

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4580

Konvolucijske neuronske mreže

Damir Kopljar

Zagreb, srpanj 2016.

Zagreb, 17. ožujka 2016.

ZAVRŠNI ZADATAK br. 4580

Pristupnik: **Damir Kopljar (0036480158)**
Studij: **Računarstvo**
Modul: **Računarska znanost**

Zadatak: **Konvolucijske neuronske mreže**

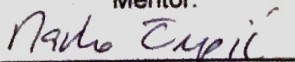
Opis zadatka:

Umjetne neuronske mreže predstavljaju robustan model koji omogućava rješavanje regresijskih i klasifikacijskih zadataka pri čemu se parametri mogu učiti iz dostupnih podataka. Postoji niz arhitektura umjetnih neuronskih mreža. Jedna od mogućih arhitektura su konvolucijske umjetne neuronske mreže koje se često koriste pri obradi slikovnih podataka.

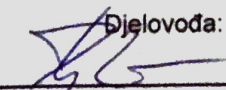
U okviru ovog rada potrebno je proučiti konvolucijske neuronske mreže i njihove podvrste. Potrebno je opisati ovu vrstu neuronskih mreža te napraviti prikladnu programsku implementaciju mreže i algoritma učenja. Na određenom skupu podataka provesti postupak učenja neuronske mreže. Za implementirani postupak potrebno je provesti vrednovanje s obzirom na različite podvrste mreže. Radu je potrebno priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 17. lipnja 2016.


Mentor:


Doc. dr. sc. Marko Čupić

Djelovoda:


Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za
završni rad modula:


Prof. dr. sc. Siniša Srbljić

*Zahvaljujem mentoru doc. dr. sc. Marku Čupiću na pomoći pri izradi ovog rada.
Zahvaljujem i obitelji i prijateljima na podršci kroz cijeli studij.*

SADRŽAJ

1. Uvod	1
2. Umjetne neuronske mreže	2
2.1. Neuron	2
2.2. Aktivacijske funkcije	3
2.3. Struktura neuronske mreže	4
2.4. Primjer rada jednostavne neuronske mreže	5
2.5. Programska izvedba rada neuronske mreže	6
3. Učenje neuronskih mreža	9
3.1. Algoritam propagacije pogreške unatrag	9
3.2. Primjer algoritma učenja	11
4. Konvolucijske neuronske mreže	14
4.1. Konvolucijski sloj	15
4.1.1. Primjer rada konvolucijskog sloja	16
4.1.2. Programska izvedba rada konvolucijskog sloja	17
4.2. Sloj sažimanja	19
4.2.1. Programska izvedba sažimanja maksimalnom vrijednosti	19
5. Učenje konvolucijske neuronske mreže	22
5.1. Konvolucijski sloj	22
5.1.1. Programska izvedba	23
5.2. Sloj sažimanja maksimalnom vrijednosti	24
5.2.1. Programska izvedba	24
5.3. Primjer arhitekture konvolucijske mreže	25
6. Primjer konvolucijske mreže	27
6.1. Baza MNIST	27

6.2. Rezultati	28
6.2.1. Jednostavna konvolucijska mreža	28
6.2.2. Primjer složenije arhitekture	29
6.2.3. Primjer s promjenjivom stopom učenja	30
6.2.4. Primjer s više epoha	31
7. Zaključak	32
Literatura	33

1. Uvod

Neuronske mreže svakim danom postaju sve popularnije, no njihov razvoj počeo je već u pedesetim godinama predstavljanjem perceptrona. Iako je matematički model već tada bio spreman, računala jednostavno nisu bila dovoljno jaka da bi se neuronske mreže mogle učiti na velikim količinama podataka.

Zbog eksplozije u količini podataka koji su dostupni u digitalnom obliku i brzog razvoja računala u zadnjih nekoliko godina počele su se razvijati velike neuronske mreže s mnogo slojeva, takozvane duboke neuronske mreže (engl. *deep neural networks*). Upotrebom takvih mreža danas je moguće riješiti veliki broj kompleksnih problema. Jedan od problema kojeg neuronske mreže vrlo uspješno rješavaju je klasifikacija rukom napisanih brojeva.

Tema ovog rada je jedna specifična vrsta neuronskih mreža koja daje izuzetno dobre rezultate pri obradi slikovnih podataka, a naziva se konvolucijska neuronska mreža. U radu će najprije biti objašnjeni osnovni pojmovi koji se vežu uz neuronske mreže i njihovu problematiku. Nakon toga bit će detaljno opisana arhitektura konvolucijske neuronske mreže i algoritmi učenja iste. Uz teoretsko objašnjenje mreže, u radu će biti objašnjena programska implementacija mreže i algoritma učenja mreže. Konvolucijska mreža testirat će se na skupu MNIST [2] rukom pisanih znamenki, koja je uobičajeni test za takve arhitekture.

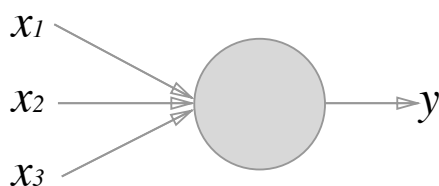
Ostatak rada organiziran je kako slijedi. U drugom poglavlju opisani su osnovni pojmovi koji se vežu uz umjetne neuronske mreže. U trećem poglavlju opisan je algoritam propagacije pogreške unatrag. U četvrtom poglavlju opisane su konvolucijske mreže. U petom poglavlju je opisano učenje konvolucijskih neuronskih mreža. U šestom poglavlju opisani su dobiveni rezultati, dok je u sedmom poglavlju dan zaključak.

2. Umjetne neuronske mreže

Umjetne neuronske mreže nastale su kao imitacija ljudskog mozga koji se sastoji od velikog broja živčanih stanica i uspješno obrađuje velik broj različitih informacija. Kako je navedeno u uvodu, ideja umjetnih neuronskih mreža nije nova. 1943. godine znanstvenici s *Massachusetts Institute of Technology* počeli su proučavati rad mozga i dali su matematički model neuronske mreže u okviru teorije automata. Deset godina kasnije počele su se pojavljivati prve praktične izvedbe, no tadašnja računala bila su preslaba kako bi se ti isti modeli koristili pri rješavanju kompleksnijih problema. Neuronske mreže danas vrlo su popularne i uspješno rješavaju velik broj problema. U sljedećim poglavljima opisat će se osnovni dijelovi svake neuronske mreže.

2.1. Neuron

Ideja umjetne neuronske mreže najlakše se može objasniti na primjeru jednostavnog *perceptrona*. Primjer jednog takvog neurona prikazan je na slici 2.1.



Slika 2.1: Primjer perceptrona

U danom primjeru sa slike ulazi u neuron označeni su s x_1 , x_2 , x_3 , dok je izlaz označen s y . Unutar svakoga neurona nalaze se težine za svaki od ulaza. Težine su, kao i ulazi, obični realni brojevi. Tako se u promatranom primjeru sa slike unutar neurona nalaze težine koje možemo označiti s w_1 , w_2 i w_3 . U ovome primjeru razmatrat će se TLU-perceptron (engl. *Threshold Logic Unit*). Izlaz takvog neurona može biti ili 0 ili 1, a definiramo ga na sljedeći način. Ako je suma umnožaka odgovarajućih težina i ulaza veća od parametra t izlaz će biti 1, dok će u suprotnome izlaz biti 0. Realni broj t

definira prag pri kojemu će se neuron aktivirati (engl. *threshold*). Izlaz neurona za dani primjer algebarski je prikazan izrazom (2.1).

$$y = \begin{cases} 0, & \text{za } \sum_{j=1}^3 w_j x_j < t \\ 1, & \text{za } \sum_{j=1}^3 w_j x_j \geq t \end{cases} \quad (2.1)$$

Ovime je definiran najjednostavniji oblik neurona, gdje težine zapravo predstavljaju koliko određeni ulaz doprinosi u odluci o aktiviranju neurona. Češći zapis izlaza jednostavnog neurona prikazan je u izrazu (2.2).

$$y = \begin{cases} 0, & \text{za } \sum_{j=1}^3 w_j x_j + b < 0 \\ 1, & \text{za } \sum_{j=1}^3 w_j x_j + b \geq 0 \end{cases} \quad (2.2)$$

Izraz (2.2) možemo dobiti iz izraza (2.1) ako vrijedi $b = -t$. Parametar b nazivamo prag (engl. *bias*), a na njega možemo gledati kao dodatnu težinu koja uvijek množi ulaz čija je vrijednost uvijek jednaka 1.

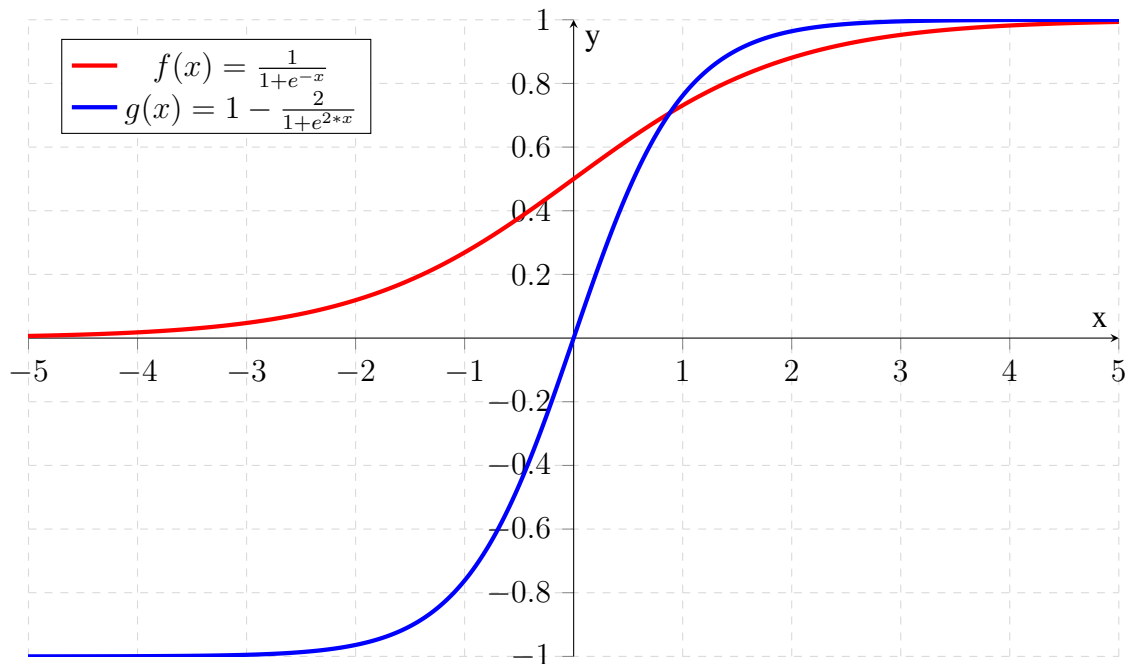
2.2. Aktivacijske funkcije

U prvome dijelu poglavlja opisan je jednostavni TLU-perceptron. Iako na prvi pogled izgleda kao dobar ideja, mreže s više takvih neurona nisu dale dobre rezultate. Jedan od glavnih razloga neuspjeha je što mali utjecaj jedne težine može imati veliki utjecaj na izlaz cijele mreže. Naime definirali smo da izlaz svakoga neurona može biti 0 ili 1. To znači da mala promjena težine može promijeniti izlaz neurona s 0 na 1 što je velika promjena. Uz takvo ponašanje mrežu je vrlo teško trenirati, stoga je potrebno smisliti drugačiju aktivacijsku funkciju kako bi dobili suprotno svojstvo.

Kao aktivacijska funkcija često se koristi sigmoidalna funkcija. Izlaz neurona za isti primjer iz prvog dijela poglavlja sada možemo zapisati pomoću izraza (2.3).

$$y = f\left(\sum_{j=1}^3 w_j x_j + b\right) \quad \text{uz} \quad f(x) = \frac{1}{1 + e^{-x}} \quad (2.3)$$

Iako sigmoidalni neuron algebarski ne izgleda vrlo slično *perceptronu* njihova ponašanja za iste ulaze vrlo su slična. Sigmoidalna funkcija za velike ulaze daje izlaz jako blizu broja jedan, dok za jako negativne ulaze daje izlaz blizu nule. Osim sigmoidalne funkcije često se koristi i tangens-hiperbolna funkcija. Graf navedenih funkcija dan je na slici (2.2).



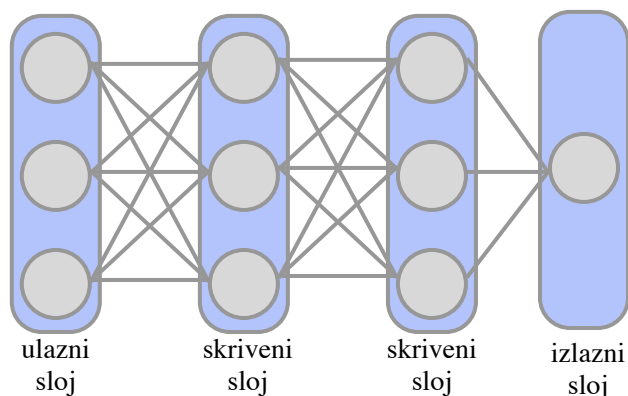
Slika 2.2: Graf aktivacijskih funkcija

2.3. Struktura neuronske mreže

U dosadašnjem tekstu svi primjeri odnosili su se samo na jedan neuron, no naravno svaka neuronska mreža sastoji se od više neurona. Kada se kaže struktura mreža, misli se na odnose između neurona odnosno kako su oni povezani unutar mreže.

Svaka neuronska mreža sastoji se od slojeva. U osnovnoj neuronskoj mreži postoje 3 vrste slojeva: ulazni sloj, skriveni sloj i izlazni sloj. Ulazni sloj, kao što mu i samo ime kaže, predstavlja prvi sloj neuronske mreže. Ulazi neurona u ulaznom sloju su ulazni podaci problema. Izlazni sloj mreže posljednji je sloj mreže i izlazi neurona koji se nalaze u njemu zapravo predstavljaju rješenja problema. Svi ostali slojevi koji se nalaze između ulaznog i izlaznog nazivaju se skriveni slojevi. Primjer strukture jedne jednostavne neuronske mreže ilustriran je na slici 2.3.

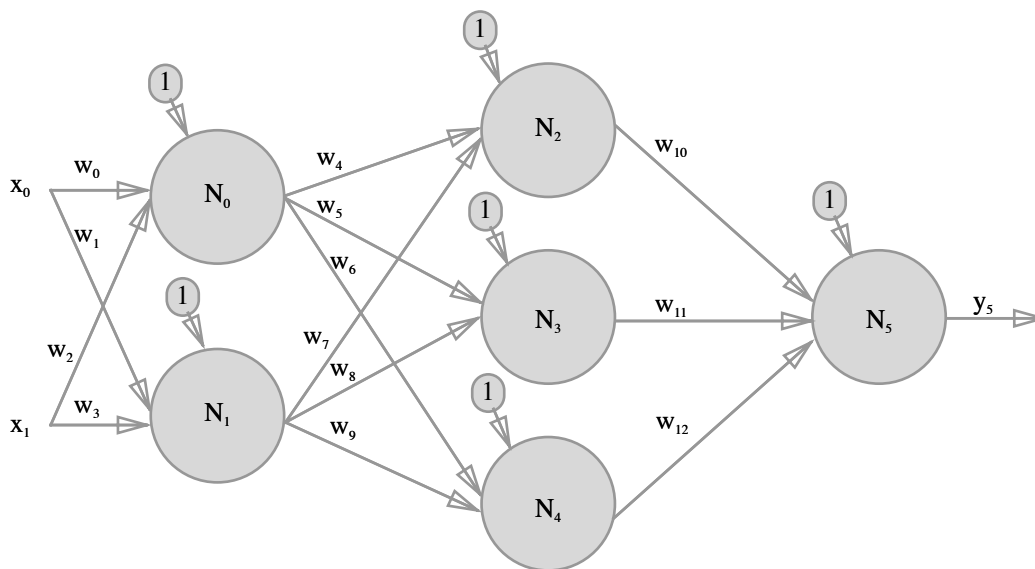
Kada se govori o neuronskim mrežama najčešće se misli na unaprijedne slojevite mreže. Unaprijedne slojevite neuronske mreže su neuronske mreže kod kojih se izlaz prethodnog sloja dovodi na ulaz sljedećeg sloja. Također kada se govori o strukturi mreže, bitno je naglasiti kako su povezani ulazi i izlazi neurona između dva sloja. U sljedećem primjeru objasniti će se rad jednostavne potpuno povezane mreže.



Slika 2.3: Struktura jednostavne neuronske mreže

2.4. Primjer rada jednostavne neuronske mreže

Na slici 2.4 prikazan je primjer jedne jednostavne neuronske mreže na kojoj će biti objašnjen rad mreže. Neuroni su označeni slovom N , dok su njihovi ulazi označeni slovom x , a izlazi slovom y . Težine za svaki pojedini ulaz označene su slovom w uz odgovarajući indeks. Za svaki neuron definiran je i prag, te za svaki neuron on iznosi 1.



Slika 2.4: Primjer rada neuronske mreže

Kako bi primjer bio što jednostavniji pretpostavimo kako svi neuroni imaju jednaku aktivacijsku funkciju. Aktivacijska funkcija za svaki neuron je sigmoidalna (logistička) funkcija definirana ranije.

Unutar tablice 2.1 nalaze se vrijednosti svake težine unutar mreže. Uz težine po-

Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12
Težina	-1	1	0	-1	-1	1	-2	1	0	1	-1	-2	0

Tablica 2.1: Tablica težina za dani primjer

trebno je definirati i ulaze x_0 i x_1 , neka je $x_0 = 0.5$, a $x_1 = -1$. Dok su pragovi svih neurona jednaki 1.

Prvi korak u određivanju konačnog izlaza mreže, izlaza y_5 , je odrediti izlaze neurona prvog sloja. Za neuron N_0 suma umnožaka težina i ulaza jednaka je: $0.5 * (-1) + (-1) * 0 = -0.5$, no potrebno je dodati i prag, pa je ukupni izlaz prije djelovanja aktivacijske funkcije jednak 0.5. Aktivacijska funkcija, koja se nalazi unutar svakoga neurona, preslikat će sumu umnožaka težina i ulaza u jedan broj prema izrazu (2.4).

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.4)$$

Stoga vrijedi $y_0 = f(0.5) = 0.62$. Postupak za neuron N_1 je identičan. Suma umnožaka težina i ulaza jednaka je $0.5 * 0 + (-1) * -1 = 1$, pa je izlaz $y_1 = f(1 + 1) = f(2) = 0.88$.

Kako je mreža potpuno povezana izlazi y_0 i y_1 sada zapravo postaju ulazi za neurone skrivenog sloja. Postupak za određivanje izlaza neuron N_2 je isti kao i za prethodne neurone. Suma umnožaka ulaza i težina jednaka je: $(-1) * 0.62 + 1 * 0.88 = 0.26$, pa je izlaz $y_2 = f(0.26 + 1) = 0.78$. Izlazi preostalih neurona iznose: $y_3 = 0.83$, $y_4 = 0.65$.

Izlazi neurona skrivenog sloja sada postaju ulazi neurona N_5 koji se nalazi u izlaznom sloju. Izlaz mreže računamo istim postupkom, suma umnožaka ulaza i težina jednaka je: $(-1) * 0.78 + (-2) * 0.83 + 0 * 0.65 = -2.44$. Izlaz posljednjeg neurona zapravo je i izlaz cijele mreže, a on iznosi: $y_5 = f(-2.44 + 1) = 0.19$. Ovim primjerom ilustriran je rad potpuno povezane unaprijedne neuronske mreže.

2.5. Programska izvedba rada neuronske mreže

Prethodno objašnjeni postupak rada jednostavne neuronske potpuno povezane mreže moguće je vrlo jednostavno izvesti i programski. Kako bi ostvarili rad mreže iz prethodnog primjera potrebno je definirati tri razreda: razred `NeuralNetwork`, razred `Layer` i razred `Neuron`. Razred `Neuron` opisuje jedan neuron u mreži. Unutar razreda `Neuron` nalazi se metoda za određivanje izlaza jednog neurona i metoda za inicijalno postavljanje težina na vrijednosti između -1 i 1. Razred `Layer` opisuje jedan sloj

u mreži. Unutar svakoga sloja nalazi se lista neurona u njemu. Unutar razreda `Layer` nalazi se i metoda za izračun izlaza svih neurona u sloju. Razred `NeuralNetwork` opisuje cijelu neuronsku mrežu. Razred `NeuralNetwork` sadrži listu svih slojeva u mreži i metodu koja računa izlaz cijele mreže.

Primjer koda 2.1: Računanje izlaza jednog neurona

```
1 Double calculateOutput(List<Double> inputs) {
2     int inputSize = inputs.size();
3     Double output = 0.0;
4     for (int i = 0; i < inputSize; i++) {
5         output += inputs.get(i) * weights.get(i);
6     }
7     output = activate(output + bias);
8     return output;
9 }
10
11 Double activate(Double output) {
12     return 1/(1+Math.exp(-output));
13 }
```

Primjer koda 2.1 prikazuje metodu koja računa izlaz neurona za dane ulaze. U primjeru 2.1 koristi se aktivacijska funkcija perceptrona opisana ranije. Unutar razreda `Neuron` nalazi se i metoda za inicijalizaciju težina i praga, prikazana primjerom koda 2.2.

Primjer koda 2.2: Inicijalno postavljanje težina

```
1 void setRandomWeights(int numberOfWeights) {
2     Random rand = RNG.getInstance();
3     for (int i = 0; i < numberOfWeights; i++) {
4         if (rand.nextBoolean()) {
5             weights.add(rand.nextDouble());
6         } else {
7             weights.add(rand.nextDouble() * (-1));
8         }
9     }
10    bias = rand.nextDouble();
11    if (rand.nextBoolean()) {
12        bias *= -1;
13    }
```

```
13     }
14 }
```

U ovome jednostavnom primjeru sve težine i prag postavljaju se na slučajne vrijednosti između -1 i 1.

Razred `Layer` sadrži metodu za izračunavanje izlaza svih neurona koji pripadaju tom sloju. Primjer takve metode prikazan je primjerom koda 2.3.

Primjer koda 2.3: Izračunavanje izlaza sloja

```
1 List<Double> propagate(List<Double> inputs) {
2     List<Double> outputs = new ArrayList<>();
3     for (Neuron neuron : neurons) {
4         outputs.add(neuron.calculateOutput(inputs));
5     }
6     return outputs;
7 }
```

U danom primjeru metoda `propagate` prima listu ulaznih vrijednosti koje dolaze na ulaz sloja. Zatim se nad svakim neuronom poziva metoda za izračunavanje njegovog izlaza za dani ulaz. Kako govorimo o jednostavnoj potpuno povezanoj mreži svaki neuron prima sve ulaze koji se dovode na ulaz sloja.

Razred `NeuralNetwork` opisuje cijelu mrežu i sadrži metodu za računanje izlaza cijele mreže. Primjer takve metoda dan je u primjeru koda 2.4.

Primjer koda 2.4: Izračunavanje izlaza mreže

```
1 List<Double> calculateOutput(List<Double> inputs) {
2     List<Double> outputs = inputs;
3     for (Layer layer : layers) {
4         outputs = layer.propagate(outputs);
5     }
6     return outputs;
7 }
```

Prikazana metoda kao argument prima ulazne vrijednosti koje se dovode na ulaz mreže. Ulazne vrijednosti dovode se na ulaz prvog sloja te se poziva metoda za izračun izlaza prvog sloja. Izlaz prvog sloja se pamti te se on dovodi na ulaz sljedećeg sloja. Cijeli proces ponavlja se za sve slojeve u mreži, dok izlaz zadnjeg sloja zapravo predstavlja i izlaz cijele mreže.

3. Učenje neuronskih mreža

U prošlom poglavlju predstavljen je model jednostavne neuronske mreže, kojom je moguće modelirati razne probleme. No kako bi neuronska mreža mogla rješavati neki konkretan problem potrebno je podesiti njene težine kako bi za pojedini uzorak izlaz neuronske mreže bio ispravan. Najčešći algoritam koji se koristi za učenje neuronskih mreža je algoritam propagacije pogreške unatrag (engl. *error backpropagation*). U nastavku teksta opisan je rad algoritma propagacije pogreške unatrag.

3.1. Algoritam propagacije pogreške unatrag

Neuronske mreže koje imaju linearnu aktivacijsku funkciju mogu opisati samo linearne odnose. Kako je često potrebno opisivati i visoko nelinearne funkcije potrebno je koristiti drugu aktivacijsku funkciju u neuronima. Kako bi opisali nelinearnu funkciju, neuroni neuronske mreže moraju sadržavati aktivacijske funkcije koje su također nelinearne [1].

Algoritam učenja neuronske mreže sastoji se od dvije faze: unaprijedne faze i unazadne faze. U prvoj fazi algoritma na ulaze neuronske mreže dovode se ulazi za koje je poznato rješenje, računaju se izlazi mreže za dane ulaze i određuje se pogreška. U drugoj fazi izračunata greška propagira se unazad i računaju se gradijenti pogreške. Gradijent se koristi kako bi znali kako treba promijeniti težine pojedinog neurona kako bi se greška smanjila. Pri računanju gradijenata najprije se određuju greške zadnjeg sloja. Zatim se ukupna greška zadnjeg sloja propagira na predzadnji sloj, te se za predzadnji sloj određuje kako je svaki neuron utjecao na grešku. Zatim se određuju komponente gradijenta i ažurira vrijednost težine u ovisnosti o izračunatom gradijentu.

Oznakom y_j označava se izlaz j -tog neurona koji pripada trenutnom sloju, dok će se izlaz neurona koji pripada prethodnom sloju označavati oznakom y_i . Oznakom z_j označava se ukupni ulaz neurona koji pripada trenutnom sloju, dok se težina koja spaja i -ti neuron prethodnog sloja s j -tim neuronom trenutnog sloja označavati s w_{ij} .

Prvi korak u određivanju gradijenata težina je odrediti grešku trenutnog sloja, a to

je označeno s $\frac{\partial E}{\partial z_j}$. Izraz (3.1) prikazuje kako se određuje greška trenutnog sloja.

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j} \quad (3.1)$$

U izrazu (3.1) $\frac{\partial E}{\partial y_j}$ označava parcijalnu derivaciju greške po izlazu neurona, dok $\frac{\partial y_j}{\partial z_j}$ označava parcijalnu derivaciju izlaza neurona po ulazu neurona (pod pojmom ulaz neurona misli se na vrijednost koja se dovodi na aktivacijsku funkciju). Slovo E u izrazu označava grešku (engl. *Error*), koja se najčešće računa kao srednje kvadratno odstupanje.

$$E = \frac{1}{2} \sum_j (y_j - t_j)^2 \quad (3.2)$$

Izrazom (3.2) prikazano je kako se računa greška izlaznih neurona, gdje y_j označava izlaz j -tog neurona, a t_j označava očekivani izlaz j -tog neurona (uz fiksirani ulaz mreže).

Nakon što je definiran izraz za izračun greške, moguće je odrediti parcijalnu derivaciju greške po izlazu neurona iz izraza (3.1).

$$\frac{\partial E}{\partial y_j} = \frac{1}{2} \frac{\partial}{\partial y_j} (y_j - t_j)^2 = \frac{1}{2} \cdot 2(y_j - t_j) = (y_j - t_j) \quad (3.3)$$

Važno je uočiti kako u izrazu (3.3) nema sume jer na parcijalnu derivaciju greške po izlazu utječe samo izlaz promatranog neurona.

Ako se za aktivacijsku funkciju koristi sigmoidalna funkcija $y_j = f(z_j) = \frac{1}{1 + e^{-z_j}}$, izraz za parcijalnu derivaciju izlaza neurona po ulazu neurona postaje:

$$\begin{aligned} \frac{\partial y_j}{\partial z_j} &= \frac{\partial}{\partial z_j} \left(\frac{1}{1 + e^{-z_j}} \right) = \frac{-1(-e^{-z_j})}{(1 + e^{-z_j})^2} = \\ &= \frac{1}{1 + e^{-z_j}} \frac{e^{-z_j}}{1 + e^{-z_j}} = y_j \frac{e^{-z_j}}{1 + e^{-z_j}} = y_j \frac{(1 + e^{-z_j}) - 1}{1 + e^{-z_j}} = \\ &= y_j \frac{1 + e^{-z_j} - 1}{1 + e^{-z_j}} = y_j (1 - y_j). \end{aligned} \quad (3.4)$$

Kombiniranjem izraza (3.3) i (3.4) dobivamo sljedeći izraz:

$$\frac{\partial E}{\partial z_j} = (y_j - t_j) y_j (1 - y_j). \quad (3.5)$$

Nakon računanja greške trenutnog sloja, moguće je odrediti grešku prethodnog sloja. Za određivanje greške prethodnog sloja potrebno je sumirati utjecaj svakoga neurona u tom sloju. Izrazom (3.6) prikazano je kako algebarski odrediti parcijalnu derivaciju greške po izlazima neurona prethodnog sloja.

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial y_i} = \sum_j w_{ij} \frac{\partial E}{\partial z_j} \quad (3.6)$$

Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12
Težina	-1	1	0	-1	-1	1	-2	1	0	1	-1	-2	0

Tablica 3.1: Tablica težina za dani primjer

Cilj algoritma unazadne propagacije je odrediti parcijalnu derivaciju greške po težinama kako bi znali kako je potrebno promijeniti težine da bi se greška smanjila. Izračun parcijalne derivacije greške po težinama prikazana je izrazom (3.7).

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} y_i \quad (3.7)$$

Važno je uočiti kako parcijalna derivacija ukupnog ulaza po težinama zapravo predstavlja izlaz prethodnog sloja.

Nakon što odredimo sve parcijalne derivacije prema izrazu (3.7) potrebno je ažurirati težine kako bi se ukupna greška smanjila. Težine se ažuriraju prema izrazu (3.8).

$$w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} \quad (3.8)$$

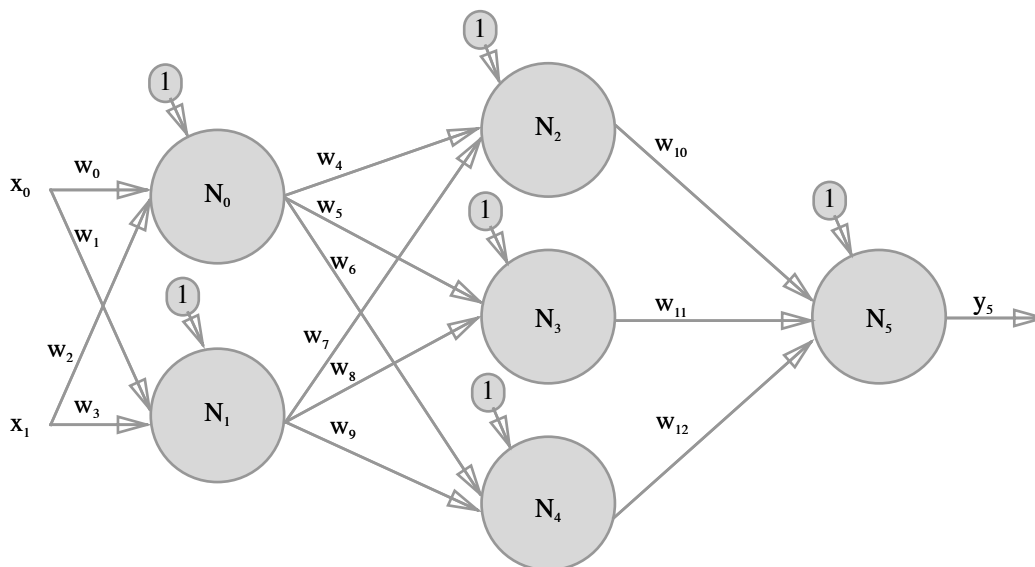
Unutar navedenog izraza nalazi se znak η koji označava parametar kojim određujemo brzinu učenja. Ovime je definiran algoritam učenja jednostavne potpuno povezane neuronske mreže.

3.2. Primjer algoritma učenja

Kako bi objasnili rad algoritam unazadne propagacije korištenjem gradijentnog spusta u sljedećem primjeru koristiti će se sigmoidalna funkcija kao aktivacijska funkcija svakog neurona. Neuronska mreža koja će se koristiti u primjeru prikazana je slikom 3.1, dok su težine dane mreže zadane tablicom 3.1. Primjer je potpuno jednak primjeru iz poglavlja 2.4 gdje je objašnjena unaprijedna propagacija rješenja.

Kako je ranije navedeno algoritam unazadne propagacije sastoji se od dvije faze. Prva faza je unaprijedna faza gdje se računa odziv mreže za poznati uzorka i greška odziva. Druga faza je faza unazadne propagacije gdje se greška propagira unazad te se ažuriraju težine i pragovi mreže kako bi se greška smanjila.

Kako je primjer potpuno jednak primjeru iz poglavlja 2.4, dobivene rezultate možemo iskoristiti i u ovom primjeru. Izlazi neurona jednaki su: $y_0 = 0.62$, $y_1 = 0.88$, $y_2 = 0.78$, $y_3 = 0.83$, $y_4 = 0.65$, $y_5 = 0.19$. U ranije navedenom primjeru očekivani izlaz da dane ulaze nije bio definiran, no kako bi odredili grešku potrebno je odrediti očekivani izlaz. Neka očekivani izlaz za ulaze $x_0 = 0.5$ i $x_1 = -1$ bude 1, odnosno vrijedi $t_j = 1$.



Slika 3.1: Primjer potpuno povezane neuronske mreže

Najprije je potrebno odrediti grešku izlaznog sloja koju računamo prema ranije definiranom izrazu (3.5).

$$\frac{\partial E}{\partial z_5} = (y_5 - t_5)y_5(1 - y_5) = (0.19 - 1)0.19(1 - 0.19) = -0.12.$$

Zatim je potrebno odrediti parcijalne derivacije grešaka po težinama izlaznog sloja:

$$\frac{\partial E}{\partial w_{10}} = \frac{\partial E}{\partial z_5} y_2 = -0.12 \cdot 0.78 = -0.09$$

$$\frac{\partial E}{\partial w_{11}} = \frac{\partial E}{\partial z_5} y_3 = -0.12 \cdot 0.83 = -0.10$$

$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial z_5} y_4 = -0.12 \cdot 0.65 = -0.08$$

Parcijalnu derivaciju greške po pragu izlaznog sloja računamo kao:

$$\frac{\partial E}{\partial b_5} = \frac{\partial E}{\partial z_5} \frac{\partial z_5}{\partial b_5} = -0.12 \cdot 1 = -0.12$$

Sljedeći korak je propagirati unazad grešku kako bi odredili greške skrivenog sloja.

$$\frac{\partial E}{\partial y_i} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial y_2} = (-1) \cdot -0.12 = 0.12, \quad \frac{\partial E}{\partial y_3} = (-2) \cdot -0.12 = 0.24, \quad \frac{\partial E}{\partial y_4} = 0 \cdot -0.12 = 0$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$$

$$\frac{\partial E}{\partial z_2} = 0.12 \cdot f'(z_j) = 0.12 \cdot 1.26(1 - 1.26) = -0.04$$

Postupak je jednak za sve ostale neurone u sloju. Preostalo je još izračunati parcijalne derivacije greške po težinama. Parcijalne derivacije greške po težinama računaju se na sljedeći način:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} y_i$$

$$\frac{\partial E}{\partial w_4} = \frac{\partial E}{\partial z_2} \cdot y_0 = -0.4 \cdot 0.62 = -0.25$$

$$\frac{\partial E}{\partial b_2} = \frac{\partial E}{\partial z_j} = -0.4$$

Ostale parcijalne derivacije greške po težinama računaju se na potpuno jednak način.

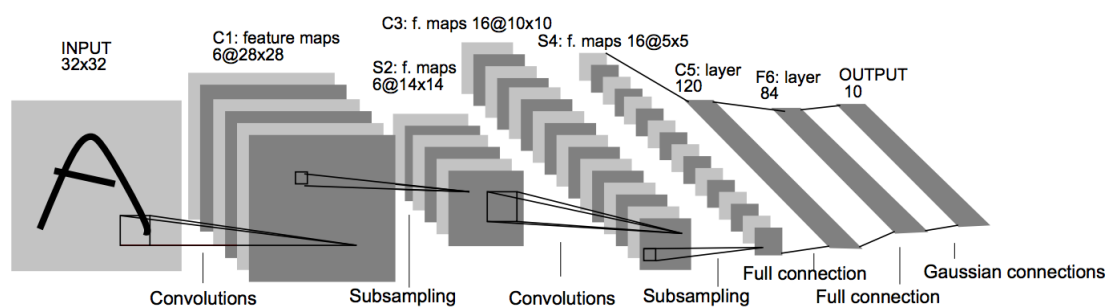
Nakon što su sve parcijalne derivacije greške po težinama izračunane potrebno je ažurirati sve težine u mreži. Težine ažuriramo prema ranije spomenutom izrazu (3.8).

Tako na primjer težina w_{10} nakon jedne iteracije algoritma postaje:

$$w_{10} = w_{10} - \eta \cdot (-0.09)$$

4. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže mogu se opisati kao nadogradnja nad običnim višeslojnim unaprijednim mrežama. Konvolucijska, kao i obična, neuronska mreža sastoji se od jednog ulaznog, jednog izlaznog i jednog ili više skrivenih slojeva. Kod konvolucijskih neuronskih mreža specifični su konvolucijski slojevi i slojevi sažimanja. Osim njih često se koriste i potpuno povezani slojevi. Konvolucijske neuronske mreže najčešće kreću s jednim ili više konvolucijskih slojeva, zatim slijedi sloj sažimanja, pa ponovo konvolucijski sloj i tako nekoliko puta. Mreža najčešće završava s jednim ili više potpuno povezanih slojeva koji služe za klasifikaciju. Arhitektura konvolucijskih neuronskih mreža pokazala se izrazito dobra u radu sa slikama i prepoznavanju značajki s istih [3]. Stoga se konvolucijska neuronska mreža nameće kao vrlo dobro rješenje pri prepoznavanju objekata na slici. Na slici 4.1 prikazana je arhitektura konvolucijske neuronske mreže poznatije pod imenom "LeNet-5". Prikazana arhitektura vrlo je popularna, te je uz takvu arhitekturu dobivena greška od 0.8% na skupu podataka MNIST.



Slika 4.1: Konvolucijska neuronska mreža LeNet-5 (slika preuzeta s [3])

U nastavku svaki od navedenih slojeva bit će objašnjen detaljnije.

4.1. Konvolucijski sloj

Konvolucijski sloj jedan je od glavnih dijelova svake konvolucijske neuronske mreže. Svaki konvolucijski sloj sastoji se od filtara koje sadrže težine koje je potrebno naučiti kako bi mreža davala dobre rezultate. Filtri su najčešće manjih prostornih dimenzija od ulaza, no uvijek su jednake dubine kao i ulaz. Tako ako se na primjer na ulazu konvolucijskog sloja nalazi slika s tri komponente boja (RGB) i veličine je 25x25 točaka, filter može biti matrica težina veličine 5x5, ali potrebno je imati tri takve matrice, odnosno jednu za svaku komponentu boje.

Pri unaprijednoj fazi potrebno je konvoluirati filter s ulazom. Rezultat konvolucije je dvodimenzionalna aktivacijska mapa koja predstavlja odziv filtra na svakoj prostornoj poziciji. Zapravo mreža će naučiti težine unutar filtra kako bi se filter "aktivirao" na mjestima gdje prepoznaje određena slikovna svojstva, kao na primjer određene vrste rubova ili slično.

Svaki sloj najčešće se sastoji od više filtara i svaki od njih će generirati jednu dvodimenzionalnu aktivacijsku mapu. Izlaz sloja bit će sve aktivacijske mape, odnosno izlaz će biti više dvodimenzionalnih matrica gdje svaka predstavlja jednu dubinu izlaza.

Kako bi bilo moguće definirati točnu veličinu izlaza konvolucijskog sloja potrebno je definirati i korak pomaka filtra (engl *stride*). Korak pomaka filtra označava za koliko će se filter pomicati po širini odnosno visini tijekom konvolucije.

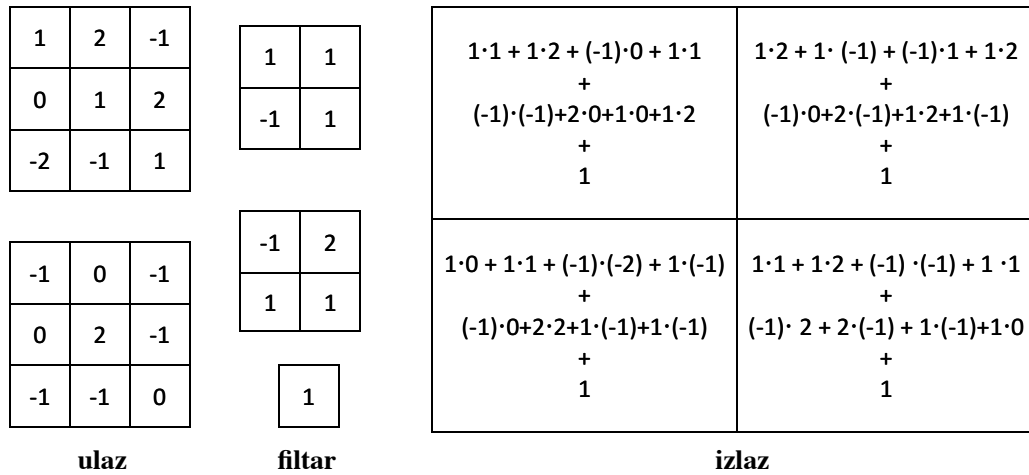
Sada je moguće definirati točnu veličinu izlaza konvolucijskog sloja. Širina izlaza označavat će se s W_y , a širina ulaza s W_x . Korak pomaka filtra označavat će se sa S , dok će se s oznakom F označavati veličina (kvadratnog) filtra. Širina izlaza definirana je izrazom (4.1).

$$W_y = \frac{W_x - F}{S} + 1 \quad (4.1)$$

U ovome radu širina svih ulaza bit će jednaka visini ulaza stoga vrijedi: $W_y = H_y$, gdje H_y označava visinu izlaza.

4.1.1. Primjer rada konvolucijskog sloja

Kako bi objasnili rad konvolucijskog sloja potrebno je definirati dvije dvodimenzionalne matrice koje predstavljaju ulaz u konvolucijsku mrežu s dvije komponente (dubine 2). Kako se ulaz sastoji od dvije komponente, onda i dubina filtra mora biti jednaka 2. Kao i kod jednostavne neuronske mreže gdje je svaki neuron imao jedan dodatan ulaz koji je uvijek bio jedan i za kojeg je bila definirana dodatna težina (prag), tako i kod filtra postoji jedna dodatna težina za prag. Opisani primjer prikazan je na slici 4.2.



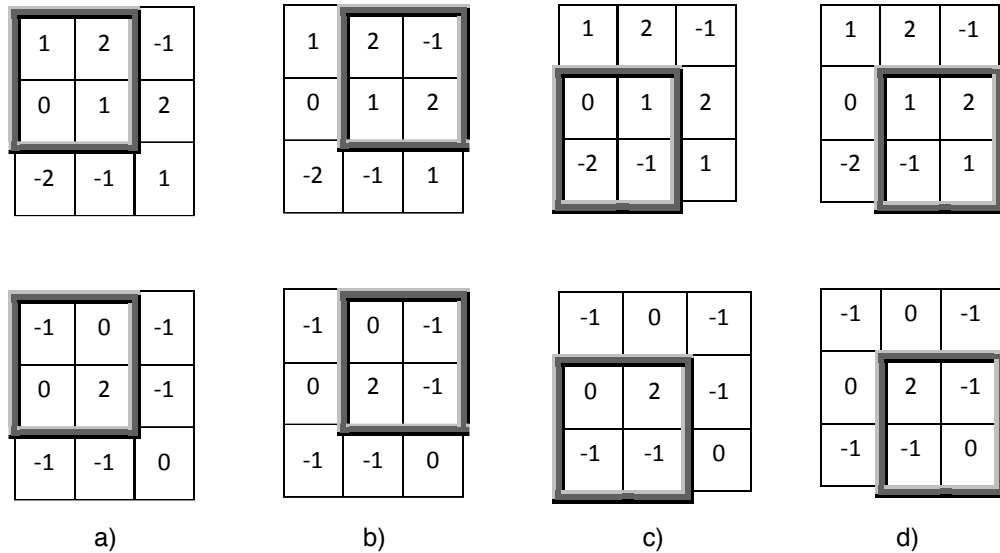
Slika 4.2: Primjer rada konvlucijskog sloja

Širina izlazne mape za dani primjer može se također izračunati prema izrazu (4.1), uz veličinu kvadratnog filtra, $F = 2$, i korakom $S = 1$.

$$W_y = \frac{W_x - F}{S} + 1 = \frac{3 - 2}{1} + 1 = 2$$

Kako bi izračunali izlaz konvolucijskog sloja, odnosno odredili aktivacijsku mapu, potrebno je provesti konvoluciju nad ulazom i filtrom. Konvolucija se provodi tako da odgovarajuće ulaze pomnožimo s težinama, dobivene vrijednosti potrebno je sumirati i dodati prag. Potrebno je naglasiti da pri izračunu vrijednosti za jedno polje potrebno je sumirati umnoške ulaza i težina svih dubina. Za izračun vrijednosti sljedećeg polja potrebno je pomjeriti filter za definirani pomak i ponoviti postupak. Pomak filtra opisan je slikom 4.3. Iz slike je vidljivo kako postoje četiri moguće pozicije na kojima se filter može nalaziti stoga je izlaz matrica veličine 2x2. Također potrebno je uočiti kako svaki korak sa slike 4.3 odgovara izračunu jedne vrijednosti u izlaznoj mapi sa slike

4.2. Nakon što se izračunaju sve vrijednosti, kao i kod jednostavne neuronske mreže, dobivene vrijednosti potrebno je provesti kroz aktivacijsku funkciju.



Slika 4.3: Primjer rada konvolucijskog sloja

Operaciju konvolucije najlakše je opisati algebarski, stoga je izrazom (4.2) opisana veza između ulaza i izlaza. Ulaz je zapravo izlaz iz prethodnog sloja pa je označen s $y_d^{l-1}(i, j)$, gdje i i j određuju poziciju unutar matrice, d označava dubinu, dok l označava o kojem sloju se radi. Težine unutar filtra označene su oznakom w . Kao i u ranijim primjerima, veličina filtra označena je s F , funkcija f predstavlja aktivacijsku funkciju, dok b označava prag.

$$y^l(i, j) = \sum_{a=0}^{F-1} \sum_{b=0}^{F-1} \sum_{d=0}^{D-1} f(w_d(i, j) \cdot y_d^{l-1}(i + a, j + b) + b) \quad (4.2)$$

Potrebno je napomenuti kako se vrijednosti a i b povećavaju za zadani korak S .

4.1.2. Programska izvedba rada konvolucijskog sloja

Prethodno objašnjeni postupak rada konvolucijskog sloja moguće je jednostavno izvesti i programski. U primjeru koda 4.1 prikaza je programska implementacija izračuna izlaza konvolucijskog sloja.

Primjer koda 4.1: Izračunavanje izlaza konvolucijskog sloja

```
1 public List<double[][]> calculateOutput() {
```

```

2    List<double[][]> result = new LinkedList<>();
3    int sw = inputData.getWidth();
4    int w2 = (sw - sx) / stride + 1;
5    int sh = inputData.getHeight();
6    int h2 = (sh - sx) / stride + 1;
7
8    for (int n = 0; n < filters; n++) {
9        double[][] = new double[h2][w2];
10       for (int i = 0, tempi = 0; tempi <= sw - sx; tempi +=
           stride, i++) {
11           for (int j = 0, tempj = 0; tempj <= sh - sx; tempj
               += stride, j++) {
12               for (int a = 0; a < sx; a++) {
13                   for (int b = 0; b < sx; b++) {
14                       for (int d = 0; d < inputDepth; d++) {
15                           results[i][j] += inputData.get(d)[i +
                               a][j + b] * allWeights.get(n *
                               inputDepth + d)[a][b];
16                       }
17                   }
18               }
19               results[i][j] += biases[n];
20           }
21       }
22       result.add(results);
23   }
24   return activateNeurons(result);
25 }

```

Metoda `calculateOutput` zadužena je za izračunavanje izlaza konvolucijskog sloja. Metoda se nalazi u razredu `ConvLayer`, gdje se osim ove metode, nalaze i ostale pomoćne metode kao i potrebni parametri i podaci za izračun izlaza konvolucijskog sloja. Ulaz konvolucijskog sloja spremljen je u listu `inputData` koja u sebi sadrži matrice s vrijednostima ulaza. Svaka matrica označava jednu komponentu (dubinu) ulaza. Osim ulaznih vrijednosti razred `ConvLayer` sadrži i listu `allWeights` u kojoj su spremljene sve težine filtara. Težine su također spremljene u obliku matrica, ali su spremljene na specifičan način. Naime kako se svaki filter može sastojati od više matrica (svaka matrica za jednu komponentu ulaza), u listi `allWeights` najprije se

nalaze sve matrice za prvi filter, zatim sve matrice za drugi filter i tako dalje. Razred `ConvLayer` sadrži i polje `biases` u kojemu su spremljene vrijednosti pragova za sve filtre.

U prvih šest linija koda definira se veličina izlaza. U liniji 8 počinje iteriranje najprije po svim filtrima. Zatim u liniji 10 i 11 iterira se po svim pozicijama ulaza na kojima se filter može nalaziti. U linijama 12 i 13 iterira se po svim pozicijama unutar filtra, dok se u liniji 14 iterira po svim dubinama. Dobivenu matricu nakon obavljene konvolucije potrebno je predati metodi `activateNeurons` koja će djelovati aktivacijskom funkcijom na predanu strukturu.

4.2. Sloj sažimanja

Slojevi sažimanja često se koriste u konvolucijskim neuronskim mrežama nakon nekoliko konvolucijskih slojeva s ciljem smanjivanja rezolucije mapi. Osim smanjivanja rezolucije slojevi sažimanja povećavaju prostornu invarijantnost (neosjetljivost na manje pomake značajki u uzorku) neuronske mreže [5].

Kao i kod konvolucijskog sloja manji dio uzorka grupira se i obrađuje se, te rezultira jednom vrijednošću. Postoji nekoliko vrsta sažimanja. Najčešći načini su sažimanje usrednjavanjem i sažimanje maksimalnom vrijednosti. Kod sažimanja usrednjavanjem grupirani podaci zamjenjuju se aritmetičkom sredinom grupiranih vrijednosti, dok se kod sažimanja maksimalnom vrijednosti grupirane vrijednosti zamjenjuju maksimalnom vrijednošću. Važno je uočiti kako broj komponenata (dubina) prije sloja sažimanja i nakon sažimanja ostaje isti.

4.2.1. Programska izvedba sažimanja maksimalnom vrijednosti

U ovome potpoglavlju objašnjen je primjer koda 4.2 kojim je programski ostvareno sažimanje maksimalnom vrijednosti.

Primjer koda 4.2: Sažimanje maksimalnom vrijednosti

```
1 public List<double[][]> calculateOutput() {
2     List<double[][]> allResults = new LinkedList<>();
3     int sw = allInputs.getWidth();
4     int sh = allInputs.getHeight();
5     maxLocations = new LinkedList<>();
6
7     for (int m = 0; m < depth; m++) {
```



```

8      double[][] results = new double[sw / sx][sh / sx];
9      Node[][] maxLocation = new Node[sw / sx][sh / sx];
10     for (int i = 0, tempi = 0; tempi <= sw - sx; tempi +=
        sx, i++) {
11         for (int j = 0, tempj = 0; tempj <= sh - sx; tempj
            += sx, j++) {
12             double maxVal = allInputs.get(m)[i][j];
13             Node maxLoc = new Node(0, 0);
14             for (int a = 0; a < sx; a++) {
15                 for (int b = 0; b < sx; b++) {
16                     double val = allInputs.get(m)[i + a][j + b];
17                     if (maxVal < val) {
18                         maxVal = val;
19                         maxLoc = new Node(a, b);
20                     }
21                 }
22             }
23             maxLocation[i][j] = maxLoc;
24             results[i][j] = maxVal;
25         }
26     }
27     allResults.add(results);
28     maxLocations.add(maxLocation);
29 }
30 return allResults;
31 }

```

Metoda `calculateOutput` zadužena je za izračunavanje izlaza sloja sažimanja. Metoda se nalazi unutar razreda `MaxPoolLayer`, gdje se nalazi i lista `allInputs` koja sadrži matrice ulaznih vrijednosti. Kao i kod konvolucijskog sloja, svaka matrica označava jednu komponentu ulaza. Izlaz sloja sažimanja sprema se u listu `allResults`. Osim izračuna izlaznih vrijednosti unutar metode `calculateOutput` pamte se lokacije na kojima se nalaze maksimalne vrijednosti unutar određenog filtra. Ovaj podatak bit će potreban pri učenju mreže. Podaci o maksimalnim lokacijama zapisani su u objektima koji su primjerci razreda `Node`. Razred `Node` pomoćni je razred unutar kojeg se nalazi x i y koordinata maksimalne lokacije.

U prvih 5 linija koda definiraju se potrebne strukture i računa veličina izlaza. U

liniji 7 počinje iteriranje po svim komponentama (dubinama) ulaza. Zatim u liniji 10 i 11 iterira se po svim pozicijama ulaza na kojima se filter može nalaziti. U linijama 14 i 15 iterira se po svim pozicijama filtera. Unutar svakog filtera određuje se maksimalna vrijednost čija se lokacija pamti.

5. Učenje konvolucijske neuronske mreže

U ranijem poglavlju objašnjen je algoritam unazadne propagacije kojim je pokazano kako se može naučiti jednostavna neuronska mreža. Isti algoritam može se primijeniti i na konvolucijske neuronske mreže. Kako je već ranije navedeno konvolucijska neuronska mreža sastoji se nekoliko različitih slojeva. Zadnji slojevi najčešće su potpuno povezani slojevi. Takvi slojevi uče se na jednak način opisan ranije, no za konvolucijske slojeve i slojeve sažimanja postupak je nešto drugačiji. Logika iza samog algoritma koja govori kako je potrebno odrediti parcijalne derivacije greške po težinama je jednaka, no sami izrazi za određivanje parcijalnih derivacija su drugačiji. U ovome poglavlju bit će objašnjeno kako se uče konvolucijske neuronske mreže algoritmom unazadne propagacije.

5.1. Konvolucijski sloj

Kao i u ranije objašnjenom primjeru s jednostavnom potpuno povezanom neuronskom mrežom učenje mreže kreće s kraja prema početku. Najprije se računa greška posljednjeg sloja koja se propagira prema ranijim slojevima. Cilj učenja konvolucijskog sloja je naučiti težine koje se nalaze unutar filtra. Kako bi naučili težine, potrebno je odrediti parcijalne derivacije greške po težinama. Parcijalne derivacije greške po težinama određujemo prema izrazu (5.1).

$$\frac{\partial E}{\partial w_{ab}} = \sum_{i=0}^{N-F} \sum_{j=0}^{N-F} \sum_{d=0}^{D-1} \frac{\partial E}{\partial z_{ij}^l} y_{(i+a)(j+b)}^{l-1,d} \quad (5.1)$$

Kao i ranije težine su označene s w , dok su s a i b određene pozicije težina unutar filtra. Suma umnožaka svih ulaza s pripadnim težina označava se sa z , dok je izlaz označen slovom y . Veličina ulaza (širina, odnosno visina) označena je s N , dok je veličina kvadratnog filtra označena slovom F . Dubina ulaznog sloja označena je s d .

Iz prethodnog izraza potrebno je objasniti kako se izračunava $\frac{\partial E}{\partial z_{ij}^l}$, a to je prikazano izrazom (5.2).

$$\frac{\partial E}{\partial z_{ij}^l} = \frac{\partial E}{\partial y_{ij}^l} \frac{\partial y_{ij}^l}{\partial z_{ij}^l} = \frac{\partial E}{\partial y_{ij}^l} \frac{\partial}{\partial z_{ij}^l} (f(z_{ij}^l)) = \frac{\partial E}{\partial y_{ij}^l} f'(z_{ij}^l) \quad (5.2)$$

Iz izraza (5.1) i (5.2) vidljivo je kako je za izračunavanje parcijalne derivacije greške po težinama potrebno znati parcijalnu derivaciju greške po izlazu trenutnog sloja, koja će biti propagirana. Izraz (5.3) prikazuje kako se greška propagira.

$$\frac{\partial E}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{F-1} \sum_{b=0}^{F-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^l} \frac{\partial x_{(i-a)(j-b)}^l}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{F-1} \sum_{b=0}^{F-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^l} w_{ab} \quad (5.3)$$

Potrebno je napomenuti kako izraz (5.3) ima smisla samo za pozicije i i j koje su udaljene za barem F od početne pozicije, jer se u suprotnom pojavljuju negativni indeksi.

5.1.1. Programska izvedba

Primjerom koda 5.1 prikazana je programska izvedba propagacije greške u konvolucijskom sloju.

Primjer koda 5.1: Propagacija greške u konvolucijskom sloju

```

1 public List<double[][]> backpropagate() {
2     int sw = inputData.getWidth();
3     int sh = inputData.getHeight();
4     List<double[][]> previousFunError = initFunError(sw, sh);
5     for (int n = 0; n < filters; n++) {
6         for (int i = 0, tempi = 0; tempi <= sw - sx; tempi +=
            stride, i++) {
7             for (int j = 0, tempj = 0; tempj <= sh - sx; tempj +=
                stride, j++) {
8                 double chainGradient = functionError.get(n)[i][j]
                    * Util.sigmoidDerivative(rawResult.get(n)[i][j]);
9                 for (int a = 0; a < sx; a++) {
10                     for (int b = 0; b < sx; b++) {
11                         for (int m = 0; m < inputDepth; m++) {
12                             allWeightErrors.get(n*inputDepth+m)[a][b]
                                += inputData.get(m)[i + a][j + b] *
                                    chainGradient;

```

```

13             previousFunError.get(m)[i+a][j+b] +=
                allWeights.get(n * inputDepth +
                m)[a][b] * chainGradient;
14         }
15     }
16 }
17     biasesErrors[n] += chainGradient;
18 }
19 }
20 }
21     return previousFunError;
22 }

```

U liniji 5 počinje iteriranje po svim filterima. Zatim u liniji 6 i 7 iterira se po svim pozicijama na kojima se može nalaziti filter. U liniji 8 dohvaća se propagirana greška sljedećeg sloja i množi s derivacijom aktivacijske funkcije od izlaza konvolucijskog sloja u unaprijednoj fazi prije djelovanja aktivacijske funkcije. Zatim u linijama 9 i 10 iterira se po svim pozicijama unutar filtra, dok se u liniji 11 iterira po svim komponentama ulaza. U liniji 12 računa se parcijalna derivacija greške po težinama, a u liniji 13 računa se greška prethodnog sloja koja će biti propagirana.

5.2. Sloj sažimanja maksimalnom vrijednosti

U ovome radu korišteno je samo sažimanje maksimalnom vrijednosti, pa će se samo za njega pokazati propagacija pogreške. Kod unazadne propagacije slojeva sažimanja potrebno je propagirati grešku manje mape trenutnog sloja u veću mapu prethodnog sloja. Propagiranje pogreške u sloju sažimanja jednostavnije je nego u konvolucijskom sloju. Kako u unaprijednoj fazi od grupiranih vrijednosti uzimamo samo maksimalnu to znači da ostale vrijednosti ne mogu imati utjecaj na grešku. Iz tog razloga vrijednost greške propagira se na mjesto gdje se nalazila maksimalna vrijednost, dok se na ostale pozicije postavlja vrijednost 0. Zbog toga je u unaprijednoj fazi bilo potrebno pamtit pozicije, kako bi u unazadnoj fazi ispravno propagirati grešku.

5.2.1. Programska izvedba

Primjerom koda 5.2 prikazana je propagacija pogreške u sloju sažimanja maksimalnom vrijednosti.

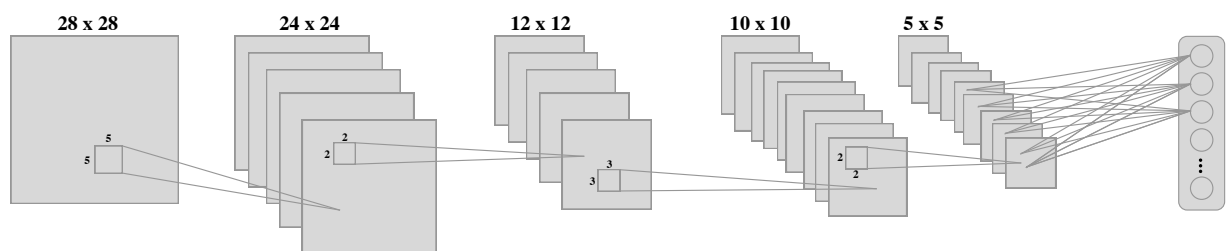
Primjer koda 5.2: Propagacija greške u sloju sažimanja maksimalnom vrijednosti

```
1 public List<double[][]> backpropagate() {
2     List<double[][]> results = new LinkedList<>();
3     int size = allInputs.getWidth();
4     for (int k = 0, n=allInputs.size(); k < n; k++) {
5         double[][] result = new double[size][size];
6         for (int i = 0, tempi=0; tempi <= size-sx; tempi+=sx,
7             i++) {
8             for (int j = 0, tempj=0; tempj <= size-sx;
9                 tempj+=sx, j++) {
10                Node loc = maxLocations.get(k)[i][j];
11                result[i+loc.x][j+loc.y]=functionError.get(k)[i][j];
12            }
13        }
14        results.add(result);
15    }
16    return results;
17 }
```

U liniji 4 počinje iteriranje po svim komponentama ulaza. Zatim u linijama 6 i 7 iterira se po svim pozicijama na kojima se filter može nalaziti. Iz strukture `maxLocation` dohvaća se pozicija na kojoj se nalazila maksimalna vrijednosti, te se na tu lokaciju propagira odgovarajuća greška.

5.3. Primjer arhitekture konvolucijske mreže

Nakon što su opisane glavne komponente konvolucijske mreže na sljedećem primjeru bit će pokazana jedna arhitektura konvolucijske mreže. Ulaz u konvolucijsku mrežu u ovome primjeru je monokromatska slika (crno-bijela slika) veličine 28x28 točaka.



Slika 5.1: Primjer arhitekture konvolucijske mreže

Prvi sloj u ovoj mreži je konvolucijski sloj s kvadratnim filtrom veličine 5 točaka. U ovome sloju filter se u svakom koraku pomiče za jednu točku. Veličina sloja računa se prema izrazu (4.1), stoga će veličina prvog konvolucijskog sloja biti: $\frac{(28 - 5)}{1} + 1 = 24$. Potrebno je definirati i broj filtera u prvom konvolucijskom sloju, neka broj filtera bude 6. Svaki filter sastoji se od $5 * 5 + 1 = 26$ težina, odnosno jedna težina za svaku poziciju unutar filtra i jedna dodatna težina za prag. Kako u prvom konvolucijskom sloju postoji 6 filtera ukupan broj težina u prvom konvolucijskom sloju je $26 * 6 = 156$.

Sljedeći sloj je sloj sažimanja maksimalnom vrijednosti. Veličina kvadratnog filtra je 2, dok se filter u svakom koraku pomiče za 2 kako ne bi bilo preklapanja. Sada je moguće odrediti veličinu izlaza prvog sloja sažimanja. Njegova veličina prepolovit će se na pola, pa je izlaz sloja veličine 12x12 točaka. Broj komponenata ostaje isti pa je dubina ovoga sloja jednaka 6. U sloju sažimanja maksimalnom vrijednosti ne postoje težine, no potrebno je pamtit pozicije na kojima su se nalazile maksimalne vrijednosti.

Sljedeći sloj je ponovno konvolucijski sloj. Veličina filtra je 3, dok se filter u svakome koraku kao i ranije pomiče za jednu točku. Veličina drugog sloja računa se kao: $\frac{(12 - 3)}{1} + 1 = 10$. Ponovno je potrebno definirati broj filtera u sloju, neka to za ovaj primjer bude 10. Svaki filter sastoji se od matrice težina veličine 3x3 za svaku komponentu ulaza. Kako se ulaz sastoji od 10 komponenata svaki filter sastoji se od $6 * 3 * 3 + 1 = 55$ težine. Ukupan broj težina u ovome sloju je $10 * 55 = 550$.

Sljedeći sloj je ponovno sloj sažimanja maksimalnom vrijednosti. Kao i prošlom veličina kvadratnog filtra je 2, dok se filter u svakom koraku pomiče za 2. Veličina izlaza ponovno se prepolavlja, pa je izlaz iz ovog sloja veličine 5x5 točaka. Broj komponenata ostaje isti i ne postoji niti jedna težina u sloju.

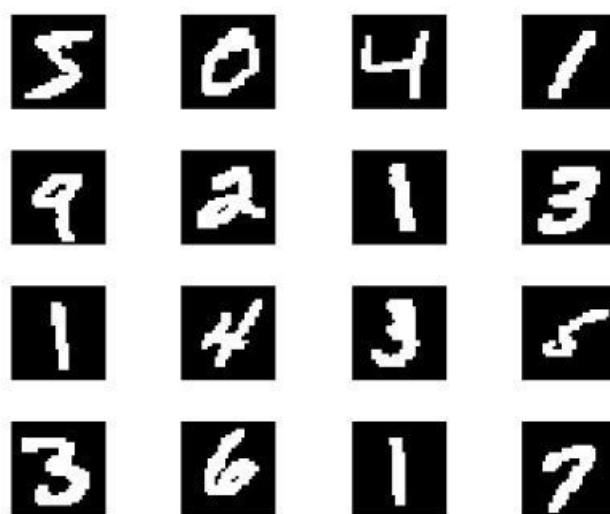
Posljednji sloj je potpuno povezani sloj. Ako želimo prepoznavati brojke sa slike dobivene na ulazu, najjednostavnije je posljednji sloj postaviti 10 neurona odnosno jedan izlaz za svaku brojku. Kako su neuroni potpuno povezani s prethodnim slojem, svaki neuron povezan je sa svakom vrijednosti prethodnog sloja. Predzadnji sloj sastoji se od 10 komponenata, dok unutar svake komponente nalazi se mapa veličine 5x5, stoga svaki neuron potpuno povezanog sloja sastoji se od $10 * 5 * 5 + 1 = 251$ težina. Kako se u posljednjem sloju nalazi 10 neurona ukupan broj težina je: $10 * 251 = 2510$.

6. Primjer konvolucijske mreže

U ovome poglavlju bit će opisano nekoliko konkretnih implementacija konvolucijske neuronske mreže. Izrađene konvolucijske mreže kao zadatak imaju prepoznati rukom napisane brojeve. Baza slika rukom napisanih brojeva koja je korištena pri učenju mreže je baza MNIST. Kroz ovo poglavlje najprije će biti objašnjen skup podataka koji je korišten za učenje mreže, a zatim i karakteristike mreža koje su korištene.

6.1. Baza MNIST

Skup podataka koji je korišten za učenje konvolucijske neuronske mreže u ovome radu je skup pod imenom MNIST (*Mixed National Institute of Standards and Technology*) baza. Korišteni skup podataka jedan je od najvećih skupova rukom napisanih brojeva. Skup se sastoji od 60 000 primjera slika za učenje i 10 000 primjera za testiranje. Svaki primjer zapravo je monokromatska slika veličine 28x28 slikovnih elemenata (engl. *pixels*). Na slici 6.1 prikazano je nekoliko primjera koji se nalaze u bazi MNIST.



Slika 6.1: Nekoliko primjera iz MNIST baze

6.2. Rezultati

U ovome dijelu rada prikazat će se rezultati učenja četiri konvolucijske mreže. Prva mreža koja će biti objašnjena je konvolucijska mreža s jednostavnom arhitekturom. U drugom eksperimentu korištena je složenija mreža s više filtara. U trećem eksperimentu korištena je ista mreža kao i u drugom eksperimentu, no stopa učenja smanjivala se u svakoj novoj epohi. U posljednjem eksperimentu korištena je složenija arhitektura neuronske mreže s više epoha učenja.

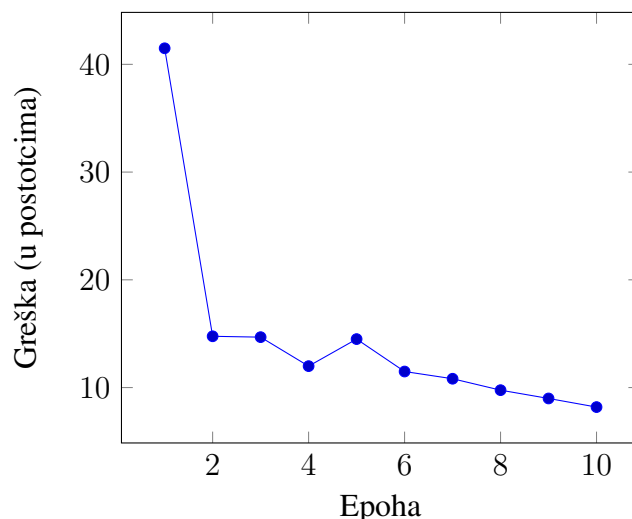
6.2.1. Jednostavna konvolucijska mreža

U ovome primjeru korištena je jednostavna arhitektura konvolucijske neuronske mreže. Mreža se sastojala od 6 slojeva. Unutar svih konvolucijskih i potpuno povezanih slojeva korištena je sigmoidalna aktivacijska funkcija. Tablicom 6.1 definirana je arhitektura korištene mreže. Kao sloj sažimanja korištena je objašnjena metoda sažimanja maksimalnom vrijednosti (engl. *max-pooling*).

Vrsta sloja	Veličina filtra	Broj filtara	Pomak filtra	Broj neurona
Konvolucijski	5	2	1	/
Max-pooling	2	2	2	/
Konvolucijski	4	5	1	/
Max-pooling	3	5	3	/
Potpuno povezani	/	/	/	7
Potpuno povezani	/	/	/	10

Tablica 6.1: Karakteristike mreže

Mreža je učena na slikama iz baze MNIST. Korišteno je ukupno 60000 slika iz baze tako da je 70% slika korišteno za učenje mreže, dok je preostalih 30% slika korišteno za testiranje mreže. Mreža je učena s 10 prolaza kroz sve slike iz skupa za učenje (10 epoha). Na slici 6.2 prikazan je odnos greške kroz epohe. Nakon 10 epoha greška na slikama iz skupa za učenje bila je 8.2%, dok je na slikama iz skupa za testiranje pogreška iznosila 7.7%. Stopa učenja korištena u ovome primjeru iznosila je 0.1 i nije se mijenjala tijekom učenja.



Slika 6.2: Greška kroz epohe

6.2.2. Primjer složenije arhitekture

U ovome primjeru korištena je nešto složenija arhitektura mreže. Mreža se sastoji, kao i prošlom primjeru, od 6 slojeva. Unutar svih konvolucijskih i potpuno povezanih slojeva korištena je sigmoidalna aktivacijska funkcija. Kao i u prošlom primjeru korišten je sloj sažimanja maksimalnom vrijednosti. Tablicom 6.2 prikazana je arhitektura korištene mreže. Skup za učenje i treniranje potpuno je jednak prethodnom primjeru, kao i stopa učenja.

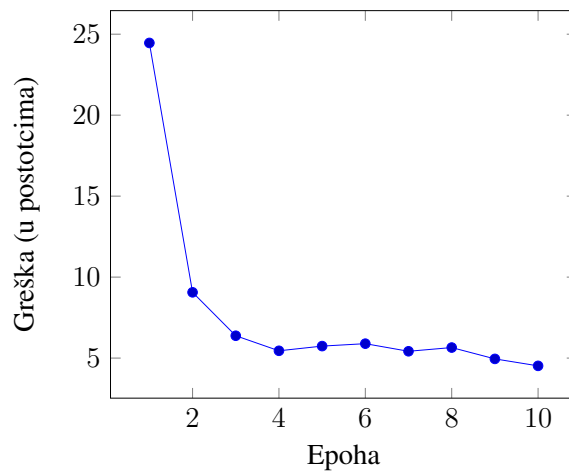
Na slici 6.3 prikazan je odnos greške kroz epohe. Nakon 10 epoha učenja greška na skupu podataka za učenje iznosila je 4.52%, dok je na testnom skupu podataka greška iznosila 4.50%.

Vrsta sloja	Veličina filtra	Broj filtara	Pomak filtra	Broj neurona
Konvolucijski	5	5	1	/
Max-pooling	2	5	2	/
Konvolucijski	4	6	1	/
Max-pooling	3	6	3	/
Potpuno povezani	/	/	/	15
Potpuno povezani	/	/	/	10

Tablica 6.2: Karakteristike mreže

Ako usporedimo rezultate ove mreže s rezultatima mreže iz prošlog primjera, može se zaključiti kako mreža sa složenijom arhitekturom daje bolje rezultate. Dobiveni

rezultat je i očekivan jer se složenijom arhitekturom, odnosno s više filtara, mreža može prepoznati više značajki sa slike koje mogu pomoći u klasifikaciji brojke.

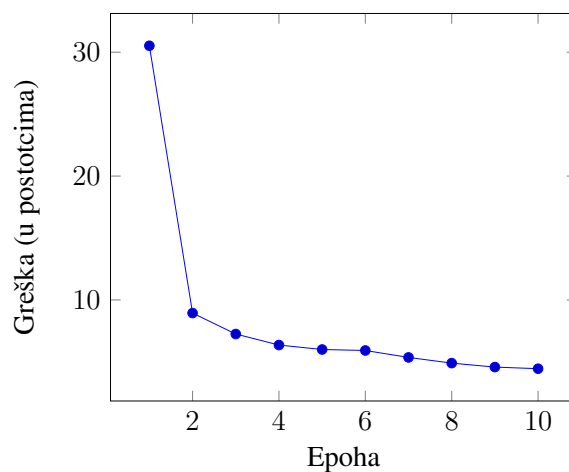


Slika 6.3: Greška kroz epohe

6.2.3. Primjer s promjenjivom stopom učenja

U ovome primjeru korištena je potpuno jednaka mreža kao i u prošlom primjeru. Svi parametri učenja također su potpuno jednaki kao i u prošlom primjeru, osim stope učenja. U ovome primjeru stopa učenja se kroz svaku epohu smanjivala za 10%.

Na slici 6.4 prikazan je odnos greške kroz epohe. Nakon 10 epoha učenja greška na slikama iz skupa za učenje iznosila je 4.45%, dok je na testnom skupu podataka greška iznosila 4.69%. Iz rezultata je vidljivo kako smanjivanje stope učenja u ovome primjeru nije imalo velik utjecaj na grešku.

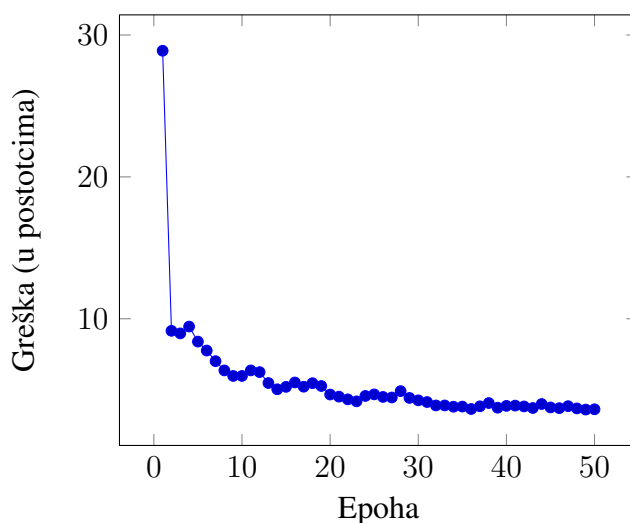


Slika 6.4: Greška kroz epohe

6.2.4. Primjer s više epoha

U ovome primjeru korištena je jednaka mreža kao i u prošlom primjeru. Početna stopa učenja iznosila je 0.1, dok se nakon svake epohe stopa učenja smanjivala za 0.7%. Mreža je učena i testirana na istom skupu podataka, ali je za razliku od prošlog primjera učena kroz 50 epoha.

Na slici 6.5 prikazan je odnos greške kroz epohe. Nakon 50 epoha učenja greška na slikama iz skupa za učenje iznosila je 3.61%, dok je na testnom skupu podataka greška iznosila 4.53%.



Slika 6.5: Greška kroz epohe

Iako je pogreška u ovome primjeru prilično mala u nekim radovima [4] dobivena je pogreška ispod jedan posto. No u ovome primjeru mreža je učena samo 50 epoha na manjem skupu podataka (cijeli skup podijeljen je na 2 dijela: dio za učenje i dio za testiranje). Također u ovome radu nisu korištene afine transformacije (rotacija, skaliranje, smik) nad slikama.

7. Zaključak

Kroz ovaj rad najprije su opisane osnovne značajke neuronskih mreža. Zatim je detaljno opisan rad konvolucijske neuronske mreže. Unutar rada proučene su specifičnosti konvolucijskih neuronskih mreža i njenih slojeva. Također osim rada konvolucijske neuronske mreže opisan je i algoritam unazadne propagacije kojim se težine unutar mreže uče. Uz rad također je razvijena i implementacija konvolucijske neuronske mreže u programskom jeziku *Java*. Mreža je razvijena bez korištenja vanjskih biblioteka.

Implementirane mreže i algoritam učenja testirani su na poznatom skupu rukom napisanih brojeva MNIST. Cilj mreže bio je točno klasificirati rukom napisani broj sa slike. U radu je prikazano nekoliko različitih mreža s kojima se pokušao riješiti navedeni problem. Rezultati su bili različiti u ovisnosti od arhitekture mreže i parametara algoritma učenja. Najmanja greška na skupu za učenje iznosila je 3.61%, dok je greška s istom mrežom na testnom skupu iznosila 4.53%.

U daljnjem radu preporučuje se treniranje mreže na cijelom skupu za učenje, kao i optimizacija parametara u algoritmu učenja. Također preporučuje se testiranje mreže s drugačijom arhitekturom, aktivacijskim funkcijama i funkcijama greške. U svrhu bolje generalizacije preporučuje se i uvođenje afinih transformacija (rotacija, skaliranje, smik) na slike iz skupa za učenje i testiranje.

LITERATURA

- [1] Jan Šnajder Bojana Dalbelo Bašić, Marko Čupić. *Umjetne neuronske mreže*. Zavod za elektroniku, mikroelektroniku i inteligentne sustave, Fakultet elektrotehnike i računarstva, 2008.
- [2] Yann LeCun. The MNIST database of handwritten digits. URL <http://yann.lecun.com/exdb/mnist/>. Pristupano: 2016-05-26.
- [3] Yann LeCun, Léon Bottou, Yoshua Bengio, i Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [4] Patrice Y Simard, Dave Steinkraus, i John C Platt. Best practices for convolutional neural networks applied to visual document analysis. IEEE, 2003.
- [5] Vedran Vukotić. Raspoznavanje objekata dubokim neuronskim mrežama. 2014. Diplomski rad.

Konvolucijske neuronske mreže

Sažetak

U ovome radu opisane su osnovne značajke svake neuronske mreže. Zatim je opisan rad konvolucijskih neuronskih mreža i pripadnog algoritma za učenje mreže. Uz rad izrađen je i programski kod koji omogućava izradu različitih konvolucijskih neuronskih mreža i njihovo učenje i testiranje. U radu su prikazana četiri primjera rada konvolucijskih mreža učenih i testiranih na bazi rukom napisanih brojeva MNIST. Na samome kraju opisani su i prikazani dobiveni rezultati.

Ključne riječi: umjetne neuronske mreže, duboke neuronske mreže, konvolucijske neuronske mreže, klasifikacija, algoritam unazadne propagacije, MNIST, računalni vid

Title

Abstract

This work describes the basic features of every neural network. This work then describes features of convolution neural networks and its respective network learning algorithm. Alongside this work, a program code is made which enables the creation of various convolution neural networks as well as their learning and testing. This work also contains four different examples of convolutional networks which are learned and tested on hand-written numbers from MNIST database. The final part of the work shows given results.

Keywords: artificial neural networks, deep neural networks, convolutional neural networks, classification, backpropagation algorithm, MNIST, computer vision