)\|,$$

i tako dolazimo do zaključka da je donja granica $-\|\nabla f(x)\|$ postignuta u $d = -\frac{\nabla f(x)}{\nabla f(x)}$, što implicira da je to optimalno rešenje našeg problema (3.1). □ [13]

Ne samo da je ovo putanja opadanja, negativni gradijent je i pravac najbržeg spusta. To znači, ukoliko analiziramo sve moguće pravce i odaberemo onaj za koji je izvod funkcije po pravcu minimalan, pravac negativnog gradijenta bi bila ta.

Metod najbržeg spusta je posebno popularan ne samo zbog svoje efikasnosti, već i zbog svoje jednostavnosti i univerzalne primenljivosti. Ove osobine ga čine osnovnim za veliki broj ostalih metoda optimizacije.

Metod najbržeg spusta je gradijentna metoda kod koje se veličina koraka t_k bira tako da se u svakoj iteraciji metode postiže maksimalno opadanje vrednosti funkcije cilja $f(x)$. Preciznije, za t_k se uzima rešenje problema minimizacije funkcije $g(t)$ za $t \geq 0$, gde je $g(t) : R_0^+ \to R$ definisana sa

$$g(t) = f(x_k - t\nabla f(x_k))$$

[30]

Algoritam 1 Metod najbržeg spusta

1: Zadajte granicu tolerancije ϵ.
2: Ustanovite početnu tačku $x_0 \in \mathbb{R}^n$ postavljajući $k = 0$.
3: Koristeći metod linijskog pretraživanja, odredite vrednost koraka t_k za funkciju

$$g(t) = f(x_k - t\nabla f(x_k)),$$

gde je $t_k \geq 0$.
4: Izračunajte novu tačku x_{k+1} pomoću formule $x_{k+1} = x_k - t_k \nabla f(x_k)$.
5: Proverite uslov zaustavljanja: ako je $\|\nabla f(x_{k+1})\| < \epsilon$, algoritam se zaustavlja, a rezultat je x_{k+1}. Ako uslov nije ispunjen, postavite $k = k + 1$ i vratite se na korak (2).

Definicija 3.1. (Linijsko pretraživanje) Linijsko pretraživanje je postupak koji se koristi u iterativnim metodama optimizacije kako bi se pronašla optimalna vrednost koraka (obično označena sa t ili α) koja će minimizovati funkciju cilja u nekom pravcu. Postupak linijskog pretraživanja ima za cilj da se pronađe korak koji će obezbediti đobar"napredak prema minimumu funkcije, ali bez potrebe za prevelikim koracima koji mogu da preskoče minimum.

U kontekstu metoda najbržeg spusta, linijsko pretraživanje koristi se za određivanje koraka t_k tako da se minimizira funkcija f u pravcu negativnog gradijenta $-\nabla f(x_k)$.

Formalno, za dato x_k i pravac d_k, linijsko pretraživanje rešava sledeći problem minimizacije:

$$\min_{t \geq 0} f(x_k + td_k).$$

Postoji mnogo metoda linijskog pretraživanja, a neki od najpopularnijih su:

1. Egzaktno linijsko pretraživanje: Gde se problem minimizacije direktno rešava do tačnog minimuma.

2. Backtracking: Postupak gde se počinje sa nekom maksimalnom vrednošću za t i smanjuje se dok se ne zadovolji neki uslov (npr. Armijo uslov).

3. Zlatni presek: Metod koji koristi svojstva zlatnog preseka za efikasno pretraživanje intervala gde se nalazi minimum.

4. Interpolacija: Koristi se interpolacija (najčešće kvadratna ili kubna) da bi se odredila nova tačka pretraživanja.[24]

Lema 3.2. Pretpostavimo da je $\{x_k\}_{k \geq 0}$ niz generisan metodom najbržeg spusta sa tačnim linijskim pretraživanjem za rešavanje problema minimizacije neprekidno diferencijabilne funkcije f. Tada se može zaključiti da za svako $k = 0, 1, 2, 3 \ldots$ sledi:

$$(x_{k+2} - x_{k+1})^T (x_{k+1} - x_k) = 0.$$

Dokaz 3.2. Neka je $\{x_k\}_{k\geq 0}$ niz generisan metodom najbržeg spusta sa tačnim linijskim pretraživanjem. Za svako $k = 0, 1, 2, 3, \ldots$, trebalo bi da pokažemo da važi

$$(x_{k+2} - x_{k+1})^T (x_{k+1} - x_k) = 0.$$

Podsetimo se da se metod najbržeg spusta definiše prema pravcu negativnog gradijenta, što znači da imamo $x_{k+1} = x_k - t_k \nabla f(x_k)$, gdje je t_k korak u k-toj iteraciji dobijen tačnim linijskim pretraživanjem.

Prema svojstvu metode najbržeg spusta, tačno linijsko pretraživanje garantuje da će se sledeći korak nalaziti duž pravca koji minimizira funkciju.Pretpostavimo da uzmemo skalarni proizvod dvaju vektora $(x_{k+2} - x_{k+1})$ i $(x_{k+1} - x_k)$. Zbog načina na koji su ovi vektori definisani, mi u suštini uzimamo skalarni proizvod gradijenata funkcije f u različitim tačkama.

Kao rezultat toga, pravac između x_{k+1} i x_{k+2} biti ortogonalan na pravac između x_k i x_{k+1}, jer to minimizira projekciju jednog gradijenta na drugi. Zato, skalarni proizvod ovih vektora je 0.

3.2 Uslovi konvergencije gradijentne metode

Razmatrajući konvergenciju gradijentne metode, usredsređujemo se na minimizaciju kvadratne funkcije date oblikom:

$$f(X) = \frac{1}{2}X^T QX - b^T X, \quad X \in \mathbb{R}^n,$$

gde je Q kvadratna, realna, simetrična, pozitivno definitna matrica reda n i $b \in \mathbb{R}^n$ je zadati vektor.

Uslovi konvergencije gradijentne metode su ključni u nastojanju da će metoda težiti ka optimalnom rešenju. Bez tih kriterijuma, ne postoji sigurnost da će iterativna metoda poput gradijentne metode ikada dostići optimalno rešenje.

Da bismo olakšali razumijevanje konvergencije kvadratne funkcije f, uvodimo dodatnu funkciju $G : \mathbb{R}^n \to \mathbb{R}$ kako slijedi:

$$G(X) = f(X) + \frac{1}{2}(X^*)^T QX^* = \frac{1}{2}(X - X^*)^T Q(X - X^*),$$

gdje je X rešenje problema $QX = b$. Budući da je $G(X) - f(X) = \frac{1}{2}(X)^T QX$, a X je konstanta, minimizacija funkcije G ekvivalentna je minimizaciji funkcije f.

Za procenu brzine konvergencije koristimo razliku $|f(X_k) - f(X^*)|$. Pošto je $X = Q^{-1}b$, odnosno $QX^* = b$, imamo:

$$\begin{aligned}
f(X_k) - f(X^*) &= \left(\frac{1}{2}(X_k)^T QX_k - b^T X_k\right) - \left(\frac{1}{2}(X^*)^T QX^* - b^T X^*\right) \\
&= \frac{1}{2}(X_k)^T QX_k - (QX^*)^T X_k - \left(\frac{1}{2}(X^*)^T QX^* - (QX^*)^T X^*\right) \\
&= \frac{1}{2}(X_k)^T QX_k - (X^*)^T QX_k - \left(\frac{1}{2}(X^*)^T QX^* - (X^*)^T QX^*\right) \text{[30]} \\
&= \frac{1}{2}(X_k)^T QX_k - (X^*)^T QX_k + \frac{1}{2}(X^*)^T QX^* \\
&= \frac{1}{2}(X_k - X^*)^T Q(X_k - X^*) \\
&= G(X_k)
\end{aligned}$$

Konvergenciju ćemo dokazati koristeći pomoćnu funkciju $G(X)$ i naredne leme.

Lema 3.3. Neka je $\{X_k\}$ iterativni niz konstruisan gradientnom metodom pri minimizaciji funkcije $f(X) = \frac{1}{2}X^T QX - b^T X$, gde je Q simetrična, pozitivno definitna matrica reda n i $b \in \mathbb{R}^n$ dati vektor. Kako je niz $\{X_k\}$ definisan sa

$$X_{k+1} + X_k - \alpha_k f_k,$$

gde je $f_k = \nabla f(X_k) = QX_k - b$, tada važi sledeća veza:

$$G(X_{(k+1)}) = (1 - \gamma_k)G(X_{(k)}).$$

Pri tome, važe sledeća pravila:

- ukoliko je $f_k = 0$, tada je $\gamma_k = 1$;
- ukoliko je $f_k \neq 0$, tada je $\gamma_k = \alpha_k \frac{f_k^T Q f_k}{f_k^T Q^{-1} f_k} \left(2\frac{f_k^T f_k}{f_k^T Q^{-1} f_k} - \alpha_k \right)$;
- ukoliko je niz $\{X_k\}$ dobijen primenom metode najbržeg spusta, tada je $\gamma_k = \frac{(f_k^T f_k)}{(f_k^T Q^{-1} f_k)(f_k^T Q f_k)}$.[30]

Lema 3.4. Neka je niz $\{X_k\}$ dobijen primenom metode najbržeg spusta na minimizaciju kvadratne funkcije $f(X) = \frac{1}{2}X^T QX - b^T X$. Ako je $f_k \neq 0$ za svako k, tada je $\gamma_k = 1$ ako i samo ako je f_k sopstveni vektor matrice Q.[30]

Lema 3.5. Neka je niz $\{X_k\}$ dobijen primenom metode najbržeg spusta na minimizaciju kvadratne funkcije $f(X) = \frac{1}{2}X^T QX - b^T X$. Tada je $\gamma_k = 1$ ako i samo ako je $f_k = 0$ ili je f_k sopstveni vektor matrice Q.[30]

Teorema 3.1. Neka je $\{X_k\}$ iterativni niz konstruisan gradijentom metodom pri minimizaciji funkcije $f(X) = \frac{1}{2}X^T QX - b^T X$, gde je Q simetrična,pozitvno definitna matrica reda n i $b \in \mathbb{R}^n$ dati vektor. Niz $\{X_k\}$ je definisan sa

$$X_{k+1} = X_k - \alpha_k f_k,$$

gde je $f_k = \nabla f(X_k) = QX_k - b$. Neka je γ_k definisano kao u lemi 3.3 i pretpostavimo da je $\gamma_k > 0$ za svako k. Pod navedenim pretpostavkama, $X_k \to X^*, \quad k \to \infty$ za svaki početni vektor X_0 ako i samo ako važi :

$$\sum_{k=0}^{\infty} \gamma_k = \infty [30]$$

4 Neuronske mreže

Duboko učenje se može razumeti kao podoblast mašinskog učenja, koje se tiče efikasnog obučavanja veštačkih neuronskih mreža sa više slojeva. Osnovni koncept veštačkih neuronskih mreža izgrađen je po hipotezama i modelima funkcionisanja ljudskog mozga za rešavanje kompleksnih problema. Istorija neuronskih mreža ne počinje od skora, već se ona veže za radove Varena Mekkaloka i Valtera Pitsa "A Logical Calculus of the Ideas Immanent in Nervous Activity"(1943), gde su opisali kako neuroni kao "osnovna jedinica mozga"mogu da funkcionišu. U godinama koje su sledile nastupila je takozvana "AI zima", tj. vreme kada implementacija neuronskih mreža nije bilo uspešno zbog toga što niko nije imao dovoljno dobro rešenje za obučavanje neuronske mreže sa više slojeva. Otkriće algoritma za propagaciju unazad "Backpropagation algorithm", prekida "AI zimu"i dovodi do razvoja neuronskih mreža.

Neuronska mreža je koncept međusobno povezanih modela koji su povezani putem veštačke inteligencije. Ovi modeli, iako sintetički, imaju isti nivo interakcija koji je primetan između ljudi. Zbog dugog perioda obuke i učenja modela, nivo njihove međusobne povezanosti će zavisiti od automatizovanog baznog sistema. [4]

Neuronske mreže su danas veoma popularne, zahvaljujući otkrićima u prethodnoj deceniji, što je dovelo do onoga što danas nazivamo algoritmi i arhitekture dubokog učenja - neuronske mreže koje su sastavljene od mnogo slojeva. One su popularne ne samo u akademskim krugovima već i u velikim tehnološkim kompanijama, kao što su "Facebook", "Google", "Microsoft", koje mnogo ulažu u istraživanje veštačkih neuronskim mreža i dubokog učenja.

U današnje vreme kompleksne neuronske mreže koje se pokreću algoritmima dubokog učenja smatraju se vrhunskim rešenjima za složene probleme, kao što su prepoznavanje slika i glasa. Popularni primeri proizvoda u svakodnevnom životu koje pokreće duboko učenje su "Googleova"pretraga slika i "Google Translate- aplikacija koja automatski prepoznaje tekst u slikama za prevod u realnom vremenu na više od dvadeset jezika. Pored ovih primena najznačajnije primene su kategorizaciji teksta, medicinskoj dijagnostici,autonomnoj vožnji. Neuronske mreže zapravo predstavljaju parametrizovanu reprezentaciju koja može da posluži za aproksimaciju drugih funkcija. Kao i u slučaju drugih metoda mašinskog učenja,pronalaženje odgovarajućih parametara se vrši matematičkom optimizacijom nekog kriterijuma kvaliteta aproksimacije i može biti računski vrlo zahtevno.

Neuronske mreže mogu se klasifikovati u različite vrste, u zavisnosti od primene za koju su namenjene. Osnovna vrsta neuronske mreže predstavlja *potpuno povezana mreža (fully connected)*. Za obradu slika, različitih vrsta signala, kao i teksta, koriste se *konvolutivne neuronske mreže (convolutional neural ne-*

tworks). Rekurentne neuronske mreže (recurrent neural networks) se primenjuju za obradu sekvencijalnih podataka promenljive dužine, dok se *rekurzivne neuronske mreže (recursive neural networks)* koriste za obradu podataka koji se mogu predstaviti u obliku stabala. Za obradu podataka koji se mogu predstaviti preko grafova, koriste se *grafovske neuronske mreže (graph neural networks).*

Treniranje neuronske mreže može se predstaviti kao problem optimizacije, u kojem se parametri mreže biraju tako da minimizuju zadatu funkciju troškova, na primer, kvadratnu funkciju troškova na osnovu greške u predviđanju izlaza. Performanse algoritama za treniranje obično se mere kroz pojam rizika, koji predstavlja očekivanu funkciju troškova na neviđenim test podacima. [14]

Bez obzira na velike probleme koji su rešeni primenom neuronskih mreža, ne treba olako posezati za korišćenjem neuronskih mreža pri rešavanju nekog problema. Pre početka pravljenja modela, potrebno je dobro razmotriti kakav je skup podataka kojim mi raspolažemo. Skupovi podataka za koje su neuronske mreže najbolji izbor su oni koji sadrže veliku količinu sirovih podataka, na skupu podataka sa malom količinom podataka moguće je da se neuronske mreže lako preprilagode ili da u slučaju da u tom "malom" skupu podataka naiđemo na disjuknte podatke koji će u tom malom broju loše obučiti naš model. Dakle na skupu podataka koji je mali i u vektorskom obliku, ne možemo da očekujemo da će neuronske mreže biti puno uspešne.[3]

4.1 Neuron i vrste neuronskih mreža

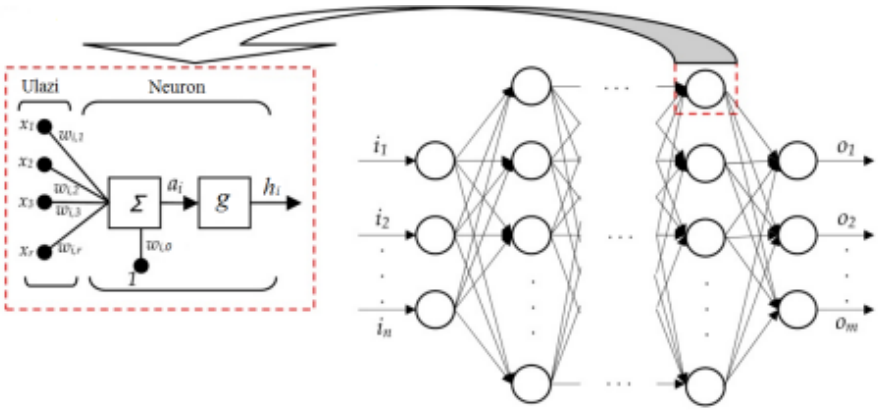
Inspiracija za veštački neuron se nalazi u radu pravih neurona u telu čoveka. Stvarni neuron radi na osnovu impulsa, preko obrađenog impulsa prosleđuje sledećem neuronu i time pravi neuronsku vezu. Sam taj proces se naziva učenje, on u stvari predstavlja jedinu vezu između veštačkog i pravog neurona. Veštački neuron predstavlja uprošten matematički model funkcionisanja biološkog neurona. Mreža koja je formirana za veštački neuron je napravljena za specifičan problem, ali se ta arhitektura naziva neuronska mreža. Zbog poistovećivanja neurona veštačkih sa biološkim, u literaturi se može sresti naziv za veštački nuron *perceptron.*

Termin ***Deep Learning*** *(Duboko učenje)* često se susreće u kontekstu neuronskih mreža i predstavlja važan koncept u oblasti mašinskog učenja. Duboko učenje je tehnika za implementaciju različitih algoritama mašinskog učenja koristeći neuronske mreže sa više slojeva. Ovi slojevi za obradu uče reprezentacije podataka sa više nivoa apstrakcije kako bi razumeli ulazne podatke. Duboko učenje se odnosi na višeslojnu neuronsku mrežu koja uči ekstrakciju karakteristika i klasifikaciju (ili neki drugi zadatak diskriminacije) na sveobuhvatan način. [26]

[3]Za više detalja, pogledajte [14], [21], [22], [2], [15], [5], [4], [12], [27], [23], [25], [31], [7].

Neuronske jedinice, u potpuno povezanoj neuronskoj mreži, računaju linearnu kombinaciju svojih argumenata i nad njom računa neku nelinearnu transformaciju, tzv. *aktivacionu funkciju (activation function)*. Svaki neuron se sastoji od slojeva koji primaju argumente - ulaze i izlaze preko kojih su vezane sa drugim slojevima. Slojevi koji svoje izlazne jedinice prosleđuju drugim slojevima nazivaju se skrivenim slojevima. Kada neuronska mreža ima više od jednog skrivenog sloja, nazivaju se dubokim neuronskim mrežama. U skrivenim slojevima se konstruišu novi atributi, pa svaki sledeći sloj nadograđuje i formira složenije atribute. Model potpuno povezane neuronske mreže možemo definisati na sledeći način:

$$h_0 = x$$
$$h_i = g(W_i h_{i-1} + w_{i0}) \qquad i = 1, 2, 3, ...L$$



Slika 2: Struktura neurona i potpuno povezane neuronske mreže. [23].

gde je x vektor ulaznih promenljivih, L je broj slojeva,W_i je matrica čija j-ta vrsta predstavlja vektor vrednosti parametara jedinice j u sloju i,w_{i0} predstavlja vektor slobodnih članova linearnih kombinacija koje i-tog sloja izračunavaju, a g je nelinearna aktivaciona funkcija. Za vektor v,$g(v)$ predstavlja vektor $(g(v_1), g(v_2), ..., g(v_m))^T$, gde je m dimenzinalnost vektora $g(v)$. Skup svih parametara modela će se ukratko označiti sa w. Važi $f_w(x) = h_L$.

4.2 Konvolutivne neuronske mreže

Konvolutivne mreže su inspirisane konvolucijom, funkcijom centralnog značaja u obradi signala. Konvoluciju primenjujemo za obradu signala, koristeći operaciju definisanu za dvodimenzionalni slučaj nad matricama f i g dimenzija $m \times n$ i $p \times q$.[23] U ovom kontekstu, f je matrica ulaza a g je filter (ili *kernel*) koji izdvaja specifične informacije iz matrice ulaza. Formulacija konvolucije je:

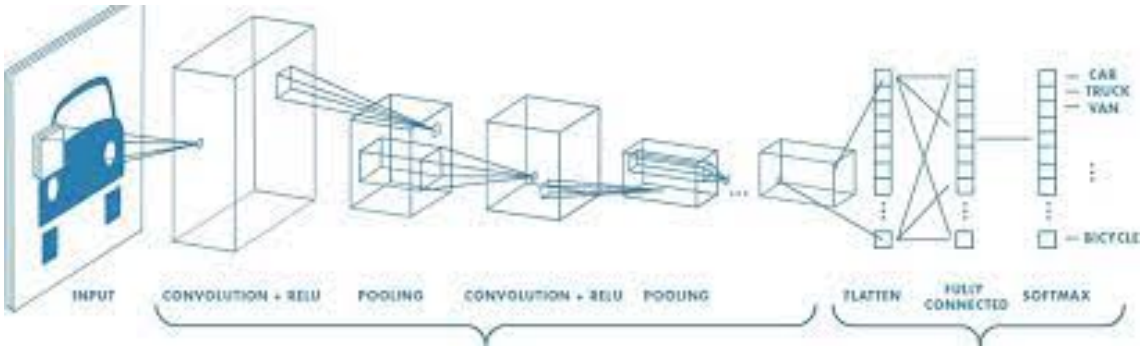
$$(f * g) = \sum_{k=0}^{p-1} \sum_{l=0}^{q-1} f_{i-k,i-l} g_{k,l},$$

Za jednodimenzionalni slučaj, konvolucija je:

$$(f * g) = \sum_{k=0}^{p-1} f_{i-k} g_k.$$

Ove mreže dizajnirane su tako da uče filtere koji, kroz konvoluciju, mogu da prepoznaju specifične karakteristike signala. Obično su strukturirane kao duboke neuronske mreže kako bi mogle da detektuju detalje na slikama kroz konstrukciju složenih struktura.

U konvolutivnim mrežama, struktura slojeva (primer prikazan na slici 3) često se sastoji od smene dve vrste slojeva: konvolutivnog sloja (*convolution layer*) i sloja agregacije (*pooling layer*). Izlaz konvolutivnog sloja podvrgava se nelinearnoj transformaciji pomoću aktivacione funkcije. Tipično se na kraju dodaje potpuno povezana neuronska mreža koja uči na osnovu atributa koje su konstruisali prethodni slojevi.



Slika 3: Struktura konvolutivne mreže [27].

Konvolucija se obično izvodi samo nad delovima ulaznog sloja, jer je dimenzionalnost ulaznog podatka često konstanta.

Pri konstrukciji konvolutivnih mreža, konvolutivni sloj i sloj agregacije se često smenjuju. Ovo pristup smanjuje broj parametara koji treba optimizovati, omogućavajući učenje invarijanti poput rotacije i skaliranja.

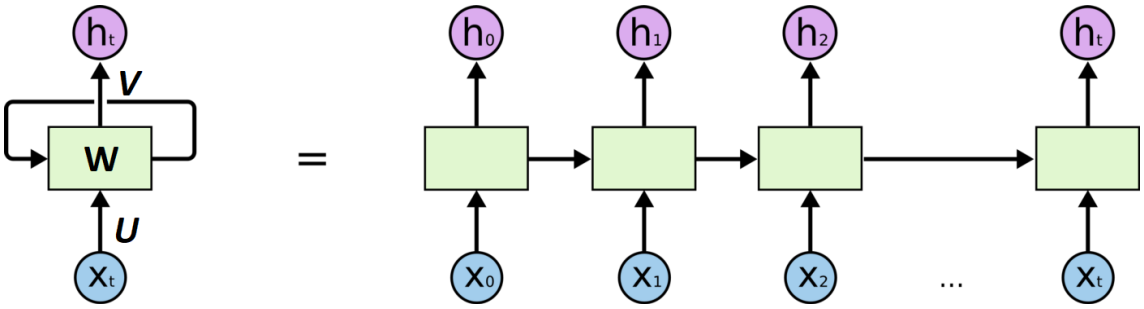
Agregacija sumira informacije dobijene iz prethodnog sloja, obično koristeći maksimalnu ili prosečnu vrednost. Takođe, može se koristiti na nivou kanala kako bi se smanjila veličina izlaza (subsampling). Agregacija omogućava zanemarivanje informacija o lokaciji specifične karakteristike, ali zadržava informaciju o njenoj prisutnosti.

Na kraju konvolutivne i agregacione slojeve prati potpuno povezana neuronska mreža. Između ovih dva dela mreže, postoji međusloj koji preoblikuje izlaz iz konvolutivne mreže u vektor. Ovaj sloj ne sadrži parametre koji se optimizuju, a zbog načina na koji špljošti" izlaz, poznat je kao "flatten" sloj.[23][31][25]

4.3 Rekurentne neuronske mreže

Rekurentne neuronske mreže (RNN) (prikazane na slici 4) su specifičan tip slojevite mreže koji karakteriše formiranje ciklusa, odnosno rekurentnih veza. Ova struktura omogućava neuronima da zadrže informacije o prethodnim izlazima te da uzmu u obzir sopstvene istorijske izlaze prilikom generisanja novih. Kroz ovaj proces, RNN su u stanju da integrišu prethodne rezultate u svoje trenutne operacije, čime postaju izuzetno korisne za predviđanje vremenskih serija i rešavanje problema gde su podaci vremenski zavisni.

Ipak, zbog ove sposobnosti čuvanja informacija, kod rekurentnih mreža se može pojaviti problem nestajućeg ili eksplodirajućeg gradijenta, u zavisnosti od odabrane aktivacione funkcije.



Slika 4: Prikaz strukture rekurentne neuronske mreže. [21]

Formalno možemo opisati rekurentne neuronske mreže sa jednim skrivenim slojem:

$$h^{(0)} = 0$$
$$h^{(t)} = f\left(b_h + Wh^{(t-1)} + Ux^{(t)}\right)$$
$$o^{(t)} = f\left(b_o + Vh^{(t)}\right)$$

za $t = 1, 2, 3, ..., T$ gde je T dužina sekvnce i može se razlikovati od sekvence

do sekvence. Vektor $x^{(t)}$ predstavlja ulaz u trenutku t, $h^{(t)}$ vektor vrednosti skrivenog sloja, a f nelinearna aktivaciona funckija, $o^{(t)}$ je vektor izlaza mreže koji daje predviđanje prave vrednosti $y^{(t)}$. Parametri su matrica transformacije W, ulaza u stanje U, stanja u izlaz V i vektori slobodnih članova b_h i b_o.[21]

Opisani model rekurentne mreže je Tjuring potpun. Broj koraka u kojem je potrebno mreža izvrši računanje je asimptomatski linearan u odnosu na broj koraka potreban Tjuringovoj mašini.

4.4 Duga kratkoročna memorija - LSTM

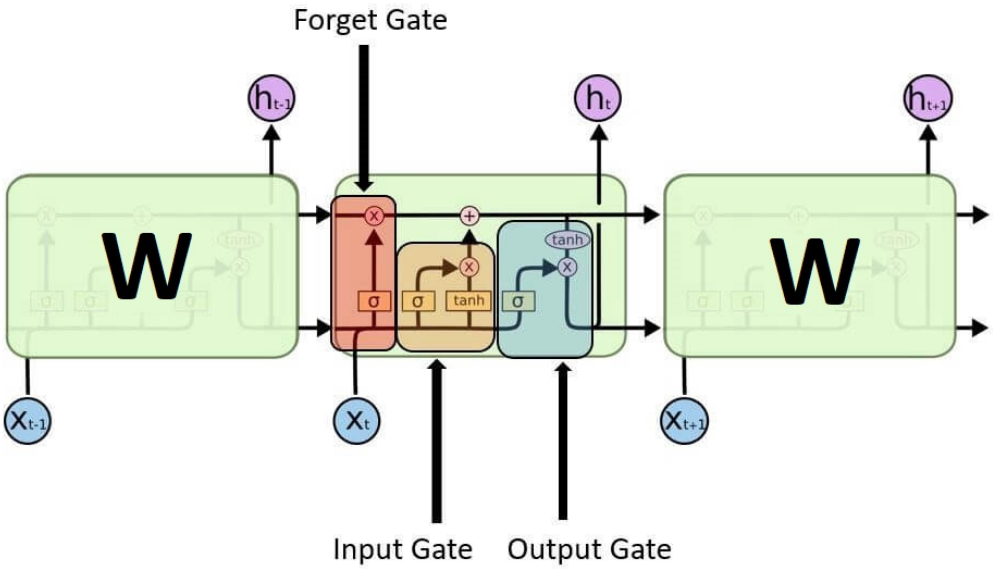
Long Short-Term Memory (LSTM) neuronske mreže su specijalizovana varijanta rekurentnih mreža dizajnirana da reši problem nestajućih i eksplodirajućih gradijenata. Ovaj problem je inherentan za tradicionalne rekurentne mreže zbog njihove potrebe za dugim lancima izračunavanja, što dovodi do eksponencijalnog smanjenja ili porasta vrednosti gradijenta. Još jedan problem sa standardnim rekurentnim mrežama je teškoća u očuvanju dugoročnih informacija, što je rezultat brzog zamagljivanja starih informacija novim.

LSTM mreže uvode koncept ćelije koja skladišti skriveno stanje i tri kapije koje kontrolišu tok informacija - ulaznu, izlaznu i kapiju zaboravljanja. Kroz ove strukture, LSTM mreže uče kada treba da pišu, čitaju ili zaborave informacije.

Formalna definicija LSTM mreže je sledeća:

$$
\begin{aligned}
&c^{(0)} = 0 \\
&h^{(0)} = 0 \\
&z^{(t)} = g(W_z x^{(t)} + U_z h^{(t-1)} + b_z) && \text{Transformisani ulaz} \\
&i^{(t)} = \sigma(W_i x^{(t)} + U_i h^{(t-1)} + b_i) && \text{Ulazna kapija} \\
&f^{(t)} = \sigma(W_f x^{(t)} + U_f h^{(t-1)} + b_f) && \text{Kapija zaboravljanja} \\
&c^{(t)} = z^{(t)} \bigodot i^{(t)} + c^{(t-1)} \bigodot f^{(t)} && \text{Ćelija} \\
&q^{(t)} = \sigma(W_q x^{(t)} + U_q h^{(t-1)} + b_q) && \text{Izlazna kapija} \\
&h^{(t)} = g(c^{(t)}) \bigodot q^{(t)} && \text{Izlaz}
\end{aligned}
$$

Oznake predstavljaju: c kao ćeliju koja skladišti stanje LSTM jedinice, z kao transformisani ulaz, i kao ulaznu kapiju, f kao kapiju zaboravljanja, q kao izlaznu kapiju, h kao izlaz LSTM jedinice i operator $\bigodot$ predstavlja član po član množenje vektora ili matrica, što znači množenje svakog elementa jednog vektora s odgovarajućim elementom drugog vektora. Sve ove kapije koriste odgovarajući skup parametara kako bi odlučile da li i u kojoj meri dozvoljavaju izvršavanje kontrole operacije.[21]



Slika 5: Struktura LSTM mreže. [21]

Kako je prikazano na slici 5, zahvaljujući kapijama koje kontrolišu ulaz u ćeliju, LSTM mreže su u stanju da efikasno čuvaju informacije kroz dugačke sekvence. Ove mreže su pokazale izuzetnu efikasnost u rešavanju problema nestajućih gradijenata i u modelovanju dugoročnih zavisnosti.

5 Aktivacione funkcije

U neuronskim mrežama, svaki neuron sadrži svoju aktivacionu funkciju (activation function), koja definiše izlaz na osnovu linearne kombinacije svojih argumenata. Uloga aktivacione funkcije je delinearizacija na rezultat izlaz neurona, kao i ograničavanje kodomena na neki zatvoren ili otvoren interval. Postoji više različitih vrsta aktivacionih funkcija, ali možemo ih definisati u dve vrste: linearne i nelinearne funkcije. Za treniranje neuronskih mreža, najvažnije svojstvo aktivacionih funkcija je njihova diferencijabilnost. Izbor aktivacione funkcije zavisi od njihovog oblika i izvoda koji je pogodan za propagaciju unazad.

Međutim, ispravljena linearna jedinica (ReLU) trenutno je najčešće korišćena aktivaciona funkcija zbog njenih pogodnih svojstava prilikom optimizacije, uprkos njenom problemu nediferencijabilnosti. Uprkos tome, ReLU se često modifikuje kako bi se izbegli potencijalni problemi u optimizaciji, posebno u segmentima funkcije gde gradijent postaje nula [23]. [4]

5.1 Sigmoidna aktivaciona funkcija

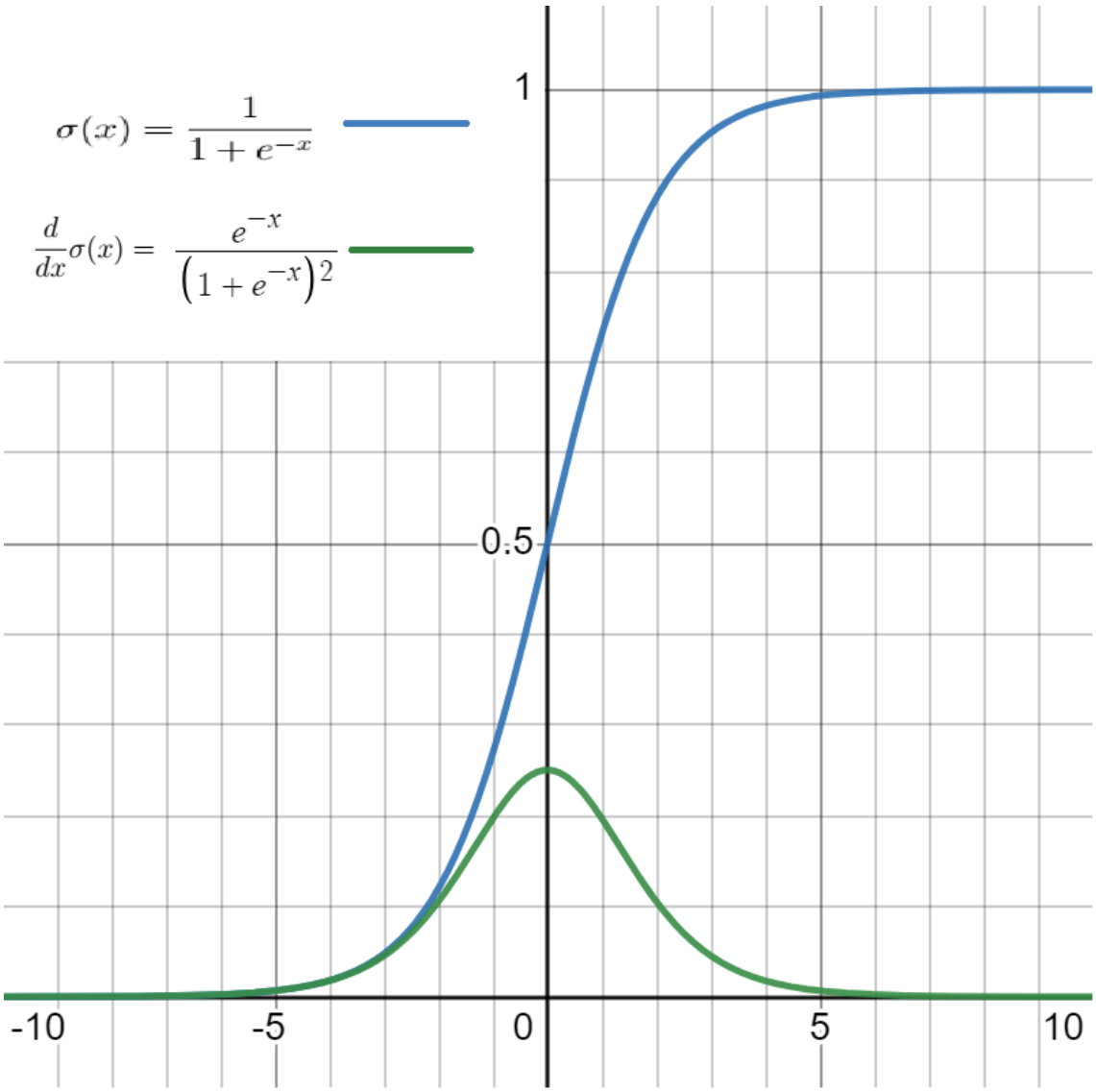
Logistička funkcija, poznatija kao sigmoidna funkcija, predstavlja jedan od ključnih izbora za aktivacionu funkciju pri oblikovanju neuronskih mreža. Definiše se formulom:

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

dok je njen izvod dat kao:

$$\frac{d}{dx}\sigma(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x))$$

[4] Za više detalja, pogledajte [12], [23], [25], [31].



Slika 6: Grafik logističke (sigmoidne) funkcije i njenog izvoda [1]

Kao što se može videti na Slici 6, sigmoidna funkcija transformiše linearnu kombinaciju argumenata na interval između 0 i 1. Stoga je prikladna za primene koje uključuju klasifikaciju, posebno na završnim slojevima mreže. Ipak, sigmoidna funkcija ima izvesne mane. Između ostalog, njena vrednost nije centrirana oko nule, a pojava nestajućih gradijenata otežava efikasnu primenu metode propagacije unazad. Njene karakteristike takođe prouzrokuju izazove u optimizaciji - konstantna je osim u blizini nule, što praktično vodi ka anuliranju gradijenta, a samim tim i sprečava učenje.[12]

5.2 Hiperbolički tangens

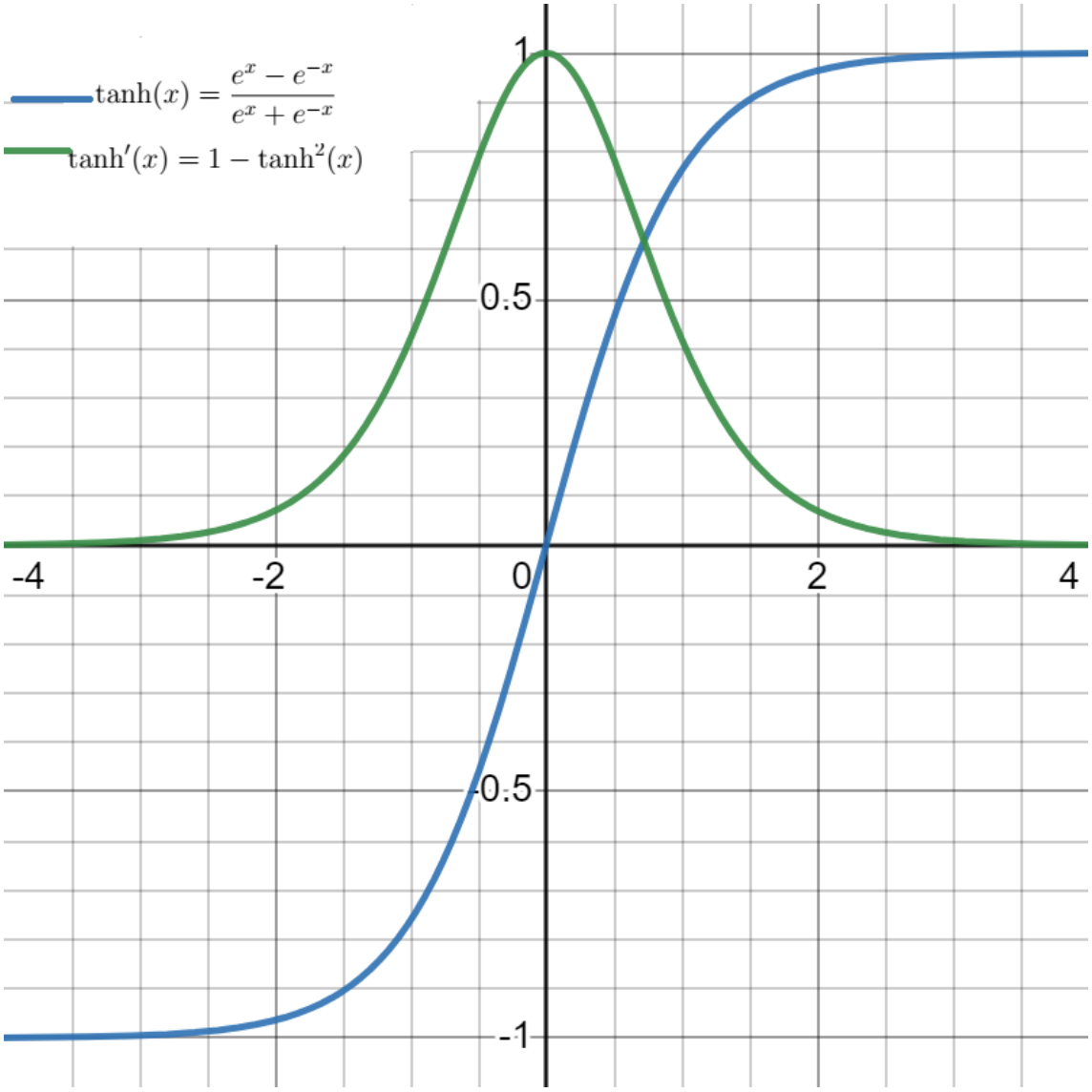
Još jedna mogućnost za aktivacionu funkciju u neuronskim mrežama jeste hiperbolički tangens. On je često odabir u kontekstu rekurentnih neuronskih mreža, gde može biti upotrebljen umesto sigmoidne funkcije. Iako deli sličan

oblik sa sigmoidnom funkcijom, hiperbolički tangens je u blizini nule sličniji funkciji identiteta, što ga čini bližim linearnoj funkciji u odnosu na sigmoidnu. Ova osobina omogućava lakšu optimizaciju. Hiperbolički tangens preslikava ulaz na interval $(-1, 1)$. Definisan je formulom:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

izvod hiperboličkog tangensa je definisan kao:

$$\tanh'(x) = 1 - \tanh^2(x).$$



Slika 7: Grafik funkcije i izvoda hiperboličkog tangensa [1]

Kao što se može videti na Slici 7, postoji jasna veza između sigmoidne funkcije i hiperboličkog tangensa, koja je izražena formulom $\tanh(x) = 2\sigma(2x) - 1$. Iako hiperbolički tangens ima strmiji gradijent u odnosu na sigmoidnu funkciju, problem nestajućih gradijenata ostaje prisutan.[12]

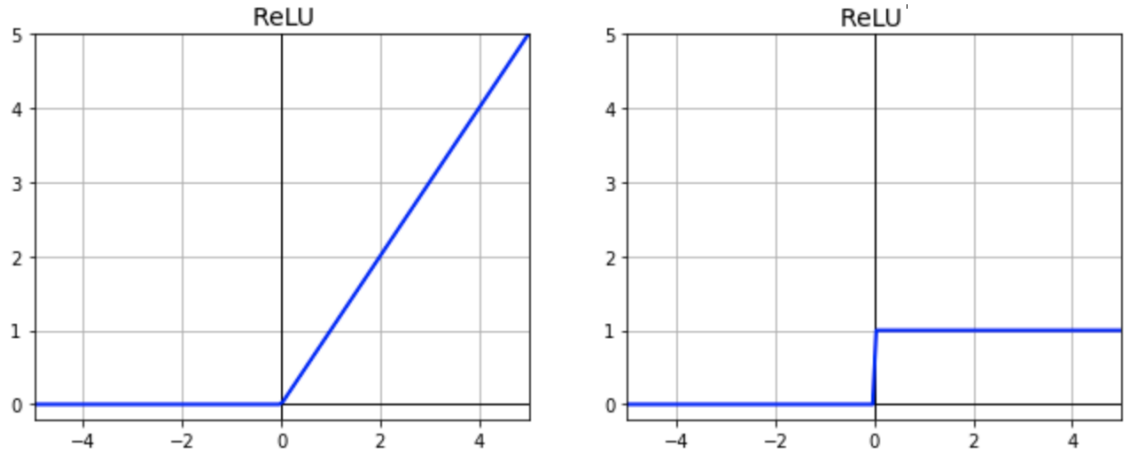
5.3 Ispravljačka linearna jedinica (ReLU)

Ispravljačka linearna jedinica, poznatija kao ReLU, postala je popularan izbor za aktivacionu funkciju u neuronskim mrežama zahvaljujući svojoj računskoj efikasnosti u odnosu na prethodno opisane funkcije. Njena sposobnost da omogući brže treniranje i konvergenciju neuronske mreže čini je privlačnom opcijom. ReLU funkcija je definisana na sledeći način:

$$rlu(x) = \max(0, x) = \begin{cases} 0 & \text{ako je } x < 0 \\ x & \text{ako je } x \geq 0 \end{cases},$$

dok je njen izvod:

$$rlu'(x) = \begin{cases} 1 & \text{ako je } x > 0 \\ 0 & \text{ako je } x \leq 0 \end{cases}.$$



Slika 8: Grafik ispravljačke linearne jedinice (ReLU)[1]

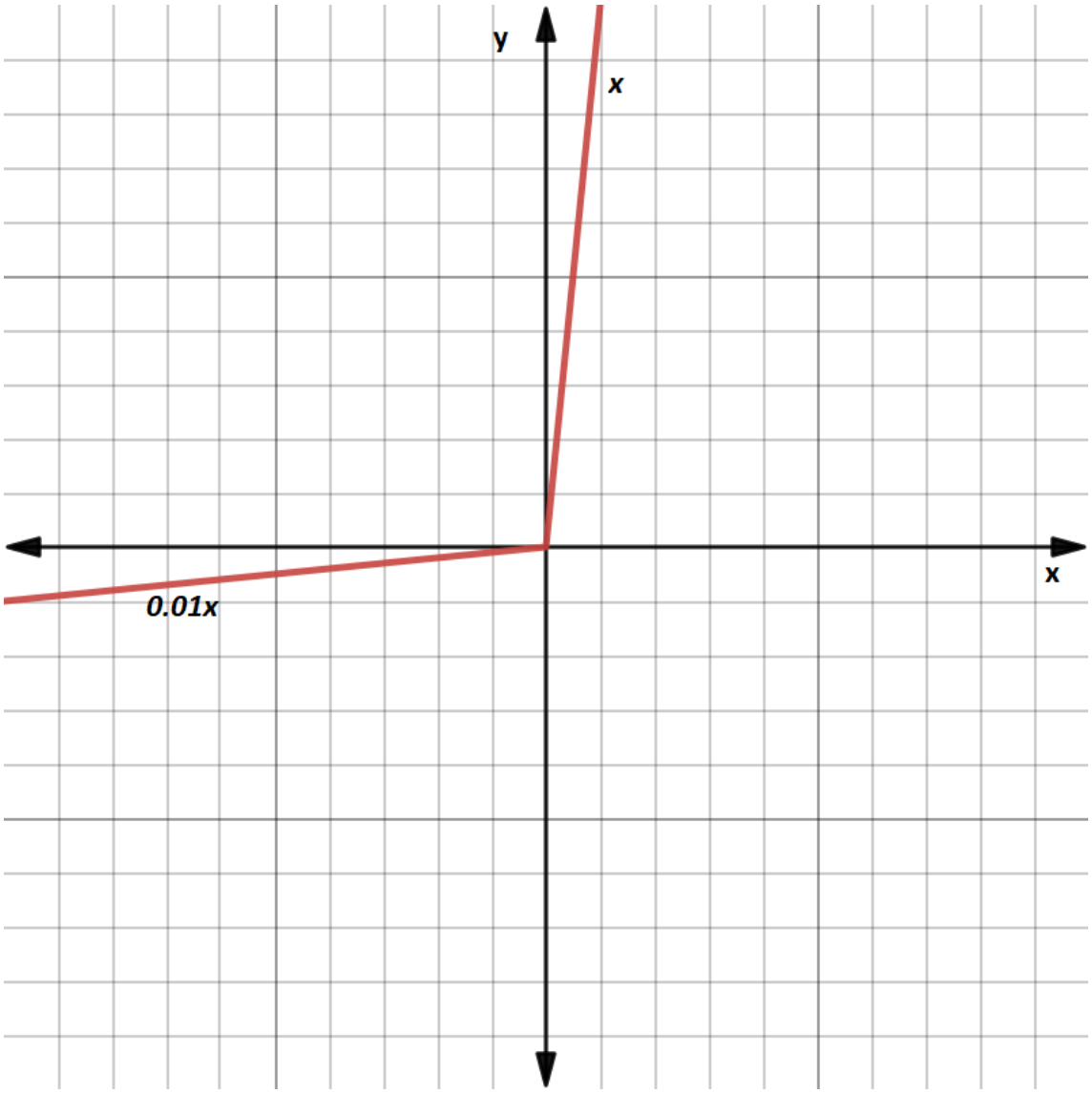
Kao što je prikazano na Slici 8, ova aktivaciona funkcija je učestalo korišćena zahvaljujući svojim povoljnim karakteristikama pri optimizaciji. Verovatnoća nailaska na nediferencijabilnu tačku nije velika, a ukoliko se i pojavi, greška se obično nadoknadi u kasnijem toku optimizacije. Međutim, jedan od nedostataka ove funkcije je fenomen "umirućeg"gradijenta, tj. situacija u kojoj neuron postaje neaktivan prilikom ažuriranja težina. Zbog toga se ReLU najčešće koristi u unutrašnjim slojevima, budući da njegovo preslikavanje na neograničeni interval nije posebno korisno na izlazu neuronske mreže.[12]

5.4 Cureća ispravljačka funkcija - Leaky ReLU

Cureća ispravljačka linearna jedinica, poznata kao Leaky ReLU, predstavlja unapređenje ReLU funkcije. Ne samo da omogućava brže učenje mreže, već

takođe obezbeđuje bolju

$$lrlu(x) = \max(0.01x, x) = \begin{cases} 0.01x & \text{ako je } x < 0 \\ x & \text{ako je } x \geq 0 \end{cases},$$

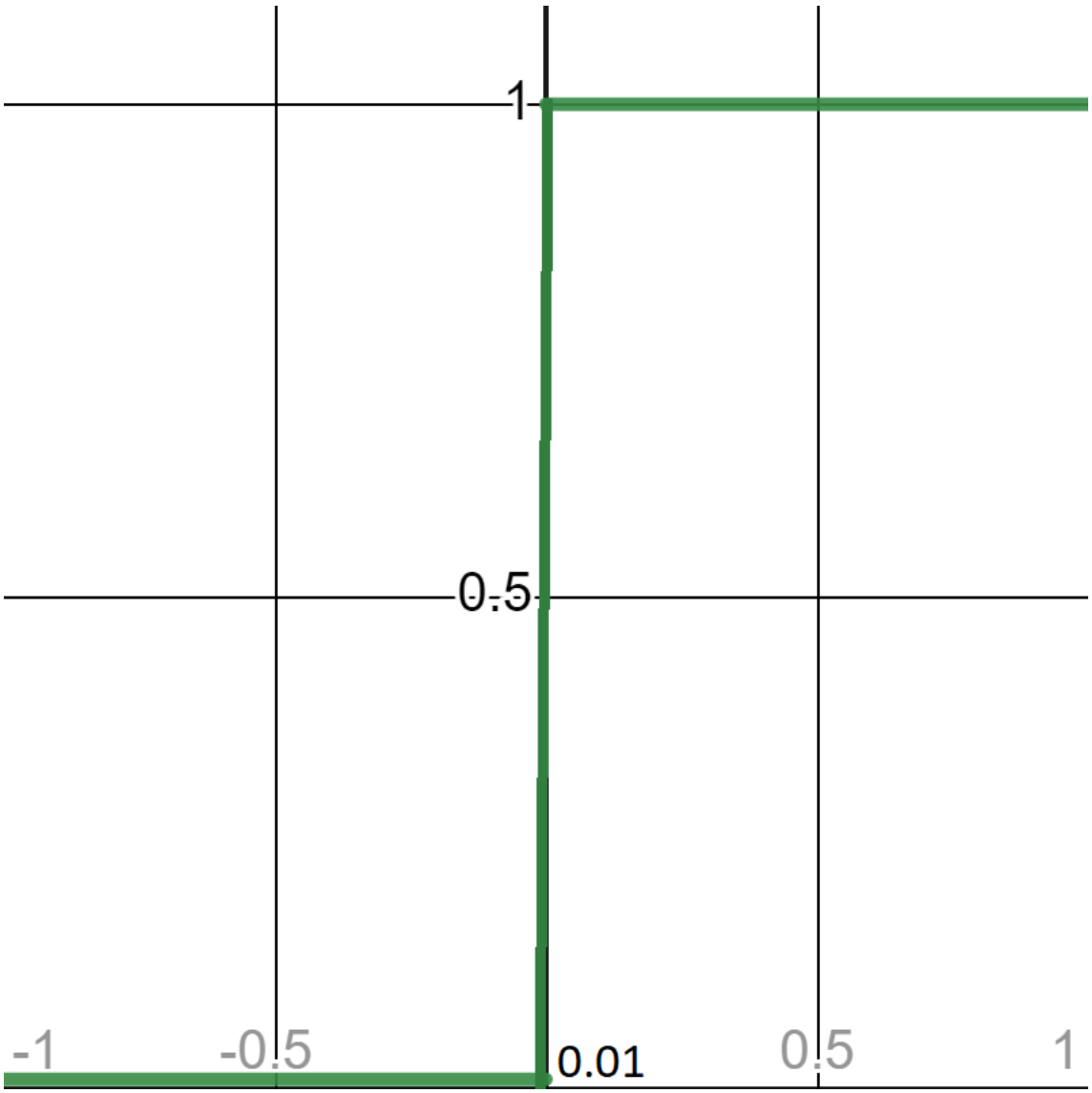


Slika 9: Grafik cureće ispravljačke linearne jedinice (Leaky ReLU)[1]

Kako je ilustrovano na Slici 9, ključna razlika između standardne i cureće ReLU funkcije je u tretmanu negativnih vrednosti x. Dok standardna ReLU funkcija takve vrednosti zanemaruje, cureća ReLU funkcija ih skalira faktorom od $0.01x$, efektivno prevazilazeći osnovni problem sa ReLU funkcijom.[12]

Izvod Leaky ReLU funkcije:

$$lrlu'(x) = \begin{cases} 0.01 & \text{ako je } x < 0 \\ 1 & \text{ako je } x \geq 0 \end{cases}$$



Slika 10: Grafik izvoda cureće ispravljačke linearne jedinice (Leaky ReLU)[1]

5.5 Parametarska ispravljačka funkcija - Parametric ReLU

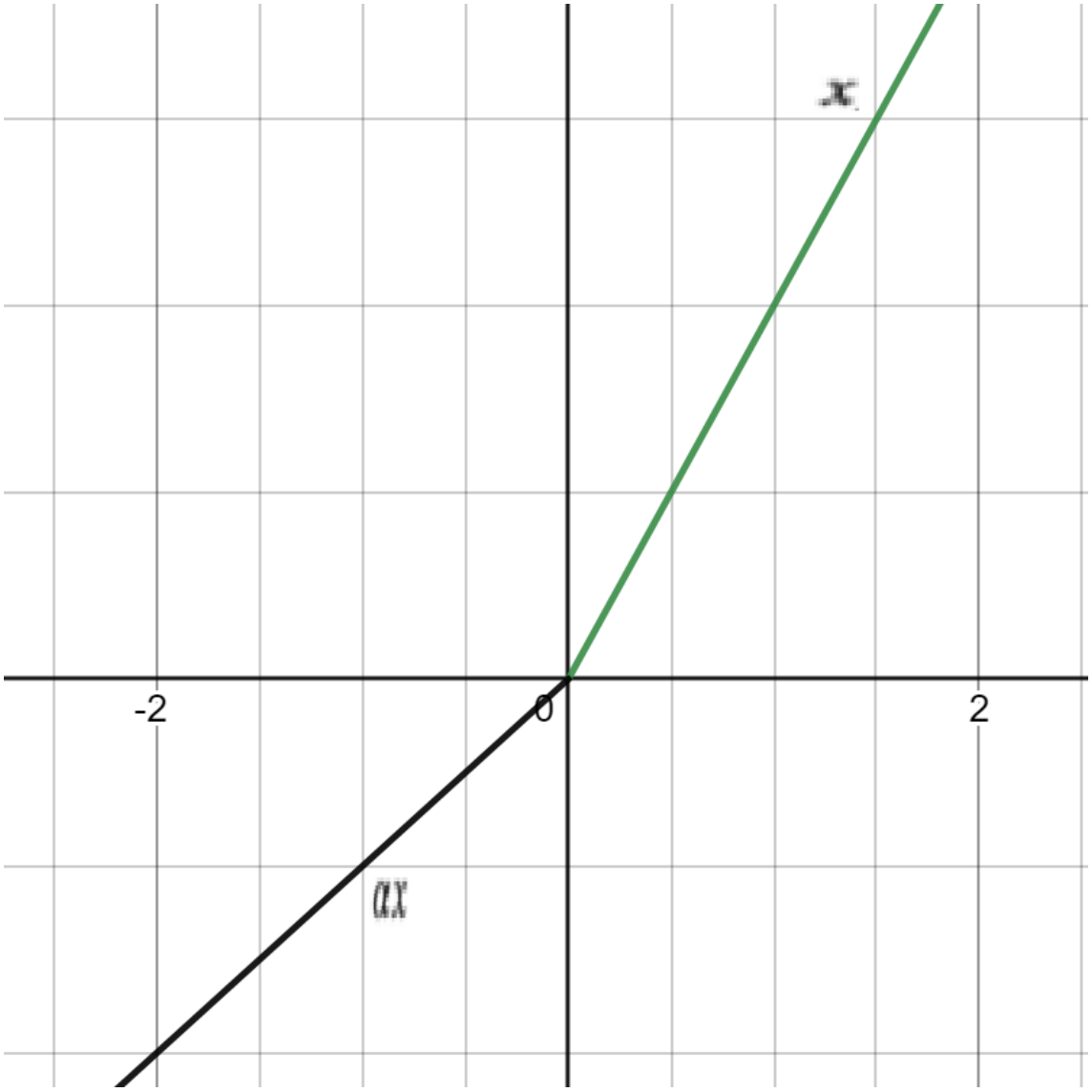
Parametarska ispravljačka funkcija (Parametric ReLU) predstavlja varijaciju Cureće ReLU aktivacione funkcije. Slično standardnoj ReLU funkciji, Parametarska ReLU koristi pozitivni deo svojih ulaza za vrijednosti $x \geq 0$, dok za $x < 0$ umesto nule koristi malu konstantu αx. Ova mala modifikacija omogućava funkciji da održi mali gradijent kada se vrednost x približava nuli, što može doprineti boljem učenju dubokih neuralnih mreža.[12]

Formalno, parametarska ReLU se definiše sledećom formulom:

$$prlu(x) = \begin{cases} \alpha x & \text{ako je } x < 0 \\ x & \text{ako je } x \geq 0 \end{cases},$$

Slika 11 prikazuje grafik funkcije PReLU. Kao što se može videti, za $x \geq 0$ funkcija je identična identitetnoj funkciji, dok je za $x < 0$ funkcija linearna sa

nagibom α.



Slika 11: Grafik funkcije Parametarske ReLU

Izvod PReLU funkcije je predstavljen sledećom formulom:

$$prlu'(x) = \begin{cases} \alpha & \text{ako je } x < 0 \\ 1 & \text{ako je } x \geq 0 \end{cases}$$

Izvod je konstantan za sve vrednosti x i ravan je 1 za $x \geq 0$ i α za $x < 0$, kako je prikazano na slici 12.

Slika 12: Izvod funkcije Parametarske ReLU, $\alpha = 0.5$. [1]

Parametarska ReLU je pokazala izvanredne rezultate u poboljšanju performansi dubokih neuralnih mreža, posebno na problemima sa velikim količinama podataka i složenim arhitekturama.

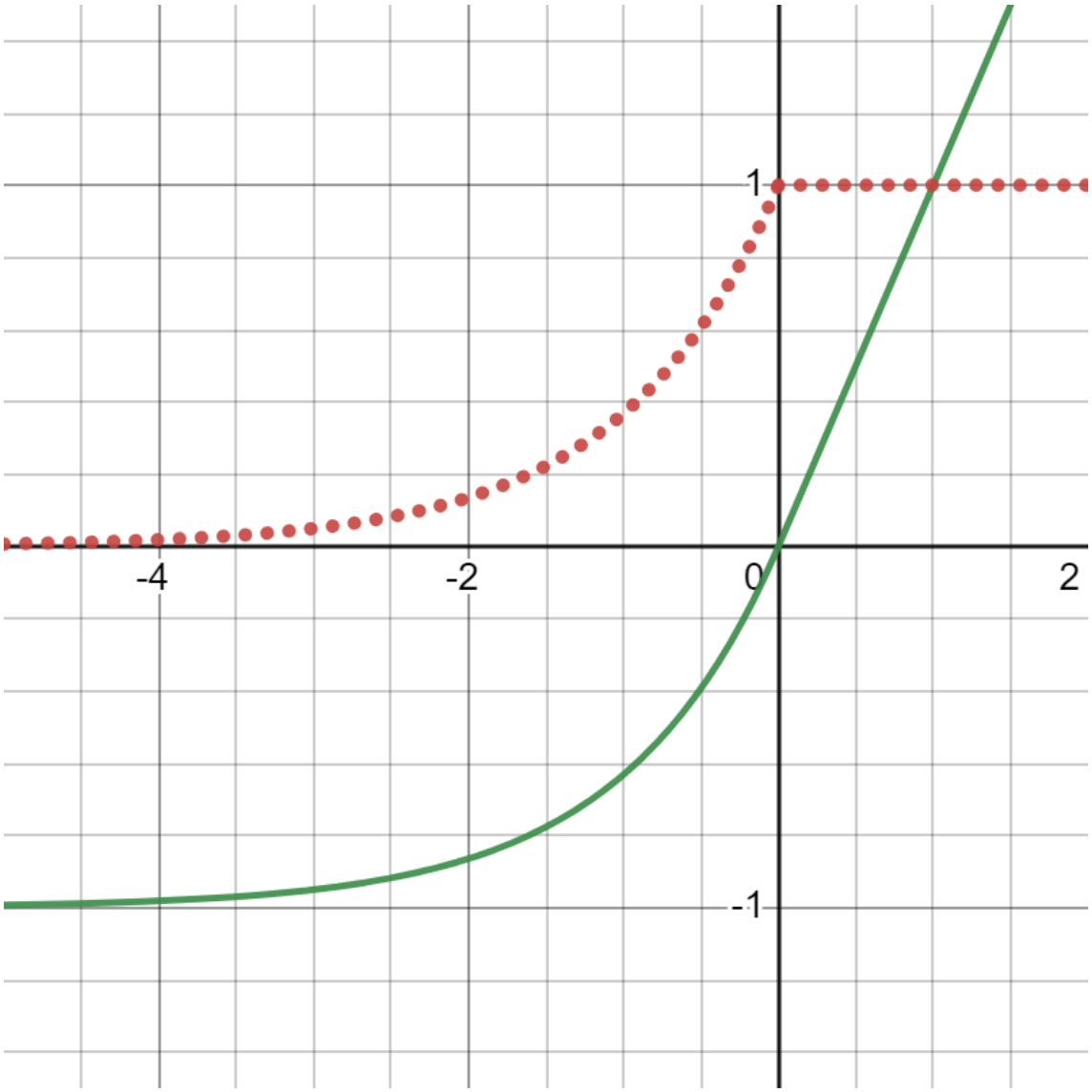
5.6 Eksponencijalna linearna funkcija - Exponential Linear Unit

Eksponencijalna linearna funkcija (Exponential Linear Unit - ELU) predstavlja još jednu varijantu ReLU aktivacione funkcije. Za razliku od ReLU i njegovih varijacija, ELU funkcija za vrednosti $x < 0$ ne prikazuje linearno ponašanje, već oblikuje eksponencijalnu krivu.[12]

Formalno, ELU se definiše formulom:

$$elu(x) = \begin{cases} \alpha(e^x - 1) & \text{ako je } x < 0 \quad \alpha > 0 \\ x & \text{ako je } x \geq 0 \end{cases},$$

Slika 13 ilustruje grafik ELU funkcije. Vidimo da za $x \geq 0$ funkcija deluje kao identitet, dok za $x < 0$ eksponencijalna kriva omogućava blagi gradijent, što može doprineti boljem učenju mreže.



Slika 13: Grafik funkcije i izvoda ELU, $\alpha = 1$. [1]

Izvod ELU funkcije je dat sledećom formulom:

$$elu'(x) = \begin{cases} \alpha e^x & \text{ako je } x < 0 \quad \alpha > 0 \\ 1 & \text{ako je } x \geq 0 \end{cases}$$

ELU, kao aktivaciona funkcija, koristi se u različitim arhitekturama dubokih neuronskih mreža, omogućavajući bolje učenje modela zahvaljujući svojoj sposobnosti da održava ne-nule gradijente za negativne vrednosti ulaza.

6 Funkcija troška

Učenje veštačkih neuronskih mreža zahteva pažljiv odabir funkcije koja opisuje performanse mreže tokom obučavanja. Ova funkcija, poznata kao funkcija troška (cost function), funkcija gubitka (loss function) ili funkcija greške (error function) [2] [31], ima ključnu ulogu u procesu optimizacije mreže, jer se njena vrednost nastoji minimizovati tokom obuke. Na osnovu ove funkcije, u svakoj iteraciji obučavanja, određuje se greška na izlazu mreže, koja se potom koristi za prilagođavanje težina u procesu propagacije unazad.

Cilj koji treba optimizirati (u mašinskom učenju, funkcija troška) obično se izračunava na temelju izmjerenih očekivanih vrijednosti.[6] U suštini, ova funkcija može biti bilo koja matematička funkcija koja uspostavlja vezu između dva ulazna podatka. Međutim, često se bira na osnovu vrste problema koji se rešava i specifičnih karakteristika ulaza i izlaza mreže.

Na primer, kada se rešava problem regresije, srednje apsolutno odstupanje (mean absolute error, MAE) može biti pogodna funkcija troška, jer efikasno meri razliku između stvarnih i predviđenih vrednosti. S druge strane, u zadacima klasifikacije s više klasa, funkcija gubitka zasnovana na unakrsnoj entropiji (cross-entropy) se često koristi, jer precizno kvantifikuje razliku između stvarne i predviđene distribucije verovatnoća. Izbor funkcije troška takođe može zavisiti od uticaja "outliera"(ekstremnih slučajeva). [22]

Dakle, specifičnost problema koji se rešava utiče na izbor funkcije troška i, u određenoj meri, na izbor metoda treniranja. Kroz pravilno odabranu funkciju troška, neuronska mreža može efikasno učiti i prilagoditi se različitim vrstama problema.[5]

6.1 Srednja apsolutna razlika

Funkcija koja se koristi za procenu performansi algoritama ima ključnu ulogu u procesu obučavanja i optimizacije. Jedna od takvih funkcija je srednja apsolutna razlika, poznata kao Mean Absolute Error (MAE) ili L1 gubitak. Ova funkcija ocenjuje tačnost modela kroz merenje prosečne apsolutne razlike između stvarnih i predviđenih vrednosti.

MAE se izračunava prema formuli:

$$L(x) = \frac{1}{N} \sum_{i=0}^{N-1} |\hat{y}_i - y_i|,$$

gde je N broj instanci u trening skupu, a $\hat{y}_i$ i y_i predstavljaju, redom, predviđenu i pravu vrednost promenljive za i-tu instancu.

[5] Za više detalja, pogledajte [2], [12], [22], [23], [6], [31].

Na primer, zamislimo da obučavamo model za predviđanje cene automobila na osnovu njihovih karakteristika. U ovom slučaju, ulazne vrednosti su numeričke, a izlazne vrednosti predstavljaju cene automobila. Nakon obuke, model će generisati predviđanja za cene automobila, koja se zatim upoređuju sa stvarnim cenama. MAE se koristi za kvantifikaciju razlike između predviđenih i stvarnih cena, gde manja vrednost MAE ukazuje na bolju tačnost modela.

Srednja apsolutna razlika (MAE) se uglavnom koristi u regresionim i klasifikacionim problemima gde su ulazne i izlazne vrednosti numeričke i manje vrednosti. Ova funkcija omogućava efikasno procenjivanje performansi modela, čime se olakšava proces optimizacije i poboljšanja modela u rešavanju različitih problema.[23][31]

6.2 Srednjakvadratna greška

Jedna od često korišćenih funkcija je funkcija srednjekvadratne greške, poznata i kao L2 gubitak.

Formula za srednjekvadratnu grešku je:

$$L(x) = \frac{1}{N} \sum_{i=0}^{N-1} ||\hat{y}_i - y_i||_2^2,$$

gde je N broj instanci u trening skupu, a $\hat{y}_i$ i y_i predstavljaju predviđene i stvarne vrednosti promenljive za i-tu instancu.

Primera radi, pretpostavimo da radimo na problemu klasifikacije vrsta biljaka na osnovu njihovih karakteristika. Model će generisati predviđanja za vrste biljaka, koja se zatim upoređuju sa stvarnim vrstama. Srednjekvadratna greška koristi se za procenu razlike između predviđenih i stvarnih vrsta biljaka.

Srednjekvadratna greška ima nekoliko prednosti, uključujući činjenicu da uvek daje pozitivne rezultate, što olakšava njenu interpretaciju. Takođe, ova metrika efikasno razlikuje velike i male greške, što omogućava lakše identifikovanje i ispravljanje grešaka u modelu. Korišćenje L2 gubitka u procesu obučavanja i evaluacije modela doprinosi boljoj optimizaciji i većoj tačnosti u rešavanju različitih problema.[23][31]

6.3 Koren srednjakvadratne greške

Funkcija korena srednjekvadratne greške (Root mean squared error) modifikacija srednjekvadratne greške i pruža dodatne prednosti prilikom treniranja modela.

Formula za koren srednjekvadratne greške je:

$$L(x) = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} ||\hat{y}_i - y_i||_2^2},$$

gde je N broj instanci u trening skupu, a $\hat{y}_i$ i y_i predstavljaju predviđene i stvarne vrednosti promenljive za i-tu instancu.

Pretpostavimo da radimo na problemu procene cena nekretnina na osnovu njihovih karakteristika, kao što su kvadratura, broj soba, i lokacija. Model će generisati predviđanja cena nekretnina, koja se zatim upoređuju sa stvarnim cenama. Koren srednjekvadratne greške koristi se za procenu razlike između predviđenih i stvarnih cena nekretnina.

Glavna prednost korišćenja korena srednjekvadratne greške u odnosu na srednjekvadratnu grešku je što nudi razlike greški skalirane na pravu razmeru pre nego što su kvadrirane. Ovaj pristup pruža bolji uvid u greške u odnosu na ciljanu promenljivu, što može biti korisno za bolje razumevanje i optimizaciju modela.[23][31]

6.4 Prelomni gubitak

Jedna od funkcija gubitka koja se koristi u određenim klasifikacionim problemima je prelomni gubitak (hinge loss). Iako se najčešće koristi u algoritmima poput SVM, prelomni gubitak može pružiti bolje rezultate u određenim situacijama klasifikacije.

Matematička formula za prelomni gubitak je:

$$L(x) = \frac{1}{N}\sum_{i=0}^{N-1} max(0, 1 - y_i\hat{y}_i)$$

gde je N broj instanci u trening skupu, a y_i i $\hat{y}_i$ predstavljaju stvarne i predviđene vrednosti za i-tu instancu.

Zamislimo primer klasifikacije e-poruka na osnovu njihovog sadržaja kao 'spam' ili 'nije spam'. U ovom scenariju, y_i predstavlja stvarnu klasifikaciju e-poruke (1 za 'spam', -1 za 'nije spam'), dok $\hat{y}_i$ označava predviđenu klasifikaciju dobijenu iz modela. Korišćenje prelomnog gubitka kao funkcije gubitka može doprineti efikasnijem i preciznijem klasifikacionom modelu u ovakvim specifičnim slučajevima.[23][31]

6.5 Huberova funkcija gubitka

Huberova funkcija gubitka je funkcija koja se često koristi u problemima regresije. Ova funkcija troška pokazuje manju osetljivost na pojedinačne uzorke

koji su daleko od većine uzoraka, za razliku od srednjekvadratne greške. Huberova funkcija gubitka tretira greške kvadratno samo u određenom intervalu, koji je definisan koeficijentom δ:

$$L(x) = \frac{1}{N}\sum_{i=0}^{N-1} L_\delta(y_i\hat{y}_i)$$

$$L_\delta(y_i\hat{y}_i) = \begin{cases} \frac{1}{2}(\hat{y}_i - y_i)^2, & |\hat{y}_i - y_i| \leq \delta \\ \delta|\hat{y}_i - y_i| - \frac{1}{2}\delta^2, & |\hat{y}_i - y_i| > \delta \end{cases}$$

Recimo da radimo na problemu predviđanja potrošnje energije za domaćinstva na osnovu raznih karakteristika, kao što su veličina domaćinstva, broj električnih uređaja i klimatski uslovi. Ukoliko se pojave neki ekstremni slučajevi potrošnje, koji su daleko od većine ostalih, Huberova funkcija gubitka će biti manje osetljiva na ove pojedinačne uzorke. Tako će model bolje generalizovati i pružati pouzdanija predviđanja za tipične slučajeve potrošnje energije.

Dakle, Huberova funkcija gubitka pruža prednost u odnosu na srednjekvadratnu grešku u slučajevima kada je potrebno smanjiti uticaj ekstremnih vrednosti, čime se poboljšava performansa modela na širokom spektru regresionih problema.[23][31]

6.6 Međuentropijska funkcija gubitka

Međuentropijska funkcija gubitka(eng. *cross-entropy loss*) je metrika koja meri razliku između dva vektora informacija. Ovaj tip funkcije gubitka koristi logaritamsku funkciju troškova, čija vrednost se povećava kada izlaz neuronske mreže odstupa od stvarnih oznaka. Glavna prednost ovog pristupa je što više kažnjava veće odstupanja, naročito kada neuronska mreža sa velikom sigurnošću daje netačne predikcije.

Funkcija gubitka međuentropije se najčešće primenjuje u problemima binarne klasifikacije, gde neuronska mreža generiše izlaze samo 1 ili 0. U ovom kontekstu, zamislite sledeći primer:

Recimo da imamo skup podataka za obuku koji se sastoji od slika mačaka (oznaka 1) i pasa (oznaka 0). Naš cilj je da obučimo neuronsku mrežu koja će ispravno klasifikovati slike u jednu od ove dve kategorije.

Kako bismo procenili koliko dobro mreža vrši klasifikaciju, koristimo funkciju troška međuentropije da bismo izračunali razliku između stvarnih oznaka i predikcija koje pruža neuronska mreža. Ako mreža daje visoku verovatnoću za netačnu klasifikaciju, funkcija gubitka međuentropije će generisati veću vrednost gubitka, signalizirajući da mreža mora da poboljša svoju sposobnost klasifikacije.

U ovom primeru, ako neuronska mreža daje izlaz blizu 1 za sliku mačke, to znači da mreža pravilno klasifikuje sliku, a funkcija gubitka međuentropije će imati nisku vrednost. S druge strane, ako mreža daje izlaz blizu 1 za sliku psa, to znači da je klasifikacija pogrešna, a funkcija gubitka međuentropije će imati visoku vrednost, ukazujući na potrebu za prilagođavanjem parametara mreže kako bi se poboljšala klasifikacija.

Pretpostavimo da je dat skup podataka za trening, $\mathbb{D} = \left\{\left(x_i, y_i\right), ..., \left(x_i, y_i\right)\right\}$ and $y_i \in \left\{0, 1\right\}$. Funkciju možemo zapisati kao

$$\hat{y_i} = f(x_i; w)$$

Gde je θ parametar mreže (težina i bias). Njega možemo izraziti kroz Bernulijevu raspodelu:

$$P\Big(x_i \to y_i | w\Big) = \hat{y_i}^{y_i}(1 - \hat{y_i})^{1-y_i},$$

Vervoatnoća na ulaznom skupu podataka je data sa :

$$P\Big(x_1, x_2, ..., x_N, y_1, y_2, ..., y_N\Big) = \prod_{i=1}^{N}(x_i \to y_i | w) = \prod_{i=1}^{N} \hat{y_i}^{y_i}(1 = \hat{y_i})^{1-y_i}$$

Ako primenimo negativni logaritam dobijamo :

$$-\log P\Big(x_1, x_2, ..., x_N, y_1, y_2, ..., y_N\Big) = -\log \prod_{i=1}^{N} \hat{y_i}^{y_i}(1 = \hat{y_i})^{1-y_i}$$

Te konačnu funkciju za međuentropijsku funkciju :

$$L(x) = -\sum_{i=0}^{N-1}(y_i \log\left(\hat{y_i}\right) + (1 - y_i) \log\left(1 - \hat{y_i}\right))$$

Ova jednačina je data za slučaj binarne klasifikacije, ali u slučaju višeklasne klasifikacije, sledeća jednačina je pogodnija:

$$L(x) = -\sum_{c=0}^{N-1} y_{i,c} \log(\hat{y_{i,c}})$$

gde je $y_{i,c}$ očekivani binarni indikator verovatnoće i pripada klasi c.[23][31]

6.7 Kulbak-Lejbler razlika

Kulbak-Lejbler razlika (Kullback-Leibler divergence) predstavlja metodu za procenu razlike između dve verovatnoćne raspodele, p i q. Ova metrika se izračunava pomoću sledeće formule:

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx.$$

Kada su $p(x)$ i $q(x)$ identične raspodele, KL divergencija je 0 na svim tačkama. Kulbak-Lejblerova divergencija se često koristi u kontekstu generativnih modela.

Kao primer, pretpostavimo da želimo proceniti koliko dobro generativni model, kao što je Variational Autoencoder (VAE), aproksimira stvarnu raspodelu podataka. Prvobitna raspodela podataka, $p(x)$, može biti stvarna raspodela slika lica, dok generativni model, $q(x)$, generiše nove slike lica na osnovu naučenih parametara.

U ovom scenariju, Kulbak-Lejblerova razlika može se koristiti za ocenu koliko se dobro generisane slike lica ($q(x)$) poklapaju sa stvarnom raspodelom slika lica ($p(x)$). Ako je KL divergencija blizu nule, to znači da generativni model uspešno aproksimira stvarnu raspodelu podataka. Ukoliko je KL divergencija veća, to znači da postoje značajne razlike između dve raspodele, što ukazuje na potrebu za daljim poboljšanjem generativnog modela.

Kulbak-Lejbler razlika se može koristiti i u drugim primenama, kao što su kompresija podataka, prenos znanja između modela i otkrivanje anomalija. U svakom od ovih slučajeva, KL razlika pruža meru koliko se dve verovatnoćne raspodele razlikuju, što pomaže u optimizaciji modela ili identifikaciji neobičnih ponašanja.[18][31]

6.8 Jensen- Šenon razlika

Poput KL razlike, Jensen- Šenon razlika (Jensen-Shannon divergence) takođe služi kao mera razlike između dve verovatnoćne raspodele, ali je simetrična i ima glađe svojstvo. Jensen-Shannon divergencija se izračunava pomoću sledeće formule:

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}\Big(p||\frac{p+q}{2}\Big) + \frac{1}{2}D_{KL}\Big(q||\frac{p+q}{2}\Big)$$

Jensen- Šenon razlika pokazuje bolje ponašanje u odnosu na KL divergenciju kada su vrednosti p i q male ili kada postoji mala verovatnoća za određeni događaj u jednoj raspodeli, ali ne i u drugoj.

Kao primer, pretpostavimo da imamo dva različita modela mašinskog učenja, koji daju verovatnoće za iste oznake. Želimo da uporedimo koliko se njihove raspodele razlikuju kako bismo odlučili koji model bolje generalizuje podatke. U ovom kontekstu, Jensen-Shannon divergencija može biti korisna metrika za upoređivanje modela, jer je simetrična i manje osetljiva na male verovatnoće.

U drugom primeru, recimo da radimo na zadatku prepoznavanja govora i imamo dve verovatnoćne raspodele za foneme, jednu iz stvarnog govora i drugu iz sintetizovanog govora. Jensen- Šenon razlika može se koristiti kako bi se procenila koliko se dobro sintetizovani govor poklapa sa stvarnim govorom, pri čemu se uzimaju u obzir i retke pojave fonema.

U oba primera, Jensen- Šenon razlika pruža korisnu metriku za upoređivanje verovatnoćnih raspodela i identifikaciju razlika u ponašanju modela ili podataka.[19] [31]

7 Propagacija unazad

Optimizacija neuronskih mreža često je izazovna zbog nekonveksne prirode problema. Moguće su različite poteškoće, kao što su zaglavljivanje u lokalnim minimumima, teškoće primene optimizacionih metoda ili sporiji rad algoritma. Da bi se minimizovala ciljna funkcija, izračunava se gradijent za svaku težinu, tj. $w_{i,j}^k$, tako što se izračuna $\frac{\partial J(x)}{x_{i,j}^k}$. Ovde $J(x)$ predstavlja funkciju troška, a $x_{i,j}^k$ je težina gradijenta funkcije J. Algoritam propagacije unazad pokazao se kao izuzetno efikasan pristup izračunavanju parcijalnih derivacija složene ciljne funkcije u neuronskim mrežama s više slojeva. Ovaj algoritam smatra se jednim od najvažnijih u oblasti mašinskog učenja.

Prilikom izračunavanja derivacija za ugnježdene funkcije, koristi se **pravilo ulančavanja**. Ovo pravilo se primenjuje na funkcije $g : \mathbb{R}^m \to \mathbb{R}^n$ i $f : \mathbb{R}^n \to \mathbb{R}$:

$$\partial_i(f \circ g) = \sum_{j=1}^{n} (\partial_j f \circ g)\partial_j g_j$$

Zamislimo da imamo funkciju $F(x)$ koja se sastoji od pet funkcija $f(x), g(x), h(x), u(x), v(x)$ ugnježdenih na sledeći način: $F(x) = f(g(h(u(v(x)))))$. Tada primenjujemo **pravilo ulančavanja** na $F(x)$:

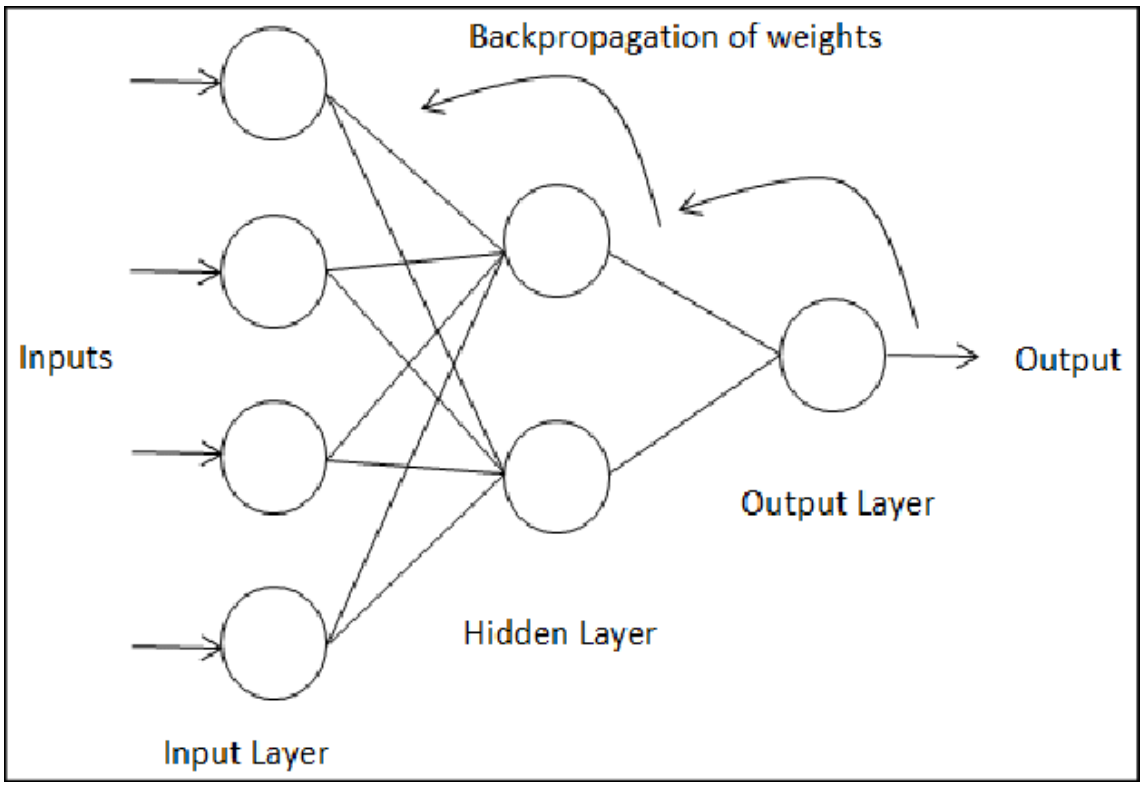
$$\frac{dF}{dx} = \frac{d}{dx}F(x) = \frac{d}{dx}f(g(h(u(v(x))))) = \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{du} \cdot \frac{du}{dv} \cdot \frac{dv}{dx}$$

Ovaj pristup omogućava efikasno izračunavanje izvoda za složene funkcije, što je ključni korak u optimizaciji neuronskih mreža. [25]

Radi boljeg razumevanja, ovde je detaljan opis rada algoritma propagacije unazad:

Algoritam 2 Algoritam propagacije unazad

1: $d = \nabla_{hL} E$
2: $d = d \odot g'(a_k)$
3: $\nabla_{w_{k0}} E(w) = d + \lambda \nabla_{w_{k0}} \Omega(w)$
4: $\nabla_{W_k} E(w) = dh_{k-1}^T + \lambda \nabla_{W_k} \Omega(w)$
5: $d = W_k^T d$
6: $k = k - 1$
7: Ako je k=0 STOP
8: Vratiti se na korak 2.



Slika 14: Ilustracija algoritma propagacije unazad [28]

Kao što je prikazano na slici 14, propagacija unazad je izuzetno koristan algoritam, ali nije jedini način za optimizaciju parametara mreže. Iako je najčešće korišćen, njegovo pronalaženje bilo je jedan od preduslova za ubrzan razvoj mašinskog učenja. Algoritam se pokazao korisnim za rešavanje praktičnih problema. Iako jednostavan, propagacija unazad nije lako primenljiva na duboke neuronske mreže zbog velikog broja uzastopnih množenja, što može dovesti do problema "nestajućih i eksplodirajućih gradijenata". Ne postoji garancija da će algoritam propagacije unazad pronaći globalni minimum funkcije greške, već samo lokalni minimum. Iako je ovaj problem, koji proizlazi iz nekonveksnosti funkcije greške u neuronskim mrežama, dugo bio smatran velikim nedostatkom, pokazalo se da u mnogim praktičnim problemima to nije ozbiljna prepreka. Propagacija unazad je računski zahtevna, što je dugo smanjivalo interes za ovaj algoritam. Međutim, interes se ponovo javio sa pojavom računara sa jačim grafičkim procesorskim jedinicama (GPU).

8 Optimizacija funkcije troškova

Neuronske mreže su razvijene za potrebe računarstva, ali one su izrežene kroz matematičke formule. Kada govorimo o optimizaciji funkcije troškova, najčešće govorimo o gradijentnim metodama. Ipak, potrebno je pozabaviti se i adaptivnim metodama. Prilikom aproksimacije funkcije, uvek težimo da budemo bliži pravom rešenju, kako bi smo dostigli što veću tačnost u neuronskim mrežama, koristimo funkcije troškova. Sledeće funkcije u praksi se najčešće koriste[6]:

- $L(x) = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$ - **srednjekvadratna greška**
- $L(x) = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}$ - **srednja apsolutna razlika**
- $L(x) = \frac{\sum_{i=1}^{n}||J(x,w_i) - y||^2}{n}$ - **kvadratni gubitak**
- $L(x) = \sum_{i=1}^{n}\max(0.1 - y_i\hat{y}_i)$ - **prelomni gubitak**
- $L(x) = -(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i))$ - **međuetropijska funkcija gubitka**
- $L(x) = \begin{cases} \frac{1}{2}(\hat{y}_i - y_i)^2, & |\hat{y}_i - y_i| \leq \delta \\ \delta|\hat{y}_i - y_i| - \frac{1}{2}\delta^2, & |\hat{y}_i - y_i| > \delta \end{cases}$ - **Huberova funkcija gubitka**

8.1 Gradijentni spust sa momentumom

Prilikom optimizacije funkcija troškova koristimo već spomenute gradijentne metode. Međutim javila se potreba za unapređenjem klasične metode gradijentnog spusta, sa momentumom kako bi se svakom iteracijom funkcija više zagladila. Ideja je da se pamćenjem prethodne iteracije, svaka sledeća iteracija predstavi kao kombinacija prethodnog i sadašnjeg gradijenta.

$$x_{k+1} = x_k - c_k \nabla f_{x_k} + \alpha \Delta x_k,$$

Gde je, $\Delta x_k = x_k - x_{k-1}$ i $\alpha \in [0, 1]$.

Ovim metodom mi ne samo da biramo korak c već biramo i koeficijent momentuma α.[23][31][25]

8.2 Stohastički gradijenti spust

Stohastički gradijenti spust predstavlja značajno unapređenje gradijentne metode spusta. Ideja kod stohastičkog gradijentog spust je da umesto gradijenta, koristi vektor čije očekivanje je kolinearno sa gradijentom i istog su smera.

[6]Za više detalja, pogledajte [11], [8], [9], [10], [16], [6], [22], [26], [4], [12], [23], [25], [31].

Na ovaj način se ubrzava potraga za minimumom funkcije. Stoga Stohastički gradijentni spust možemo predstaviti kao :

$$x_{k+1} = x_k - \alpha_k \nabla f_i(x_k),$$

gde se i bira u skaldu sa uniformnom raspodelom, a funkcija L je prosek funkcija L_i, dok je očekivanje $\nabla f_i(x)$ ista kao i $f(x)$. Prednost stohastičkog spusta u odnosu na gradijenti je to što mu njegova fluktacija omugućuje da pređe na potencijalni bolji lokalni minimum, dok sa druge strane to komplikuje stvari pri konvergenciji jer je moguće da stohstički spust preskoći minimum funkcije.[23][31][25]

8.3 Nesterovljev ubrazani gradijent

Dok momentum zavisi od oscilacija gradijentnog spusta, Nesterovljev metod dopušta da se tačka spušta niz padinu napred, računajući gradijent i vodi računa o budućoj poziciji. U suštini, umesto računanja gradijenta za x_k, mi koristimo $x_k + \gamma \Delta x_k$(gde je $\Delta x = x_k - x_{k-1}$), sa kojim smo bliži mestu gde bi bili sledećim korakom. Te imamo sledeće :

$$x_{k+1} = x_k + \alpha \Delta x_k - c_k \nabla f(x_k + \gamma \Delta x_k).$$

Takođe možemo kombinovati momentum sa Nesterovljevim ubrzanim gradijentom, stavljajući da je $\alpha = \gamma$, što nam daje $x_{k+1} = y_k - c_k \nabla f(y_k)$ i $y_{k+1} = x_{k+1} + \alpha(x_{k+1} - x_k)$. Ovde dobijamo tri parametra c, α, γ koja treba izabrati, umesto dva kod Momentuma.[23][31][25]

8.4 Adaptivni gradijenti spust

Adaptivni gradijentni spust je metoda koja u suštini koristi gradijent prethodnog spusta da usmeri potragu i korak da nas dovede do brže konvergencije. Postoje dve vrste metoda kod Adaptivnog gradijentnog spusta, to je Adaptivni gradijent (**Adagrad**) i Adaptivni moment procene (**Adam**). Naš cilj je pronalazak x, koji minimizuje funkciju troškova.

Gradijentni metod najbržeg spusta ima formu $x_{k+1} = x_k - c_k G_k$, gde su G_k gradijent $k - tog$ koraka.

U slučaju **Adagrad**, imamo da je

$G_{k+1} = \nabla L(x_k)$ i $c_{k+1} = c_k \Big(\sum_{i=1}^{k} ||\nabla L_i(x)||^2 \Big)^{\frac{1}{2}}$, koju ako ubacimo u prethodnu jednakost dobijamo

$$x_{k+1} = x_k + c_k \Big(\sum_{i=1}^{k} ||\nabla L_i(x)||^2 \Big)^{\frac{1}{2}} \nabla L(x_k).$$

Kao što vidimo, kvadratni koren zbira kvadrata funkcije troškova unapređuje veličini koraka za svaku iteraciju, što eliminiše potrebu da to radimo sami. AdaGrad je algoritam koji dobro funkcioniše za retke gradijente. [16]

Adam takođe pamti prethodne gradijente, ali za razliku od **Adagrad**, on čuva ekponencijalno kretanje proseka skvadratnih gradijenata i gradijenta. To možemo zapisati na sledeći način :

$G_{k+1} = \beta G_k + (1-\beta)\nabla L(x_k)$ i $c_{k+1} = c_k\Big((1-\gamma)\sum_{i=1}^{k}\gamma^{k-i}||\nabla L_i(x)||^2\Big)^{\frac{1}{2}}$

Adam zbog svoje osobine da kooridnantno izvodi veličine G_k i c_k otvara mu se mogučnost bržeg rada, kao i to da memorijski manje košta. Takođe, Adam je vrlo lagan za podešavanje, pri inicijalizaciji. Mada njegova mana je da ima tendenciju ka bržem konvergiranju ili da uopšte ne može da konvergira u zavisnosti od stope učenja. Generalno, Adam algoritam se pokazao kao robustan i pogodan za širok spektar nekonveksnih problema optimizacije u oblasti mašinskog učenja. [16]

9 Primena metoda optimizacije funkcije troškova

Upotreba optimizacije u neuronskim mrežama kako bi se pronašle težine, svodi se na minimizaciju funkcije troškova. Minimizacijom funkcije troškova, neuronska mreža je sposobna da daje tačnija predviđanja i klasifikacije. Optimizacija je važan postupak u procesu treniranja, on određuje konačan skup težina koji se koriste za predviđanja. Bez optimizacije, mreža ne bi bila u mogućnosti da uči i poboljšava svoje performanse. Poboljšavanjem tačnosti predviđanja, optimizacija takođe može pomoći prilikom preprilagođavanja, gde modeli postaju previše komplesni a performanse modela previše spore za nove podatke. Minimizacijiom funkcije troškova, model je sposoban da bolje generalzuje nove podatke, koji dovode do poboljšanja u stvarnim problemima.

Za potrebe ovog master rada posmatrali smo ponašanje funkcija troškova prilikom optimizacije, primenom u praksi najčešće korišćenih metoda optimizacije. Ova sekcija je podeljena na dva modela, prvi model se odnosi na klasifikaciju a drugi se odnosi na regresiju. Model neuronskih mreža je sastavljen od tri sloja (ulaznog,skrivenog i izlaznog). Kod klasifikacionog modela koristimo skup podataka ***MNIST*** koji sadrži 60000 sličica ručno napisanih brojeva od **0 - 9**, sa pratećim opisom koji broj označava ta slika. Za potrebe regresionog modela korišćen je skup podataka ***Diabetes***, gde je potrebno formirati regresioni model koji predviđa koliko je diabetes napredovao. Oba skupa predstavljaju osnovne skupove za testiranja i sadržana su u bibliotekama koje se koriste u *Pythonu*.

Klasifikacioni model

Klasifikacioni model koristi dve funkcije troškova *Međuentropijsku funkciju troškova* i *funkciju prelomonog gubitka*. Kako bi mogli videti delovanje optimizatora na ove dve funkcije troškova, parametri su za svaki od četiri optimizatora isti, njih smo definisali na sledeći način[7][17] :

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.001)
momentum = tf.keras.optimizers.SGD(learning_rate=0.001,
                                   momentum=0.9)
adagrad = tf.keras.optimizers.Adagrad(learning_rate=0.001)
adam = tf.keras.optimizers.Adam(learning_rate=0.001)
```

Ovaj deo koda[7] prikazuje četiri različita optimizatora iz TensorFlow biblioteke, koji se koriste za treniranje neuronskih mreža. Svaki optimizator ima specifične karakteristike koje utiču na brzinu i efikasnost treniranja modela.[8]

[7]Ovaj deo koda, kao i svaki naredni kod u ovom master radu, je preuzet sa `https://keras.io/about/a` i prilagođen za potrebe testiranja.[7][17]

[8]Prilagođeni kod možete pronaći na GitHubu.[7][17]

`sgd`: Inicijalizuje instancu optimizatora Stohastički gradijentni spust (SGD) sa stopom učenja 0.001. Prema definiciji **Stohastički gradijenti spust**, SGD se može zapisati kao:

$$x_{k+1} = x_k - \alpha_k \nabla f_i(x_k)$$

`momentum`: Inicijalizuje instancu optimizatora Gradijentnog spusta sa stopom učenja 0.001 i momentom 0.9. SGD sa momentom se može zapisati kao:

$$x_{k+1} = x_k - c_k \nabla f_{x_k} + \alpha \Delta x_k$$

`adagrad`: Inicijalizuje instancu optimizatora Adagrad sa stopom učenja 0.001. Adagrad se može zapisati kao:

$$x_{k+1} = x_k + c_k \Big(\sum_{i=1}^{k} ||\nabla L_i(x)||^2 \Big)^{\frac{1}{2}} \nabla L(x_k)$$

`adam`: Inicijalizuje instancu optimizatora Adam sa stopom učenja 0.001. Adam se može zapisati kao:

$$G_{k+1} = \beta G_k + (1 - \beta) \nabla L(x_k)$$

i

$$c_{k+1} = c_k \Big((1 - \gamma) \sum_{i=1}^{k} \gamma^{k-i} ||\nabla L_i(x)||^2 \Big)^{\frac{1}{2}}$$

Za metriku modela koristi se *accuracy*.[7][17]

```
model_adagrad.compile(optimizer=adagrad,
       loss=tf.keras.losses.Hinge(),metrics=['accuracy'])
model_adam.compile(optimizer=adam,
       loss=tf.keras.losses.Hinge(), metrics=['accuracy'])
model_momentum.compile(optimizer=momentum,
       loss=tf.keras.losses.Hinge(),metrics=['accuracy'])
model_sgd.compile(optimizer=sgd,
       loss=tf.keras.losses.Hinge(), metrics=['accuracy'])
```

Kod iznad prikazuje četiri modela neuronske mreže koji koriste različite optimizatore prilikom treniranja. Ovi modeli su kompilirani pomoću *TensorFlow Keras API-ja*. Suština ovog koda je da poredi performanse različitih optimizatora na istom zadatku kroz obučavanje modela.

`model_adagrad`: Model koji koristi Adagrad optimizator.

`model_adam`: Model koji koristi Adam optimizator.

`model_momentum`: Model koji koristi optimizator sa momentom (*Momentum*).

`model_sgd`: Model koji koristi Stohastički gradijentni spust (*SGD*) optimizator. Svaki model se kompilira pozivom metode *compile*(), koja prima sledeće argumente:

`optimizer`: Optimizator koji se koristi za treniranje modela. U ovom slučaju, to su Adagrad, Adam, Momentum i SGD.

`loss`: Funkcija troška koja se koristi za evaluaciju performansi modela tokom treniranja. U prikazanom kodu, koristi se funkcija gubitka "*Hinge*", dok se za poređenje koristi "*Cross entropy*".

`metrics`: Lista metrika koje se koriste za evaluaciju modela. U ovom slučaju, koristi se tačnost (accuracy) kao metrika.

Kroz obučavanje ovih modela sa različitim optimizatorima, možemo uporediti njihove performanse i odabrati optimizator koji daje najbolje rezultate za zadati problem.

Oba modela su trenira u 100 epoha.[7][17]

```
history_adam = model_adam.fit(X_train, y_train,
       batch_size=64,epochs=100,
       verbose= True,validation_data=(X_val, y_val))
history_adagrad= model_adagrad.fit(X_train, y_train,
       batch_size=64,epochs=100,
       verbose= True,validation_data=(X_val, y_val))
history_momentum = model_momentum.fit(X_train, y_train,
       batch_size=64,epochs=100,
       verbose= True,validation_data=(X_val, y_val))
```

```
history_sgd = model_sgd.fit(X_train, y_train,
       batch_size=64,epochs=100,
       verbose= True,validation_data=(X_val, y_val))
```

Kod prikazuje obučavanje četiri modela neuronske mreže koji koriste različite optimizatore. Ovde se koriste *Adagrad, Adam, Momentum i Stohastički gradijentni spust (SGD)*. Obučavanje se vrši pomoću metode $fit()$ i zasniva se na istom skupu podataka za obuku i validaciju.

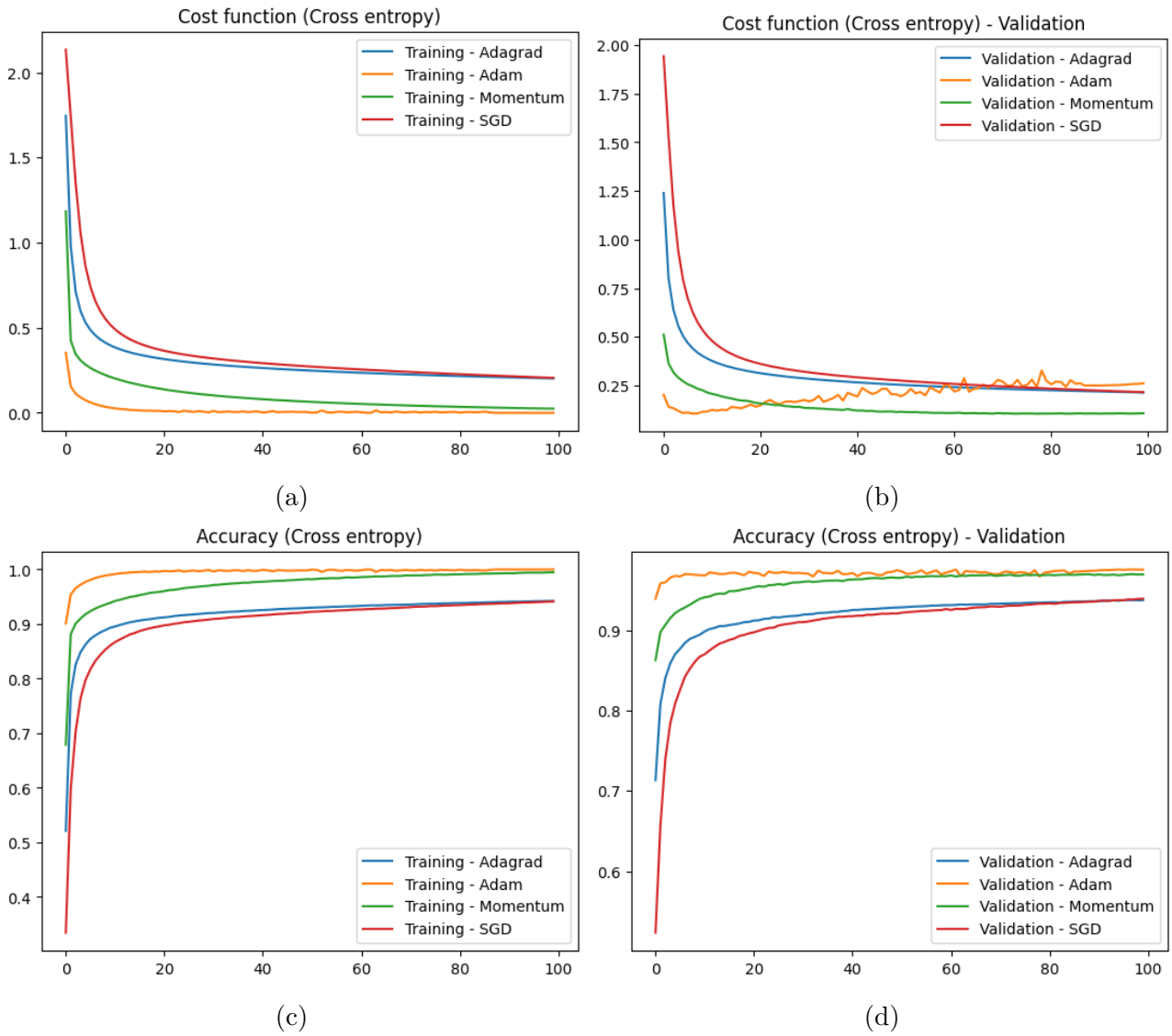
Za svaki model, metoda $fit()$ prima sledeće argumente:

- `X_train, y_train`: skup podataka za obuku, gde `X_train` predstavlja ulazne podatke, a `y_train` odgovarajuće oznake.
- `batch_size`: veličina paketa (batch) za obučavanje, u ovom slučaju 64.
- `epochs`: broj epoha (prolaza kroz ceo skup podataka) za obučavanje, u ovom slučaju 100.
- `verbose`: nivo ispisivanja informacija tokom obučavanja. `True` znači da će biti ispisane informacije o napretku obučavanja.
- `validation_data`: skup podataka za validaciju, koji se koristi za praćenje performansi modela tokom obučavanja. U ovom slučaju, to su `X_val` i `y_val`.

Rezultati treniranja, uključujući metrike performansi tokom svake epohe, čuvaju se u promenljivim `history_adam`, `history_adagrad`, `history_momentum` i `history_sgd` za svaki odgovarajući model. Ovi rezultati se mogu koristiti za analizu i poređenje performansi različitih optimizatora.[9]

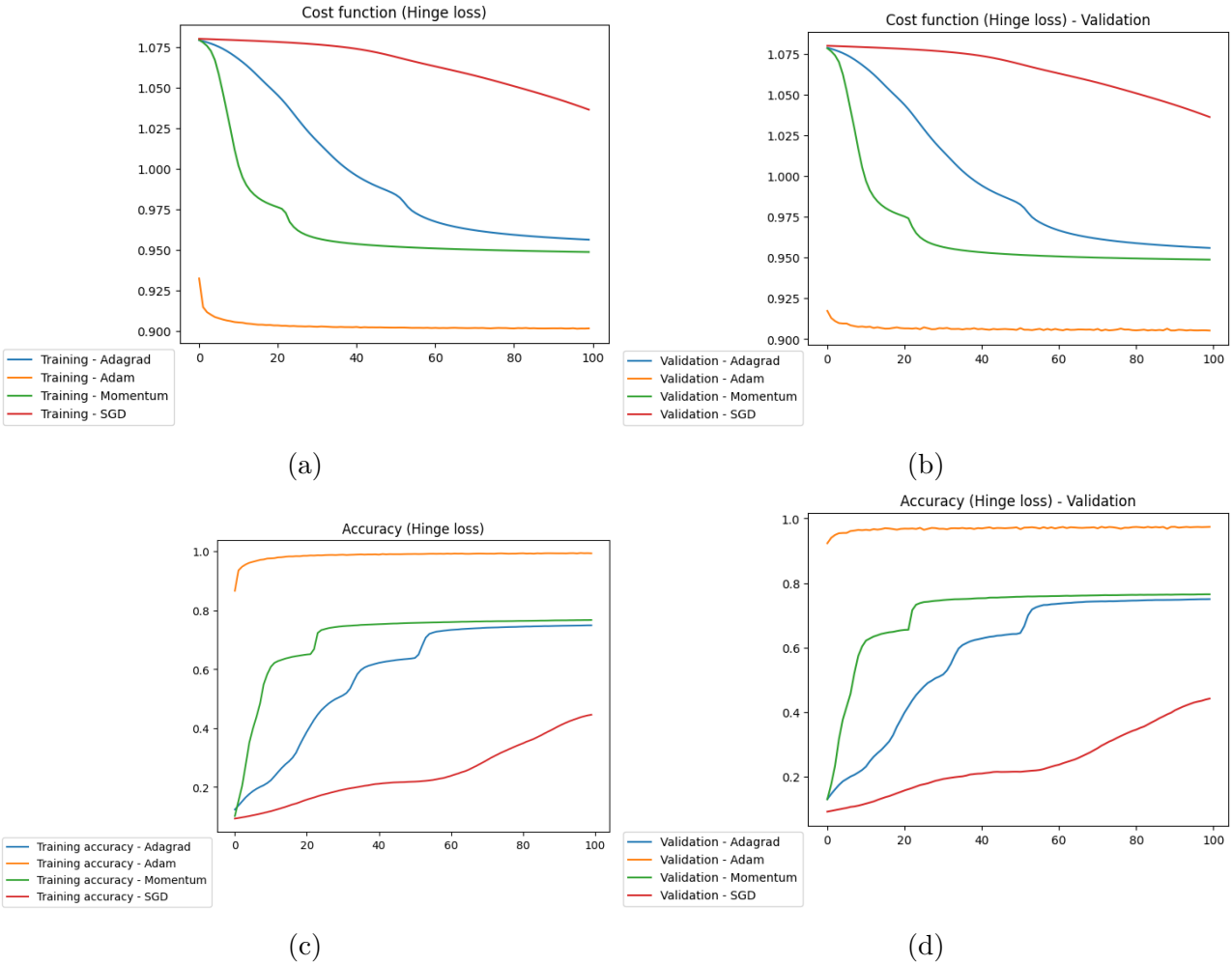
Grafikone kretanja funkcije troškova prilikom različitih opitmizacija, prikazaćemo uporedno na trening i validacionom skupu.

[9]Prilagođeni kod i rezultate možete pronaći na GitHubu.[7][17]



Slika 15: Rezultati sa Međuentropijskom funkcijom gubitka kao funkcijom troškova[10]

[10]Grafici su kreirani koristeći Python biblioteku `matplotlib`.



Slika 16: Rezultati sa Prelomnim gubitkom kao funkcijom troškova

[11]

Rezultati modela koji su koristili međuentropijsku funkciju gubitka kao funkciju troška ilustrovani su na slikama 15a, 15b, 15c i 15d koje su deo eksperimenta 15. Analizom ovih rezultata, dolazimo do zaključka da je Adam optimizator dostigao impresivnu tačnost od 97,84% na validacionom skupu podataka.

Slično tome, modeli koji su koristili prelomni gubitak kao funkciju troška prikazani su na slikama 16a, 16b, 16c i 16d koje čine eksperiment 16. U ovom slučaju, Adam optimizator je i dalje bio superiorniji, dostižući vrhunsku tačnost od 97,58% na validacionom skupu podataka.

U eksperimentu sa SGD optimizatorom, model je postigao tačnost na trening skupu od 94,33% i tačnost na validacionom skupu od 94,03% koristeći međuentropijsku funkciju gubitka kao funkciju troška, što se može videti na slikama 15c i 15d. Kada je korišćena funkcija prelomnog gubitaka, tačnost treninga je bila 53,85%, a tačnost validacije 53,90%, što je ilustrovano na slikama 16c i 16d. Iako su ovi rezultati pokazali određeni nivo efikasnosti, druge metode optimizacije su pokazale bolje performanse u ovom eksperimentu.

[11] Grafici su kreirani koristeći Python biblioteku `matplotlib`.

Kada je primenjen Momentum optimizator, model je postigao tačnost treninga od 99,44% i tačnost validacije od 97,12% sa međuentropijskom funkcijom gubitka, što je prikazano na slikama 15c i 15d. Sa funkcijom prelomnog gubitka, tačnost treninga je iznosila 67,31%, a tačnost validacije 67,08%, što je ilustrovano na slikama 16c i 16d.

Koristeći Adagrad optimizator, model je postigao tačnost treninga od 94,17% i tačnost validacije od 93,78% koristeći međuentropijskom funkcijom gubitka, što je prikazano na slikama 15c i 15d. Kada je korišćen prelomni gubitak, tačnost treninga je iznosila 57,77%, a tačnost validacije 57,91%, što je ilustrovano na slikama 16c i 16d. Adagrad optimizator je pokazao poboljšanje u odnosu na SGD u određenim situacijama, ali u ovom eksperimentu nije uspeo da nadmaši Adam ili Momentum optimizator.

Iz ovih podataka, može se izvesti zaključak da je Adam optimizator najpogodniji izbor za treniranje ovakvog modela dubokog učenja na MNIST skupu podataka, bez obzira na izabrane funkcije gubitka. Međutim, važno je napomenuti da brza konvergencija koju Adam optimizator omogućava može dovesti do preprilagođavanja, što bi moglo negativno uticati na opštu primenljivost modela. Stoga, izbor optimizacionog algoritma treba pažljivo odabrati, uzimajući u obzir specifične zahteve i ograničenja datog projekta.

Regresioni model

Model neuronskih mreža koji smo koristili za regresiju koristio je takođe dve funkcije troškova *Srednjekvadratnu grešku* i *Huberovu funkciju gubitka*. Kao i kod modela za klasifikaciju, ostavili smo iste optimizatore za model regresije gde su parametri jednaki za sve optimizatore.

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.01,clipnorm=0.1)
momentum = tf.keras.optimizers.SGD(learning_rate=0.01,
               momentum=0.9,clipnorm=0.1)
adagrad = tf.keras.optimizers.Adagrad(learning_rate=0.01,
               clipnorm=0.1)
adam = tf.keras.optimizers.Adam(learning_rate=0.01,clipnorm=0.1)
```

Prilikom evaluacije modela korišćena je metrika *apsolutne razlike.*[12]

- `sgd = tf.keras.optimizers.SGD(learning_rate=0.01,clipnorm=0.1)`

  Ova linija koda inicijalizuje Stohastički gradijentni spust (SGD) optimizator. Parametar `learning_rate` određuje veličinu koraka kojim se parametri modela ažuriraju tokom svake iteracije treniranja. `clipnorm` je parametar koji ograničava normu gradijenata. Ograničavanje norme gradijenata na

[12]Prilagođeni kod i rezultate možete pronaći na GitHubu.[7][17]

određenu vrednost može pomoći u stabilizaciji procesa treniranja i sprečiti eksploziju gradijenata.

- `momentum = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9,clipnorm=0.1)`

  Ovde se takođe inicijalizuje Momentum optimizator, ali sa dodatnim parametrom `momentum`. Momentum pomaže u ubrzanju Gradijentnog spusta u relevantnom pravcu i smanjuje oscilacije, što može dovesti do bržeg i stabilnijeg treniranja. Vrednost 0.9 je često korišćena vrednost za momentum.

- `adagrad = tf.keras.optimizers.Adagrad(learning_rate=0.01, clipnorm=0.1)`

  AdaGrad optimizator je inicijalizovan ovom linijom koda. AdaGrad prilagođava stope učenja za svaki parametar modela pojedinačno, dajući veće ažuriranje za retko ažurirane parametre.

- `adam = tf.keras.optimizers.Adam(learning_rate=0.01, clipnorm=0.1)`

  Konačno, Adam optimizator se inicijalizuje ovde. Adam kombinuje prednosti AdaGrad i RMSProp optimizatora, i često se koristi u praksi jer je robusni optimizator za različite vrste problema.

```
model_adam.compile(optimizer=adam, loss='mse',
            metrics=['mae'])
model_adagrad.compile(optimizer=adagrad, loss='mse',
            metrics=['mae'])
model_sgd.compile(optimizer=sgd, loss='mse',
            metrics=['mae'])
model_momentum.compile(optimizer=momentum, loss='mse',
            metrics=['mae'])
```

- `model_adam.compile(optimizer=adam, loss='mse', metrics=['mae'])`

  Ova linija koda kompilira model koji koristi Adam optimizator (definisan prethodno). Za funkciju gubitka koristi se kvadratna greška (mean squared error - MSE), koja je često korišćena funkcija gubitka za probleme regresije. Za metriku evaluacije koristi se apsolutna greška (mean absolute error - MAE), koja daje prosečnu apsolutnu vrednost razlika između predviđenih i stvarnih vrednosti.

- `model_adagrad.compile(optimizer=adagrad, loss='mse', metrics=['mae'])`

Slično prethodnom, ova linija koda kompilira model koji koristi AdaGrad optimizator. Takođe koristi MSE za funkciju gubitka i MAE za metriku evaluacije.

- `model_sgd.compile(optimizer=sgd, loss='mse', metrics=['mae'])`

  Ovde se kompilira model sa SGD optimizatorom, koristeći MSE za funkciju gubitka i MAE za metriku evaluacije.

- `model_momentum.compile(optimizer=momentum, loss='mse', metrics=['mae'])`

  Na kraju, ova linija koda kompilira model koji koristi Gradjintim spustom kao optimizator sa momentumom. Takođe koristi MSE za funkciju gubitka i MAE za metriku evaluacije.

Takođe i ovaj model je treniran u 100 epoha sa podeljenim validacionim skupom.[13]

```
history_adam = model_adam.fit(X_train, y_train,
        epochs=100, batch_size=32,
        validation_data=(X_test, y_test))
history_adagrad = model_adagrad.fit(X_train, y_train,
       epochs=100, batch_size=32,
       validation_data=(X_test, y_test))
history_sgd = model_sgd.fit(X_train, y_train,
       epochs=100, batch_size=32,
       validation_data=(X_test, y_test))
history_momentum = model_momentum.fit(X_train, y_train,
       epochs=100, batch_size=32,
       validation_data=(X_test, y_test))
```

- `history_adam = model_adam.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))`

  Ova linija koda trenira model koristeći Adam optimizator. Treniranje se vrši u 100 epoha, gde svaka epoha predstavlja jedan prolazak kroz celokupan skup podataka za treniranje. Parametar `batch_size` određuje broj uzoraka koji se propuštaju kroz model pre nego što se ažuriraju parametri modela. `validation_data` parametar ukazuje na podatke koji će se koristiti za validaciju modela nakon svake epohe.

- `history_adagrad = model_adagrad.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))`

[13] Prilagođeni kod i rezultate možete pronaći na GitHubu.[7][17]

Ova linija koda trenira model koristeći AdaGrad optimizator, sa istim parametrima za broj epoha, veličinu serije i podatke za validaciju kao i prethodni model.

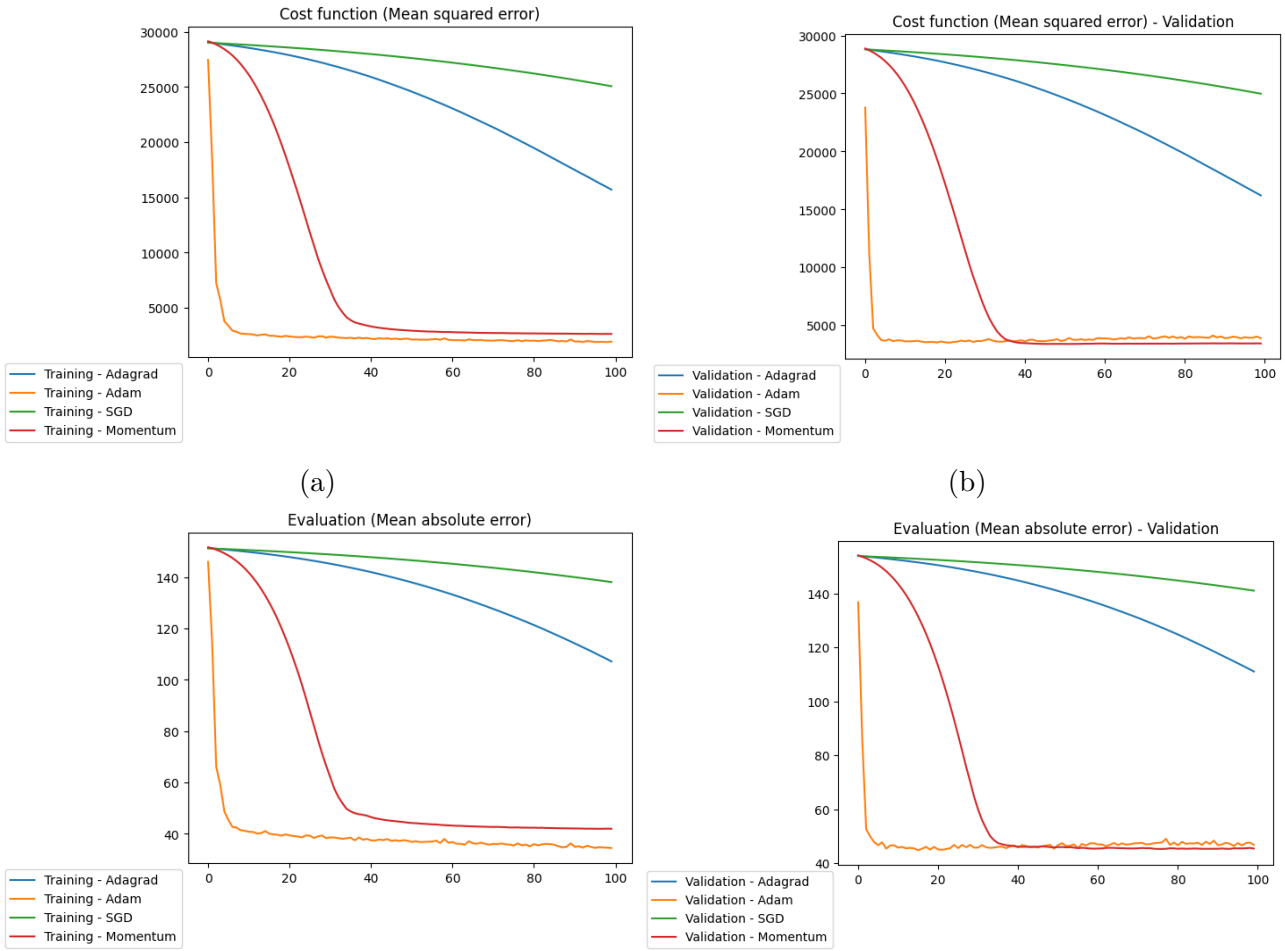
- `history_sgd = model_sgd.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))`

  Ovde se trenira model koristeći SGD optimizator, sa istim parametrima za broj epoha, veličinu serije i podatke za validaciju kao i prethodni modeli.

- `history_momentum = model_momentum.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))`

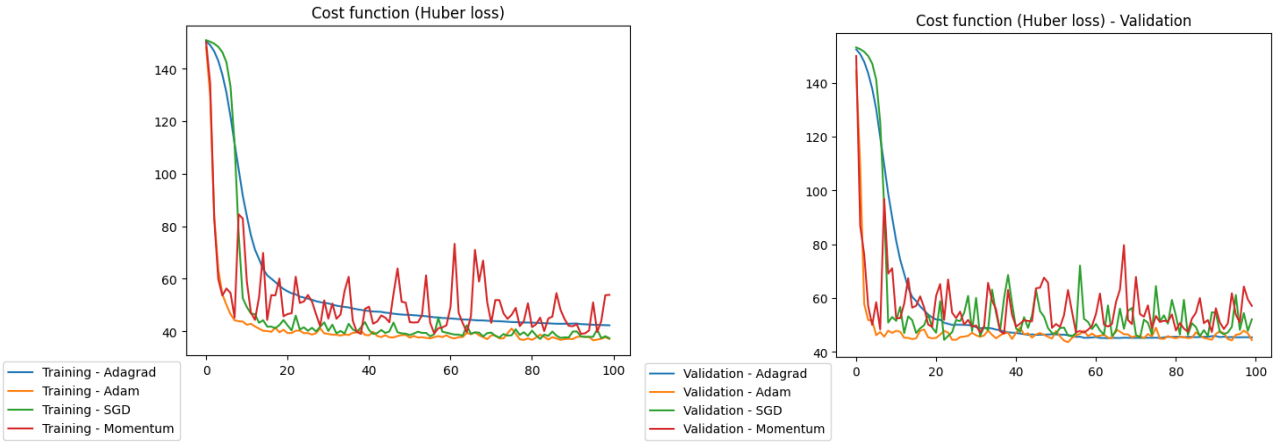
  Na kraju, ova linija koda trenira model koristeći SGD optimizator sa momentumom, sa istim parametrima za broj epoha, veličinu serije i podatke za validaciju kao i prethodni modeli.

Kako bi bolje sagledali brzinu konvergencije i evaluaciju modela prilikom svake optmizacije, analiziraćemo sledeće grafike i njih uporediti sa prethodnim graficima kod modela za klasifikaciju.



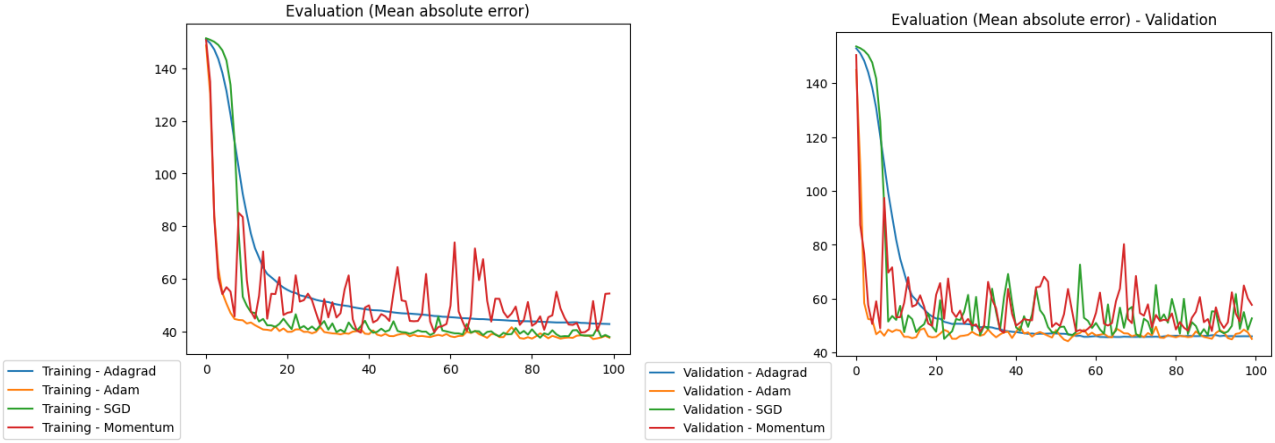
Slika 17: Rezultati sa Srednjekvadratno greškom kao funkcijom troškova

14



(a)

(b)



(c)

(d)

Slika 18: Rezultati sa Hubertovom funkcijom gubitka kao funkcijom troškova [15]

Rezultati modela koji su koristili srednjekvadratni gubitak kao funkciju troška ilustrovani su na slikama 17a, 17b, 17c i 17d koje su deo eksperimenta 17. Analizom ovih rezultata, dolazimo do zaključka da je Momentum optimizator postigao najniži test gubitak od 3447.8149. Adam optimizator je postigao test gubitak koji je bio 78,65% niži u poređenju sa Adagradom, i 84,77% niži u poređenju sa SGD, ali je imao 11,20% viši test gubitak u poređenju sa Momentumom. Najmanji gubitak tokom treninga od 2008.1139, Adam optimizator je postigao na 93. epohi.

S druge strane, modeli koji su koristili Huberov gubitak kao funkciju troška prikazani su na slikama 18a, 18b, 18c i 18d koje čine eksperiment 18. U ovom slučaju, Adam optimizator je imao najniži test gubitak od 45.2312, što je bilo 1,71% niže u poređenju sa Adagradom, 5,42% niže u poređenju sa SGD, i 4,60% niže u poređenju sa Momentumom. Adam optimizator je postigao najmanji gubitak tokom treninga od 36.4400 na 100. epohi.

Iz ovih podataka, može se izvesti zaključak da Adam optimizator pokazuje

[14] Grafici su kreirani koristeći Python biblioteku `matplotlib`.
[15] Grafici su kreirani koristeći Python biblioteku `matplotlib`.

najbolje performanse kada se koristi Huberov gubitak kao funkcija gubitka, dok Momentum optimizator ima najbolje performanse kada se koristi srednjekvadratni gubitak. Bez obzira na to, izbor optimizacionog algoritma treba pažljivo odabrati, uzimajući u obzir specifične zahteve i ograničenja datog projekta.

Ova analiza otvara put za dalja istraživanja koja bi mogla da se fokusiraju na kombinaciju različitih optimizacionih algoritama ili na prilagođavanje parametara kako bi se postigla bolja optimizacija. Takođe, istraživanje na različitim skupovima podataka bi moglo da pruži dodatne uvide o tome kako se ovi optimizatori ponašaju u različitim kontekstima.

Posmatrajuči grafike kod modela za regresiju primečuje se da *Adam* optimizacijom dobijamo dosta na brzini pri konvergenciji funkcije troškova ka minimumu. Upoređujuči grafike sa oba modela, vidimo da evaluacija modela zavisi od što bolje optimizacije funkcije troškova, ali ipak brzina konvergencije ne znači nužno da ćemo dobiti bolji model i tačniji model. Optimizacione metode *Adam* i *Adagrad* zbog svoje brzine mogu se primeniti kao prvi izbor pri kreiranju modela, ali ukoliko se dobro ne podese parametri model se može preprilagoditi podacima, što će nas na kraju koštati tačnosti modela. Optimizacione metode *Stohastički gradijenti spust* i *Gradijentni spust sa momentumom* zbog svoje jednostavnosti pri implementaciji, primenljivi su kod velikih skupova i kod skupova koji se mogu predstaviti kao retki (*sparse*) podaci - podaci čiji vektori sadrže veliki broj nula ili su nedostajučih vrednosti. Ali takođe imaju mogučnost da nekad sporo konvergiraju, ukoliko nije dobro određena stopa učenja.

Modeli koji su korišćeni za ovaj master rad dati na sajtu :
`https://github.com/kaaiiser/Master_rad_Cost_function` [17]

10 Zaključak

Optimizacija funkcije troškova neuronskih mreža je neizostavan deo treniranja modela neuronskih mreža. Funkcija troškova nije uvek konveksna funkcija, što znači da ima višestrukih lokalnih minimuma te je moguće da optimizacioni algoritam zapne u okolini tog lokalnog minimuma. Cilj treniranja neuronskih mreža je pronalazk skupa težina koji će minimizariti funkciju troškova, tako da neuronska mreža može dati što tačnije podatke.

Tačka lokalnog minimuma ima osobinu da daje najmanju vrednost funkciji troškova od ostalih tački u okolini, ali ona ipak nije tačka globalnog minimuma. Problem zapinjanja u tačku lokalnog minimuma funkcije troškova, dovodi do toga da optimizacioni algoritam ne može pronaći skup sa boljim težinama koji će unaprediti model.

Optimizacioni algoritmi kao što su *Adam*,*Adagrad*, *Stohastički gradijentni spust* , *Gradijentni spust sa momentumom* mogu pomoći modelu da izađe iz okoline lokalnog minimuma i počne konvergirati ka globalnom minimumu tako što će dodati momentum pri nekom koraku, prilagoditi korak učenja potrebama optimizacije, dodati težinu podacima koji se ne pojavljuju često ali su važni. Ovi algoritmi takođe omogućavaju da funkcija troškova brže konvergira, kao i to da oscilira ili nekad divergira.

Izbor optimizacionog algoritma za optimizaciju funkcije troškova predstavlja značajan korak prilikom pravljenja modela neuronskih mreža. Pogrešan izbor algoritma može nam oduzeti dosta dragocenog vremena prilikom treniranja mreže, te je potrebno poznavati dobre i loše strane algoritama koje primenjujemo prilikom optimizacije.

Kao što se pokazalo u primerima u ovom radu *Adam* optimizator se predstavlja kao dobar izbor pri kreiranju mreže. On je u praksi vrlo popularan izbor prilikom treniranja modela zbog svoje računske efikasnosti, korišćenja minimalno memorije,pogodan je za treniranje velikih skupova podataka. Međutim *Adam* može biti osetljiv na stopu učenja gde zahteva obzirno podešavanje, takođe se loše ponaša kada je u pitanju visoka nekonveksna optimizacija.

Stohastički gradijenti spust je jednostavan algoritam za implementaciju. U praksi je predstavlja jedan od najkorišćenijih algoritama za optimizaciju funkcije troškova zbog njegove osobine da se može koristiti prilikom treniranja različitih vrsta modela. Kao što se na graficima može viditeti, SGD je dosta osetljiv prilikom podešavanja parametara i mora se posebno voditi računa prilikom njihovog podešavanja. Takođe on je jednostavan algoritam, te ne može konvergirati brže kao neki napradniji algoritmi.

Optimizacija funkcije troškova pomoću *Adagrad* algoritma zahteva veću stopu učenja na početku treniranja kako bi model napravio progres i manju na kraju, stoga zahteva posebno posvečivanje pažnje parametrima. Njegova dobra

strana se pokazala u praksi pri *NLP* problema.

Gradijenti spust sa momentumom je pokazao da može promeniti pravac prilikom optimizacije i dati na brzini konvergencije jednom prostom optimizacionom algoritmu *Gradijentnog spusta*. Momentum takođe može izaći iz okoline sedlaste tačke i nastaviti konvergenciju, prednost je i ta što može zagladiti funkciju. Gradijenti spust sa momentumom može biti osetljiv na podešavanje momentuma, na visokim ne konveksnim problemima njegove performanse se ne pokazuju kao najbolji izbor. U nekim slučajevima, ne može značajno unaprediti optimizaciju.

Pokazuje se da je vrlo važna stvar izbor parametara prilikom optimizacije. Parametri mogu uključivati izbor stope učenja, momentuma, broj epoha itd. Dobar podešavanje parametara može pomoći da se osigura da proces optimizacije funkcije troškova konvergira do dobrog rešenja u nekom razumnom vremenu. Optimizacija je krucijalni deo treniranja neuronskih mreža, te pažljivi izbor optimizacionig algoritma i podešavanja parametara mogu imati velikog efekta na sposobnosti neuronskih mreža u izvršavanju zadatka.

Međutim, važno je naglasiti da izbor optimalnog optimizacionog algoritma i funkcije gubitka zavisi od mnogih faktora, uključujući specifičnosti samog problema, kompleksnost modela, kao i karakteristike i veličinu dostupnih skupova podataka.

Ova analiza pruža korisne smernice za odabir optimizacionih algoritama u kontekstu modela dubokog učenja. Takođe, otvara put za dalja istraživanja koja bi se mogla fokusirati na ispitivanje različitih kombinacija optimizacionih algoritama, prilagođavanje parametara za optimizaciju ili ispitivanje performansi na različitim skupovima podataka.

Kao sveobuhvatan zaključak, rezultati ovog eksperimenta naglašavaju složenost i višedimenzionalnost problema optimizacije u dubokom učenju. Očigledno je da ne postoji univerzalni optimizator ili funkcija troška koja će biti najefikasnija u svim situacijama. Stoga, izbor optimizacionog algoritma i funkcije troška zahteva pažljivo razmatranje i prilagođavanje specifičnostima određenog problema.

Literatura

[1] Desmos — graphing calculator. https://www.desmos.com/calculator/, 2023.

[2] Swapna Agarwalla and Kandarpa Kumar Sarma. Machine learning based sample extraction for automatic speech recognition using dialectal assamese speech. 2015.

[3] Charu C. Aggarwal. *Linear Algebra and Optimization for Machine Learning.* Springer Cham, 2020.

[4] Matt Algore. *Machine Learning with Python.* 2021.

[5] Paris Smaragdis Brian King, Cedric Fevotte. Optimal cost function and magnitude power for nmf-based speech separation and music interpolation. 2012.

[6] Shuxiang Cao, Leonard Wossnig, Brian Vlastakis, Peter Leek, and Edward Grant. Cost-function embedding and dataset encoding for machine learning with parametrized quantum circuits. 2020.

[7] François Chollet et al. Keras. https://keras.io, 2015.

[8] codebasic. Machine learning tutorial python - 4: Gradient descent and cost function. https://youtu.be/vsWrXfO3wWw, 2018. YouTube.

[9] codebasic. Gradient descent for neural network — deep learning tutorial 12 (tensorflow2.0, keras & python). https://youtu.be/pXGBHV3y8rs, 2020. YouTube.

[10] codebasic. Loss or cost function — deep learning tutorial 11 (tensorflow tutorial, keras & python). https://youtu.be/E1yyaLRUnLo, 2020. YouTube.

[11] codebasic. Stochastic gradient descent vs batch gradient descent vs mini batch gradient descent —dl tutorial 14. https://youtu.be/IU5fuoYBTAM, 2020. YouTube.

[12] Jay Dawani. *Hands-On Mathematics for Deep Learning.* Pact Publishing, 2019.

[13] Katarina Džepina. *Gradijentne metode i metode konjugovanih gradijenata.* 2020.

[14] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Beating the perils of non-convexity: Guaranteed training of neural network using tensor methods. 2016.

[15] Yong Liu Wei-Hua Xu Jun-Gang Lou Jong-Hyok Ri, Guanzhong Tian. Extreme learning machine with hybrid cost function of g-mean and probability for imbalanced learning. 2020.

[16] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stohastic optimization. 2015.

[17] Adil Kolaković. Github. `https://github.com/kaaiiser/Master_rad_Cost_function`, 2023.

[18] Solomon Kullback and Richard A Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[19] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[20] Olvi L. Mangasarian. *Nonlinear Programming*. Society for Industrial and Applied Mathematics, 10th edition, 1994.

[21] Aditi Mittal. Understanding rnn and lstm. `https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e`, 2019.

[22] Aurora Y. Mu. A hybrid machine learning model with cost-function based outlier removal an its application on credit rating. 2020.

[23] Mladen Nikolić and Anđelka Zečević. *Mašinsko učenje*. 2019.

[24] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research, 9th edition, 2006.

[25] Sebastian Raschka and Vahid Mirjalili. *Python mašinsko učenje*. 2020.

[26] Shagan Sah. Machine learning: A review of learning types. 2020.

[27] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. `https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/`, 2018.

[28] Durvesh Shah. Backpropagation neural networks- classification and regression from scratch with python. `https://medium.com/mlearning-ai/backpropagation-neural-networks-classification-and-regression-7858685` 2022.

[29] SM Sinha. *Mathematical Programming: Theory and Methods*. ELSEVIER, 1th edition, 2006.

[30] Zorica Stanimirović. *Nelinearno programiranje*. 2014.

[31] Igor Ševo. *Specijalizovana neuronska mreža za klasifikaciju i segmentaciju aero-snimaka*. 2020.