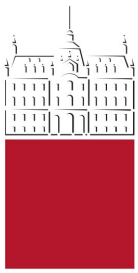
Univerza *v Ljubljani*
Fakulteta *za matematiko in fiziko*



Nevronske mreže

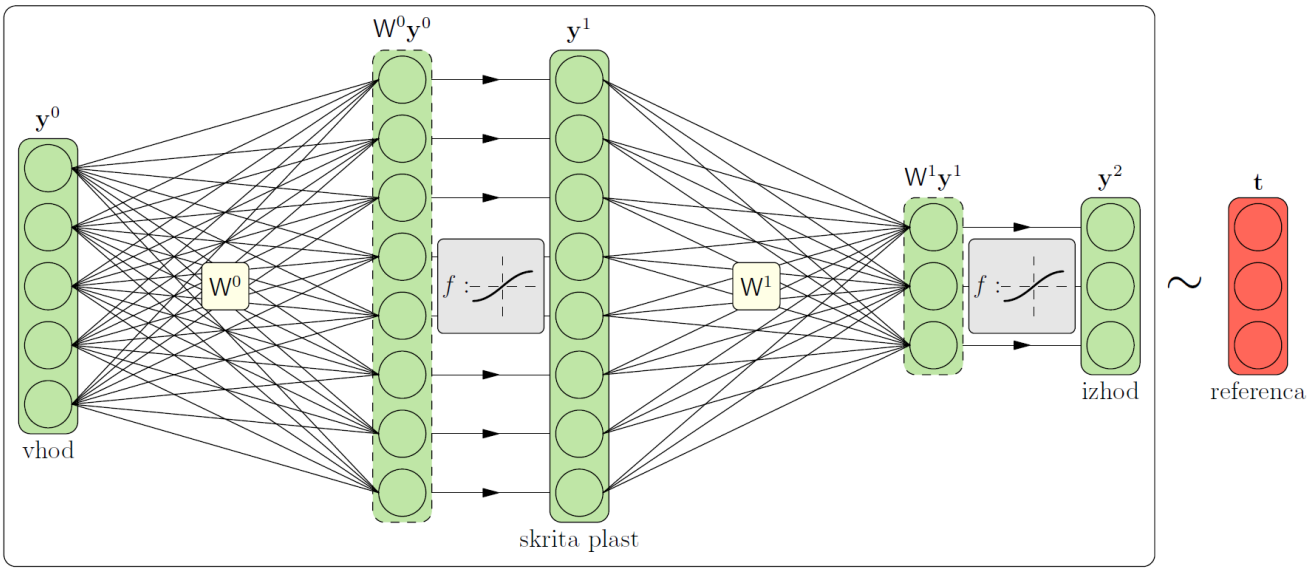
Avtor: Simon Perovnik
Predavatelj: prof. dr. Simon Širca
Asistent: doc. dr. Miha Mihovilovič

Trinajsta naloga pri Modelski analizi I

Ljubljana, avgust 2022

1 Teoretični uvod

Modeli nevronskih mrež so verige preslikav med vektorji y^l (l-ta "plast" nevronov) v obliki $y^{l+1} = f(W^l y^l)$, kjer so W^l linearne preslikave (v splošnem nekvadratne matrike), f pa je nelinearna aktivacijska funkcija, ki deluje na vektor po komponentah. Nevronske mreže so tip univerzalnih funkcij, ki v principu lahko predstavljajo poljubno funkcijsko zvezo med vhodnim vektorjem y^0 (običajno veliko komponent) in izhodnim vektorjem y^L (običajno malo komponent), kjer je L število preslikav med plastmi. Aktivacijska funkcija je običajno sigmoidna funkcija, ki poreže signal na interval $(-1, 1)$, v našem primeru bomo vzeli $f(x) = \tanh(x)$. Glede na namen ločimo različne vrste nevronskih mrež, ki se razlikujejo po številu skritih plasti (in s tem številu matrik z utežmi), številu nevronov v skritih plasteh (dimenzije vektorjev), ter načina povezav (ali so matrike polne, ali morda bločne glede na izbrano lokalno povezanost, kot v primeru konvolucijskih nevronskih mrež). Nevronske mreže se pogosto uporabljajo za klasifikacijo, kjer izhodne komponente interpretiramo kot odgovore da/ne o pripadnosti izbranim kategorijam. Vsa zapletena hevristika odločanja se skriva v koeficientih utežnih matrik. "Deep learning" mreže se sklicujejo na modele z večjim številom skritih plasti, ki hierarhično združujejo prepoznane detajle v vedno kompleksnejše pojme. Izvrednotenje rezultata nevronske mreže je trivialna operacija, zahtevni del je inverzni problem – določitev parametrov W^l nevronske mreže – saj poznamo le željeni rezultat za nekaj primerov vhoda. Ta problem rešujemo s treniranjem nevronske mreže na veliki količini vhodnih vektorjev y^0 z znanim izidom t. Primer take mreže z eno skrito plastjo je prikazan spodaj:



Slika 1: Nevronska mreža z eno skrito plastjo.

Odstopanje med znanim izidom in rezultatom nevronske mreže, definirano s funkcijo $E = \frac{1}{2}||y^L - t||^2$ ("loss function"), minimiziramo z vzvratno propagacijo napake, ki je poseben primer minimizacije z gradientnim spustom. Definirajmo residual, ki se rekurzivno propagira z zadnje plasti nevronov proti prvi,

$$\begin{aligned} \delta^{L-1} &= -(y^L - t) \odot f'(W^{L-1} y^{L-1}) \\ \delta^l &= \delta^{l+1} W^{l+1} \odot f'(W^l y^l), \end{aligned} \tag{1}$$

kjer $\odot$ predstavlja množenje istoležnih komponent. Ker velja $\frac{d}{dx}\tanh(x) = 1 - \tanh(x)^2$, lahko zmanjšamo število računskih operacij, saj smo vse $\tanh(W^l y^l)$ že izračunali med izvrednotenjem mreže od leve proti desni. Popravek matrikam uteži, ki ga izvedemo na vsakem učnem primeru, potem zapišemo kot

$$W^l_{ij} \mapsto W^l_{ij} + \eta \delta^l_i y^l_j \tag{2}$$

pri čemer je η hitrost učenja (korak gradientnega spusta). Primerna vrednost je na primer $\eta = 0.01$. Začetne uteži W^l so običajno matrike Gaussovih naključnih števil z disperzijo $\sigma_l^2 = 2/\dim(y^l)$.

2 Nevronske mreže

2.1 Naloga

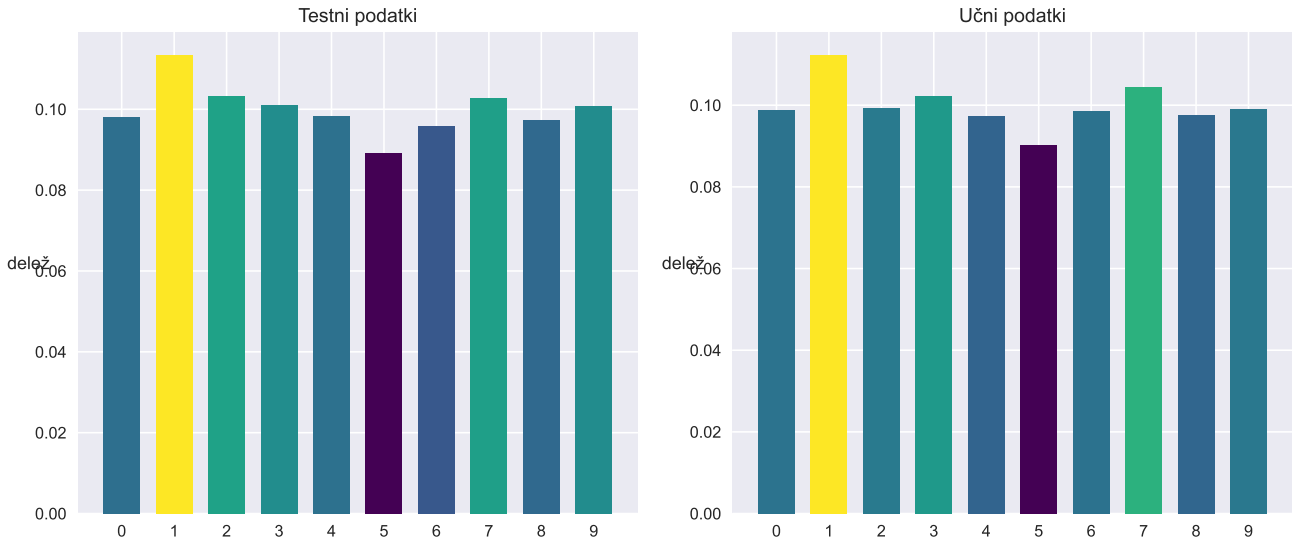
V datotekah **train-images-idx3-ubyte** in **train-labels-idx1-ubyte** najdeš nabor črnobelih slik ročno napisanih števk od 0 do 9 s pripadajočimi oznakami 0 - 9. Skonstruiraj in nauči nevronsko mrežo s tremi plastmi – na prvo plast pripeljemo vhodne podatke ($28^2 = 784$ pikslov slike, reskaliranih na interval med 0 in 1), skrita plast je lahko kar enake dimenzije kot prva, izhodna plast pa predstavlja 10 izhodov, za vsako števko enega, katerih ciljne vrednosti so 1 za pravo števko in -1 za napačne. Poskusiš lahko tudi z različnimi dimenzijami skritih plasti ali za več skritih plasti, ter z različnimi hitrostmi učenja. Tekom učenja spremljaj napako E ter delež pravilno določenih števk. Nekaj podatkov (npr. desetino) prihrani za testiranje po koncu učenja. Katere števke algoritem največkrat zamenjuje med sabo in katere najslabše določi? Bi sami naredili isto napako? Nevronska mreža se lahko odziva nepredvidljivo, če ji ponudimo podatke, ki močno odstopajo od podatkov, na katerih je bila mreža trenirana. Kakšen izid dobimo, če na vhod pošljemo slike, ki ne predstavljajo števk?

2.2 Analiza podatkov

Preden začnemo s treniranjem mreže, se spoznajmo s setom podatkov, ki jih bomo uporabili. Gre za dobro poznan set slik MNIST, do katerega najbolj enostavno dostopamo preko Pythonove knjižnice **TensorFlow** in vgrajene funkcije TF.KERAS.DATASETS.MNIST.LOAD_DATA(). Poglejmo si, s kakšnimi slikami imamo pravzaprav opravka.

Slika 2: Primeri slik iz zbirke MNIST.

Hitro lahko opazimo, da naloga za našo mrežno ne bo prav enostavna, saj obstaja več načinov, kako lahko zapišemo neko števko. Dober primer tega je slika števila 2, ki je narisana z izrazito zanko, kar v splošnem predstavlja značilno karakteristiko števila 6. Pravtako se zdi, da bo problematična številka 4, saj ni nujno , da je na vrhu sklenjena, kar bi lahko zmedlo naš program. Številka 1 je zapisana zgolj kot ravna črta, kar bo za naš program dobro, saj je tako manj podobna številki 7.



Slika 3: Zastopanost posamezne števke v zbirki MNIST.

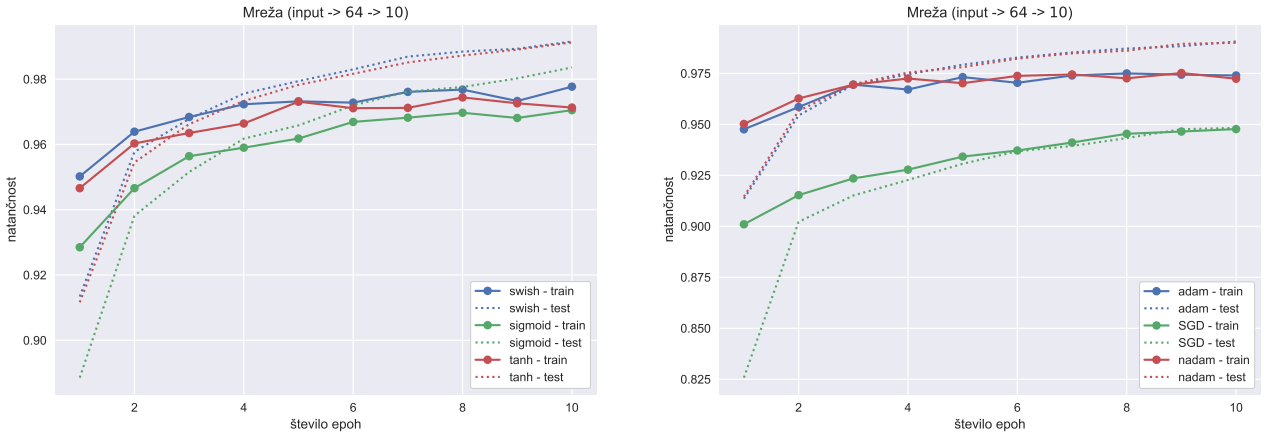
- V našem setu so številke številčno različno zastopane. Največ slik predstavlja enice, najmanj pa petke, kar je verjetno slučajno. Verjetno bi bilo smiselno, da bi bilo več slik namenjenim številkam, ki se jih težje ločiti med sabo.
- Zanimivo je, da testni in učni podatki med sabo nista uravnotežena, saj se histograma obeh setov podatkov med seboj ne ujemata povsem. Odstopanja niso velika (največje je pri številki 2, kjer znaša 4 promile.), je pa zanimivo, da sploh so, saj bi bilo to razmeroma enostavno popraviti. Verjetno lahko iz tega zaključimo, da majhna odstopanja niso problematična, saj sta set podatkov in njegova delitev zelo uveljavljena.

2.3 Sestavljanje modela

Pri konstruiranju nevronske mreže imamo veliko parametrov, ki jih lahko variiramo in s tem bistveno spreminjamo arhitekturo modela in seveda kvaliteto končnega rezultata. Spreminjamo lahko število skritih plasti (L), število nevronov v skritih plasteh (N), aktivacijske funkcije v posamezni plasti, optimizator modela, vrednost parametra hitrosti učenja (η), število epoh (kolikokrat se model uči na istem (le premešanem) setu slik), 'batch size', ipd. Ker je celoten model torej zelo kompleksen, ne bomo mogli v polnosti raziskati efektov vseh teh parametrov.

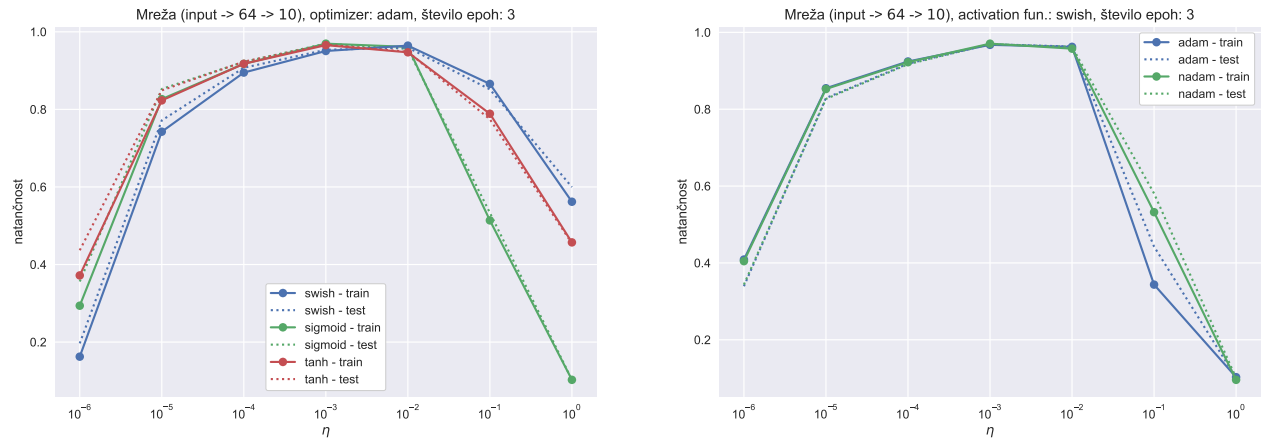
Model ustvarimo z ukazom TF.KERAS.MODELS.SEQUENTIAL(), kjer smo torej določili sekvenčno arhitekturo naše mreže. Prva plast naše mreže bo sprejela podatke v velikosti $28 \times 28 = 784$. Slike bodo na tej točki pretvorjene v en sam niz podatkov (torej ne bodo več v matrični obliki), kar storimo z ukazom TF.KERAS.LAYERS.FLATTEN(). Vsako naslednjo plast dodajamo z ukazom TF.KERAS.LAYERS.DENSE(), kjer lahko določimo poljubno število nevronov in poljubno aktivacijsko funkcijo. Zadnja plast mora vsebovati 10 vozlišč, saj model izbira med desetimi različnimi rezultati - desetimi števkami. Mrežo inicializiramo z ukazom MODEL.COMPILE(), kjer določimo optimizator, funkcijo izgube in metrike, ki jih želim spremljati tekom učenja. Model nato poženemo z ukazom MODEL.FIT, kjer podamo število epoh, 'batch size' in velikost validacijskega seta (v našem primeru bo vseskozi enak petini testnega seta).

Da bomo znali sestaviti model, ki bo kar se da dobro napovedal številke testnega seta, si oglejmo, kako nekaj izmed pomembnejših parametrov modela vpliva na rezultate. Pri tem se bomo omejili na spremljanje metrike natančnosti, torej uspešnosti pri napovedovanju cifer številk testnega seta. Vseskozi bo 'batch size' enak 32. Pravtako bo število epoh zaradi časovne potratnosti v splošnem zelo nizko.



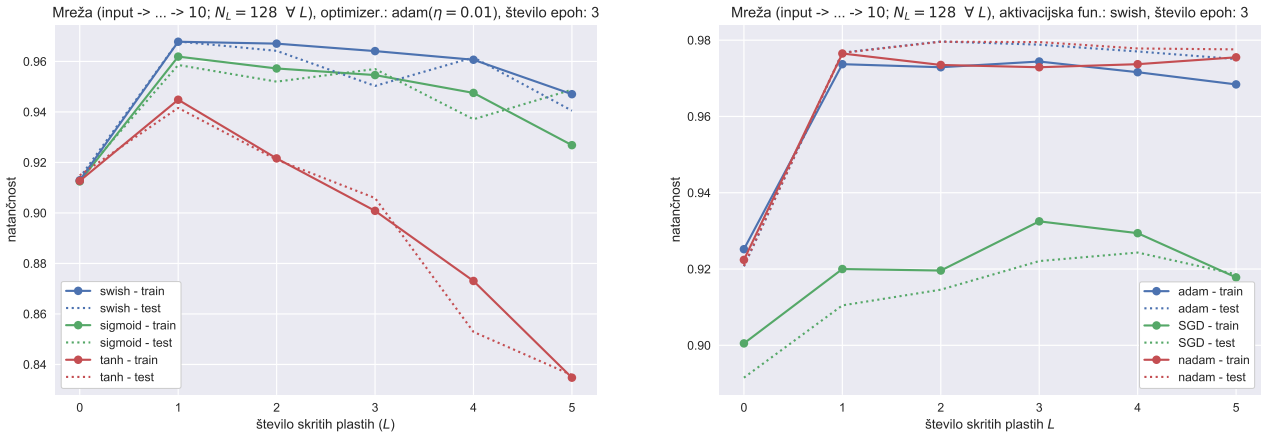
Slika 4: Vpliv števila epoh na natančnost napovedi.

Na grafih sta prikazana dva različna rezultata modela - natančnost modela na učnem setu in pa testnem. Slednja je boljša ocena za natančnost modela, saj ni podvržena 'over-fitting'-u podatkov. Ne glede na izbiro aktivacijske funkcije in optimizatorja se natančnost napovedi z vsako epoho izboljša. Izkaže se, da povečevanje z 10 na recimo 20 epoh ne prinese bistvene spremembe. Najboljše rezultate dobimo pri uporabi optimizatorja tipa 'adam' in aktivacijske funkcije 'swish', ki je definirana kot $\text{sigmoid}(x) \cdot x$.



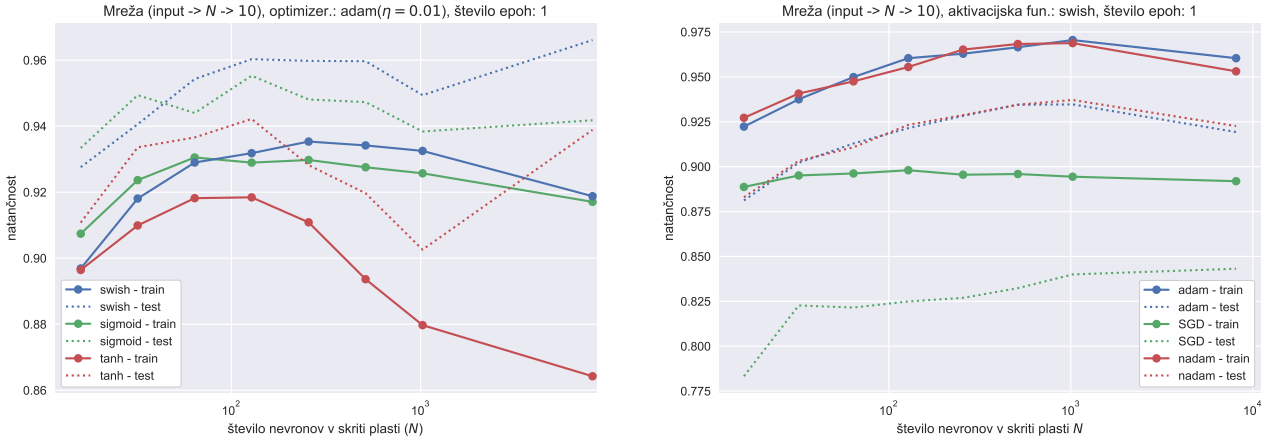
Slika 5: Vpliv hitrosti učenja na natančnost napovedi.

Pri optimizatorjih tipa 'adam' lahko določamo tudi hitrost učenja. Kot vidimo je optimalna vrednost med 0.01 in 0.001, kjer smo bili z vsemi optimizatorji in aktivacijskimi funkcijami približno enako uspešni. Opazimo zelo strm padec ob prevelikih hitrostih učenja. Iz primerjava obeh slik lahko vidimo, da je proces močno stohastične narave, saj dvakrat rišemo ob istih pogojih - z aktivacijsko funkcijo 'swish' in optimizatorjem 'adam'.



Slika 6: Vpliv števila skritih nivojev na natančnost napovedi.

Opazimo zelo različno obnašanje pri uporabi različnih aktivacijskih funkcij - funkcija tanh sunkovito pade če ima mreža več kot en skriti nivo. Kljub vsemu pa moramo biti previdni pri interpretaciji rezultatov, saj se zdi logično, da bodo mreže, ki so globlje in imajo več parametrov potrebovale več časa, da se dobro naučijo. Ker s povečevanjem L-ja, nismo povečevali tudi recimo števila epoh, ne moremo zares sklepati o tem, ali so globlje mreže res tako bistveno slabše. A ker si zaradi časovne potratnosti ne moremo privoščiti obširnejše analize, se bomo zadovoljili z rezultatom, da je en skriti nivo očitno boljši od nobenega.



Slika 7: Vpliv števila nevronov v skritem nivoju na natančnost napovedi.

Rezultati grafa so nekoliko podobni prejšnjim s slike (7) - spet opazimo izrazit padec pri uporabi funkcije tanh, medtem ko dobimo s sigmoidnima funkcijama boljše rezultate ob večjem številu nevronov. Ponovno pa moramo biti previdni z interpretacijo teh grafov, zaradi prej omenjenega razloga. Ponovno se potrdi tudi teza, da je za tovrsten primer optimizator tipa 'adam' ustreznejši od 'SGD'-ja.

Na podlagi vsega raziskanega lahko skonstruiramo mrežo, za katero imamo dobre razloga, da bo dobro prepoznavala slike zbirke MNIST. Mreža, ki jo bomo konstruirali bo imela dva skrita nivoja, oba s po 1024 nevroni in aktivacijsko funkcijo 'swish'. Ob tem bomo uporabili za učenje optimizator 'adam' s hitrostjo učenja $\eta = 0.001$ in desetimi epohami.

To zagotovo ni najboljša mreža, saj nismo v podrobnosti raziskali vseh parametrov in njihovih odvisnosti. Predvsem je ostalo veliko prostora za pregled različnih kombinacij parametrov - npr. več nivojem z različnimi aktivacijskimi funkcijami, kot tudi sledenje drugim metrik za vrednotenje uspešnosti naše mreže (do sedaj smo se

omejili zgolj na točnost napovedi).

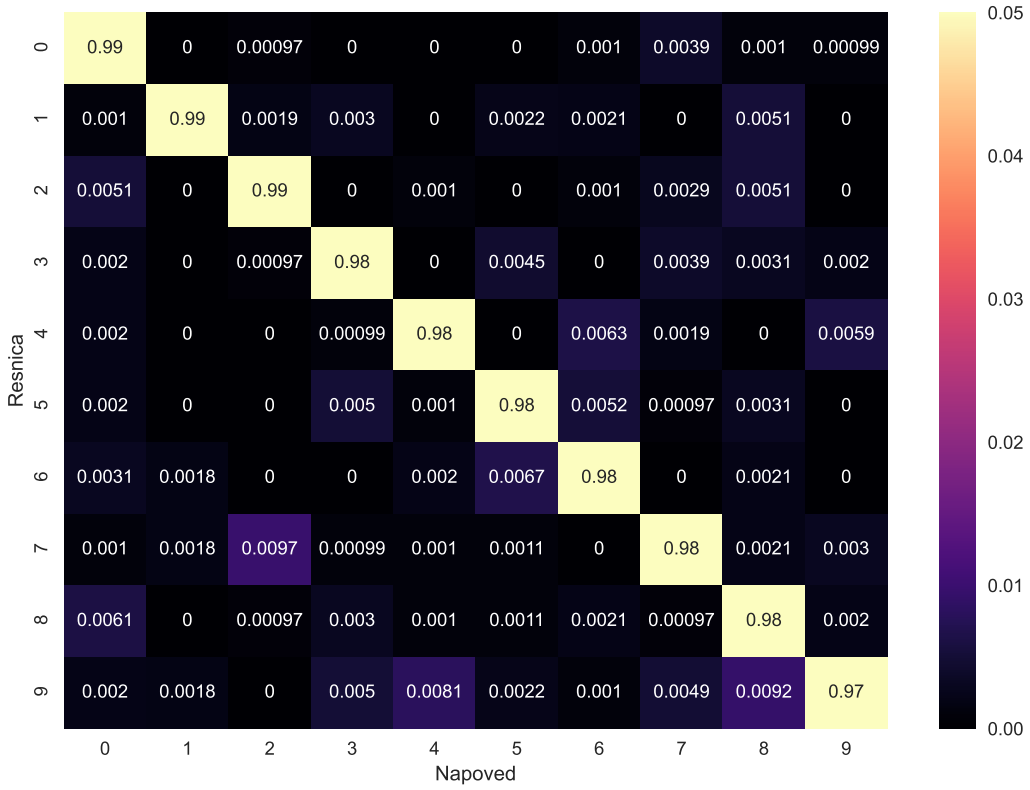
2.4 Analiza Rezultatov

Nevronska mreža, natrenirana na način opisan v prejšnjih odstavkih nam da naslednje rezultate.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 1024)              803840

 dense_1 (Dense)             (None, 1024)              1049600

 dense_2 (Dense)             (None, 10)                10250

=================================================================
Total params: 1,863,690
Trainable params: 1,863,690
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1875/1875 [==============================] - 62s 33ms/step - loss: 0.2140 - sparse_categorical_accuracy: 0.9353
Epoch 2/10
1875/1875 [==============================] - 73s 39ms/step - loss: 0.0979 - sparse_categorical_accuracy: 0.9698
Epoch 3/10
1875/1875 [==============================] - 56s 30ms/step - loss: 0.0668 - sparse_categorical_accuracy: 0.9789
Epoch 4/10
1875/1875 [==============================] - 56s 30ms/step - loss: 0.0486 - sparse_categorical_accuracy: 0.9850
Epoch 5/10
1875/1875 [==============================] - 60s 32ms/step - loss: 0.0404 - sparse_categorical_accuracy: 0.9876
Epoch 6/10
1875/1875 [==============================] - 54s 29ms/step - loss: 0.0323 - sparse_categorical_accuracy: 0.9899
Epoch 7/10
1875/1875 [==============================] - 47s 25ms/step - loss: 0.0267 - sparse_categorical_accuracy: 0.9921
Epoch 8/10
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0263 - sparse_categorical_accuracy: 0.9921
Epoch 9/10
1875/1875 [==============================] - 50s 27ms/step - loss: 0.0240 - sparse_categorical_accuracy: 0.9932
Epoch 10/10
1875/1875 [==============================] - 43s 23ms/step - loss: 0.0212 - sparse_categorical_accuracy: 0.9942
313/313 [==============================] - 3s 8ms/step - loss: 0.0869 - sparse_categorical_accuracy: 0.9825
313/313 [==============================] - 2s 7ms/step
```

Slika 8: Treniranje nevronske mreže.

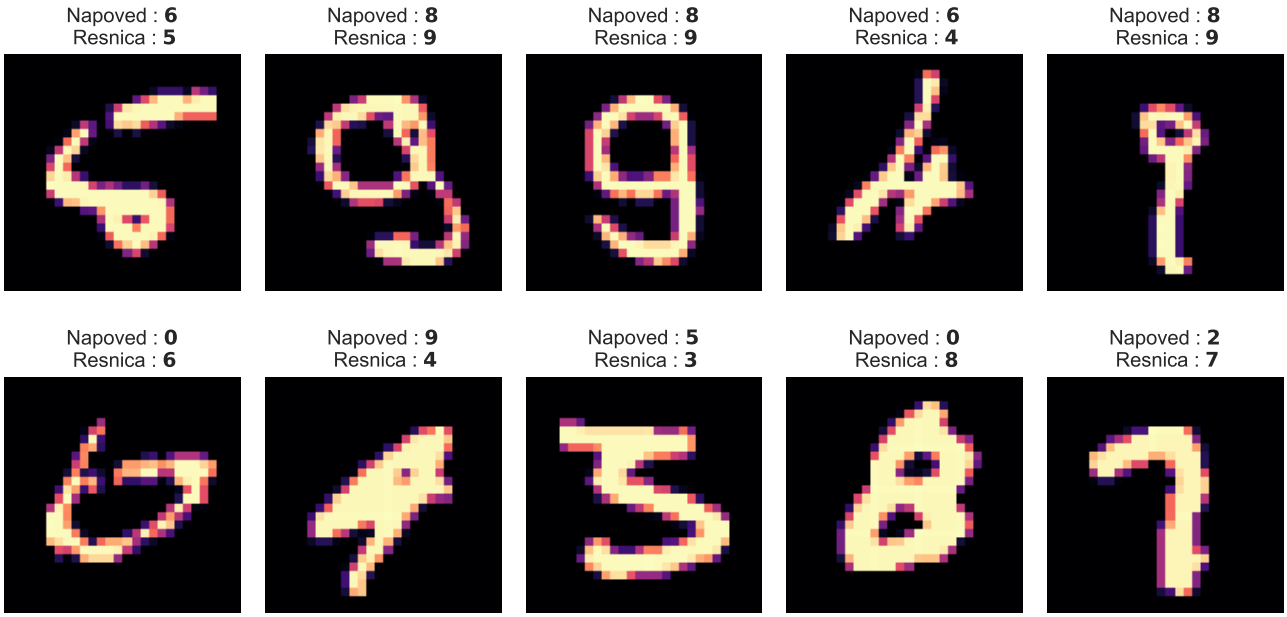
Mrežo smo učili deset epoh, kar je naneslo nekaj več kot 9 minut. Videti je, da se je natančnost mreže z vsako epoho izboljšala, kar pa je nekoliko varljivo, saj ob primerjavi z natančnostjo na testnem setu vidimo, da je očitno prišlo do 'over-fitting'-a. Mreža si je namreč že predobro poznala učni set podatkov - v ekstremu bi mreža pravilno prepoznala vse slike iz učnega seta, nato pa skoraj nobenega iz testnega seta, saj bi bila zelo prepričana, da pozna vse možne verzije števk, te ki jih je spoznala v učnem setu.



Slika 9: Matrika deležev pravilno in napačno razvrščenih slik.

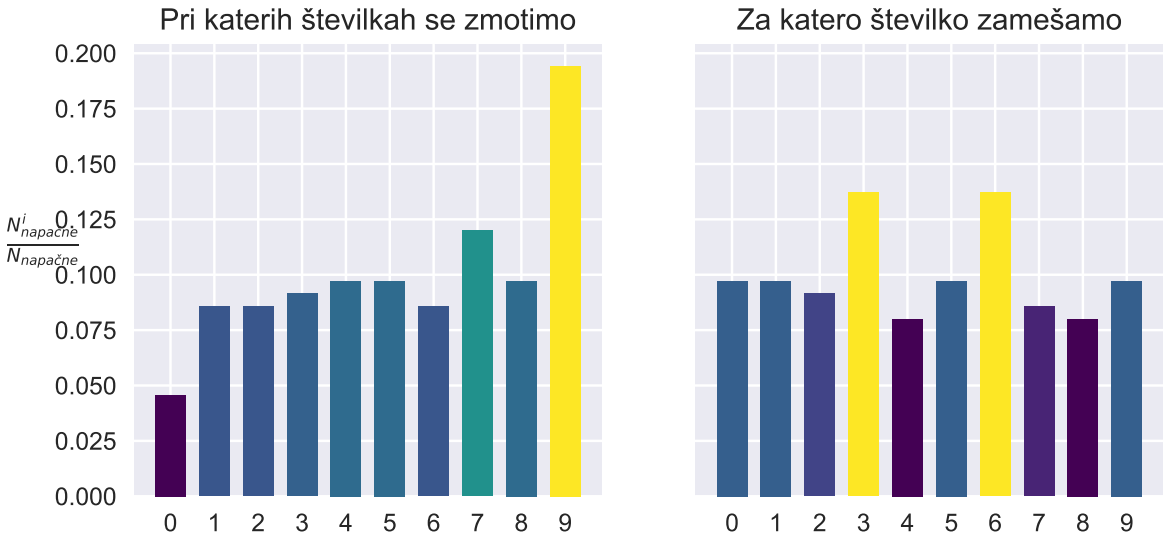
Poglejmo si t.i. 'confusion matrix' (CM), matriko, kjer so (v deležih), normirani na število slik posamezne števke v testnem setu, prikazani vsi rezultati klasifikacije slik. Matrika je močno diagonalna, kar pomeni, da je v splošnem dobro razvrščala slike. Opazimo lahko tudi nekaj najbolj tipičnih problemov, kjer se je mreža zmotila in napačno klasificirala števko. Treba je poudariti, da je nabor napačno razvrščenih slik zelo majhen in zato močno podvržen stohastični naravi klasifikacije z nevronskimi mrežami. Kljub vsemu so bile nekatere napake pogostejše:

- 7 zamenjamo z 2 - to se zdi zelo naravna napaka, saj je celotna oblika sedmice nekako zajeta v vsaki dvojki.
- 9 zamenjamo s 4 - ponovno zelo logična napaka, saj lahko štirka ob uporabi bolj zaobljenih črt hitro izpade kot devetka, sploh če je napisana na način, da je na vrhu sklenjena.
- 9 zamenjamo z 8.
- 6 zamenjamo s 5.



Slika 10: Nekaj primerov napačno razvrščenih slik.

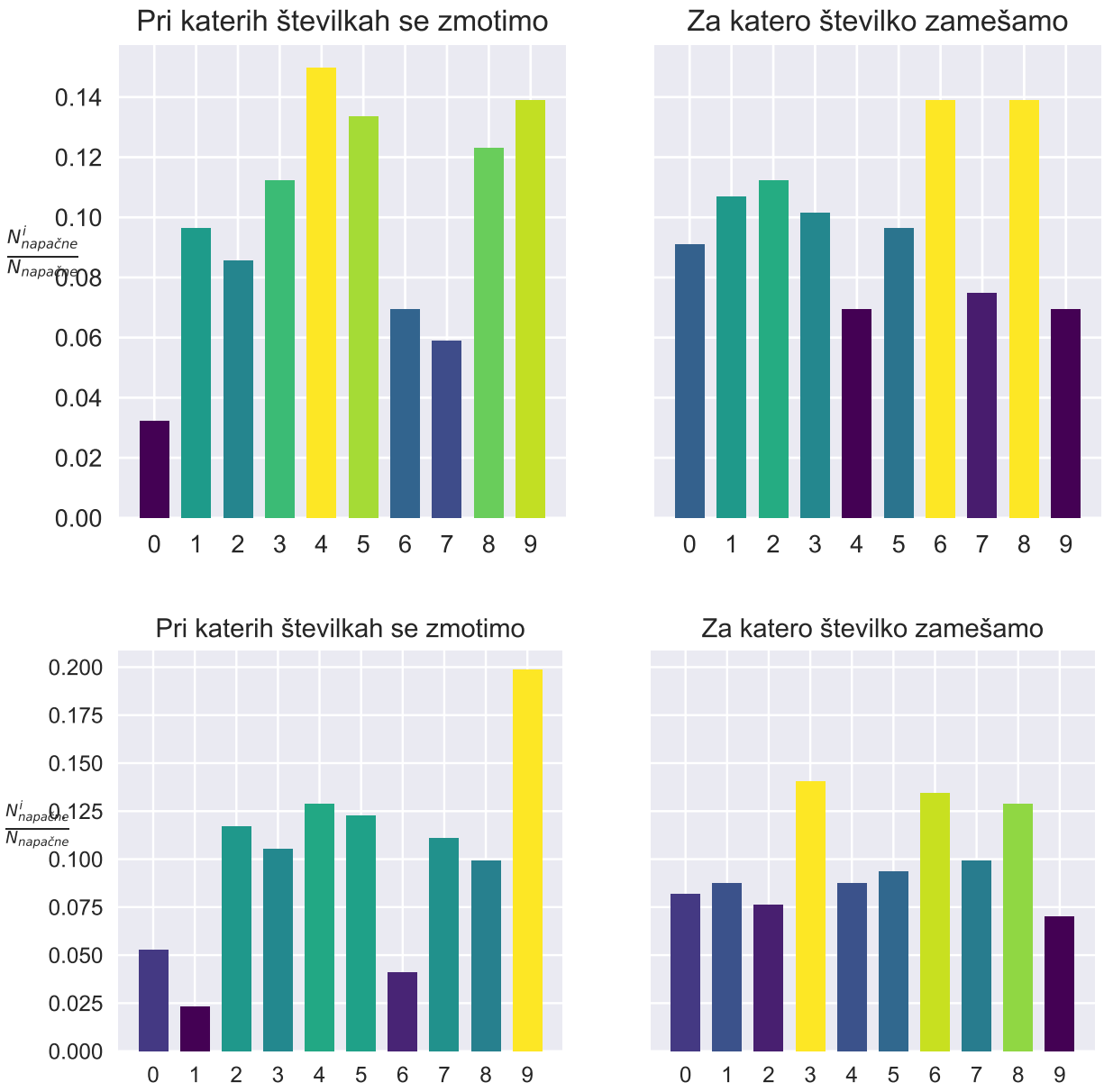
Ogledamo si lahko nekaj primerov slik, ki je mreža napačno klasificirala. Večino napak se zdi presenetljivo zelo 'človeških', oziroma vsaj jasno razumljivih, zakaj se je mreža zmotila. Prepoznamo lahko tudi nekaj vzorcev napak, ki smo jih napovedali pri analizi CM. Še najbolj nas preseneti napačno razvrščena devetka (tretja slika v zgornji vrsti), saj se zdi kot skoraj popolna devetica - morda pa je ravno to problem, da je bila preveč pravilno napisana, da bi jo mreža znala prav prepoznati, kar je zelo zanimiv efekt!



Slika 11: Podrobnejši pregled napačnih klasifikacij s pomočjo histogramov - na levi je prikazano, pri slikah katere številke se mreža največkrat zmoti, na desni pa so prikazani deleži številk, s katerimi mreža zameša pravo cifro.

- Največkrat smo napačno klasificirali sliko, na kateri je bila prikazana devetka.
- Mreža je precej dobro vedela, kakšne morajo biti $0, 1, 2, 4, 5$ in 8, saj jih v večji meri ni niti spregledala, niti ni zamešala kakšne druge števke za njih.
- Zanimivo je, da je mreža ob napačni napovedi največkrat zamešala števko za trojko ali šestko, česar ne znamo čisto dobro pojasniti. Morda imata obe takšno univerzalno obliko, da sta v mejnih primerih najbolj sigurna rešitev.

Kot smo izpostavil že prej, je set napačno razvrščenih slik zelo majhen, zato lahko ob ponovnih simulacijah enake mreže dobimo precej drugačne rezultate. Nekaj primerov je prikazanih spodaj.



Slika 12: Še nekaj primerov histogramov napačnih klasifikacij.

Rezultati se res kar bistveno spremenijo ob ponovnih treniranjih mreže. Kljub temu lahko prepoznamo nekaj

trendov - števko devet največkrat napačno klasificiramo, pogosto kakšno števko zamešamo s trojko in v splošnem dobro napovedujemo nekatere številke omenjene v prejšnjem odstavku.

3 Globoko sanjanje

3.1 Naloga

Pri učenju mreže smo minimizirali napako z iterativnim popravljanjem uteži W^l. Na že naučeni mreži pa lahko vprašamo, kateri vhod minimizira napako na izhodu za izbrani izhod. Z enakim postopkom posrednega odvajanja, s katerim smo določili popravke uteži, dobimo iteracijski popravek za vhodni vektor

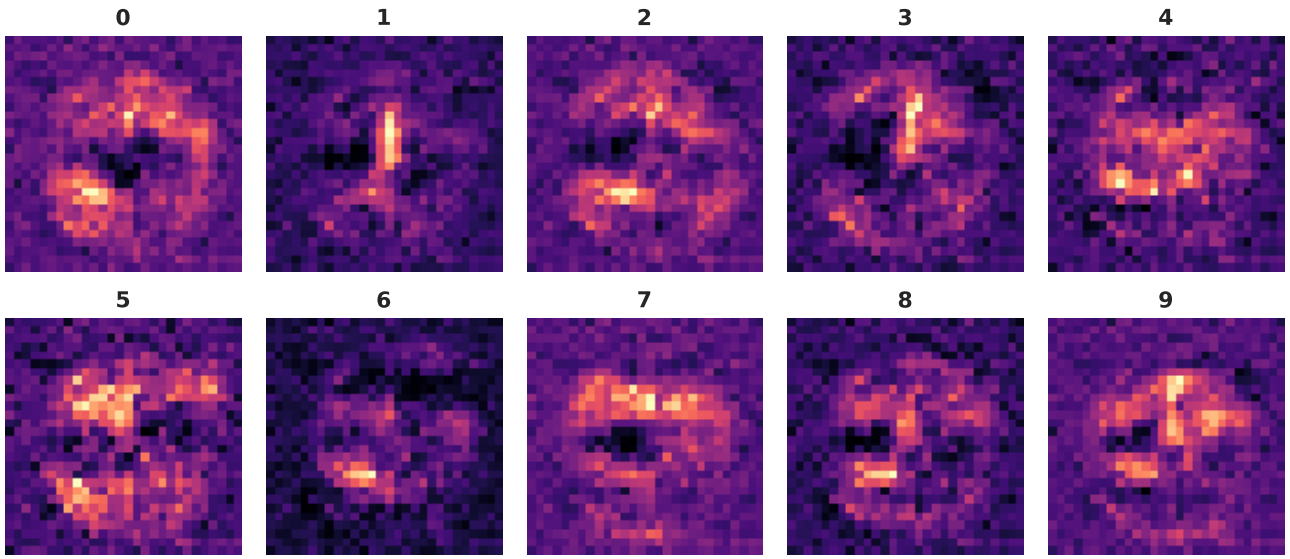
$$y_0 \mapsto y_0 + \eta \delta^0 W^0 \tag{3}$$

pri čemer do δ^0 pridemo s pomočjo rekurzije. To iteracijsko shemo ponavljamo, dokler napaka na izhodu ne doseže zadovoljivo majhne vrednosti. Začetni približek za sliko y_0 je v našem primeru lahko kar črna slika, na katero želimo, da nevronska mreža nariše izbrano števko glede na naučene značilnosti. Ta postopek je znan tudi pod frazo "deep dream" in ga lahko uporabimo kot vpogled v lastnosti, ki jih mreža zazna kot pomembne za klasifikacijo.

3.2 Implementacija in rezultati

V tem razdelku bomo celoten proces treniranja nevronskih mrež nekoliko obrnili na glavo. V naučeno mrežo bomo pošiljali slike in gledali njihovo napako na izhodu. Z iterativnim postopkom bomo iskali, kakšen vhod nam minimizira napako, kar predstavlja ekvivalent, kaj so v mreži najbolj intrinzične lastnosti posamezne števke. To seveda ne bodo idealne števke, pač pa bolj unija različnih konfiguracij posameznih števk. Podobno kot da bi se vprašali, kakšna je povprečna vrednost pikslov pri vseh slikah posamezne števke.

Izkaže se, da je sam postopek zelo slabo konvergira, tako da je potrebnega kar nekaj naprezanja in vztrajnosti pri iskanju optimalnih parametrov, za doseg vsaj nekoliko smiselnih rezultatov.



Slika 13: Globoko sanjane števke.

Izkaže se, da mreža razmišlja precej drugače kot ljudje - namesto osnovne oblike posamezne števke, prepozna dele, v katerih se posamezna števka najbolj razlikuje od ostalih in tako preko tega ve, s katero števko ima opravka. To je dobro razvidno iz slike ničle, kjer sta sta bistvena dela diagonalna okljuka, ki jih res ne srečamo pri nobeni

drugi števki. Podobno lahko razmišljamo tudi ob sliki štirici, ki je edina števka, ki ima vodoravno črtno na sredini - točno to karakteristiko prepozna mreža. Najbolj se človeška in 'mrežina' predstava o števkah ujemata pri enici, ki je hkrati tudi najbolj enostavna, in pa petki , kateri bi lahko iz zgornje slike tudi človek pravilno prepoznal. Vsekakor nam dajo sliko bogat vpogled v to, kaj pomeni, da nevronska mreža nekaj zna.