Zahvaljujem se Marku Đuraseviću na tome što je dobra osoba i odličan mentor.
Također se zahvaljujem svojoj obitelji, bez koje ne bih imala ni Z od završnog rada.
Zahvaljujem se i Lauri Petan koja me motivirala da ne odustanem.

Sadržaj

Uvod

Neuronske su mreže korištene za rješavanje raznih problema poput prepoznavanja slika, analize podataka iz društvenih mreža, razvoja tehnologije u svemirskom inženjerstvu, predviđanje prognoze, ali također i igranja raznih igara. Svi su čuli za izradu igrača za šah i go. Kako su te igre poznate diljem svijeta s mnoštvom vrlo poštovanih natjecanja gdje se može lako prikazati njihova učinkovitost protiv vrhunskih ljudskih igrača, više se računalnih inženjera posvećuje njihovom poboljšanju. U ovom je projektu cilj bio izraditi igrača za igru *Minesweeper* korištenjem neuronskih mreža. Mnogi su ljudi bili intrigirani izradom računalnih igrača i za manje poznate igre. *Minesweeper* je, zbog svojih jednostavnih pravila te dizajna da bude poput slagalice, popularan izbor igre. Drugi ljudi su postigli razne rezultate koji variraju s jednostavnosti igre i vrstom mreže. Najuspješniji model na svijetu je postigao 40 posto pobjeda na težini *Expert* koristeći se pojačanim učenjem i konvolucijskim neuronskim mrežama. U nastavku će se proučavati i objašnjavati razni načini dizajna računalnog igrača za *Minesweeper*.

1. *Minesweeper*

Minesweeper je poznata igrica čiji su koautori Curt Johnson i Robert Donner. Prvi je put bila izdana 1990. na Windows 3.0 operacijskom sustavu [1]. Doživjela je veliki uspjeh te ubrzo postala svjetski poznata. Vodila su se mnoga natjecanja još od 1995. Trenutni rekord na težini *Expert* drži Ze-En Ju s 27.53 sekunde [2].

1.1. Dizajn

Minesweeper se sastoji od pravokutne ploče zadanih dimenzija i broja skrivenih mina. Težine su sljedeće:

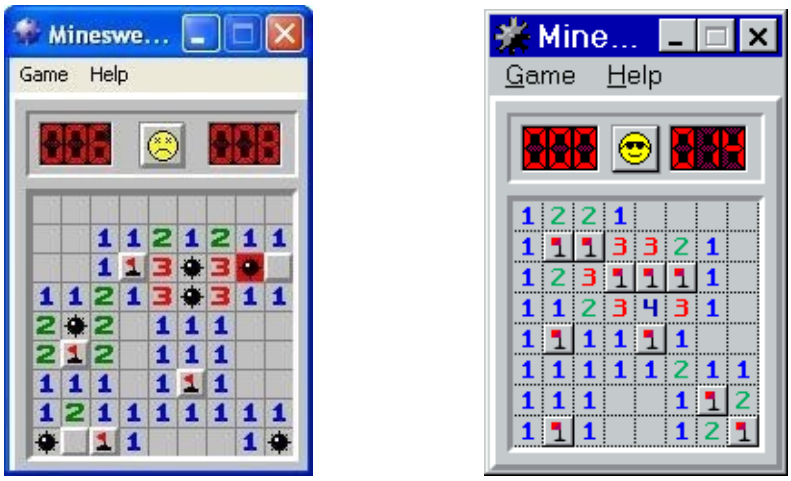
- *Beginner* (8x8 ploča, 10 mina)
- *Intermediate* (16x16 ploča, 40 mina)
- *Expert* (16x30 ploča, 99 mina)

Omogućen je i izbor ploče s proizvoljnim dimenzijama i brojem mina.

Igra se generira prvim klikom na proizvoljno polje. To polje i njegovi susjedi osigurani su da ne sadrže minu. Mine su nasumično raspoređene po ostatku ploče. Na svakom polju koje ne sadrži minu, piše broj susjednih mina kojih može biti od nula do osam. Broj nula se ne ispisuje.

Sl. 1.1 Brojevi u odnosu na broj susjednih mina (mine se označene zastavicom)

Na početku igre neotvoreni dio ploče je skriven. Polja se otvaraju jedno po jedno. Igra traje dok se ne otvori polje s minom (to se smatra gubitkom igre) ili dok nisu otvorena sva polja osim onih koji imaju minu (to se smatra pobjedom).



Sl. 1.2 i 1.3 *Beginner* težina gubitak (lijevo) i pobjeda (desno)

1.2. Strategije igranja

Kada se dođe u poziciju gdje nije lako vidjeti koja od više mogućih polja sadrži minu, jedan od načina je da se proizvoljno polje proglasi minom te se prate susjedna polja dok se dokaže da je mina zaista na tom polju ili dok se ne dođe u kontradikciju. Proces se nastavlja.

Strategija koju koriste natjecatelji je pamćenje i pronalaženje uzoraka. To im omogućava brzo igranje bez gubljenja vremena na razmišljanje.

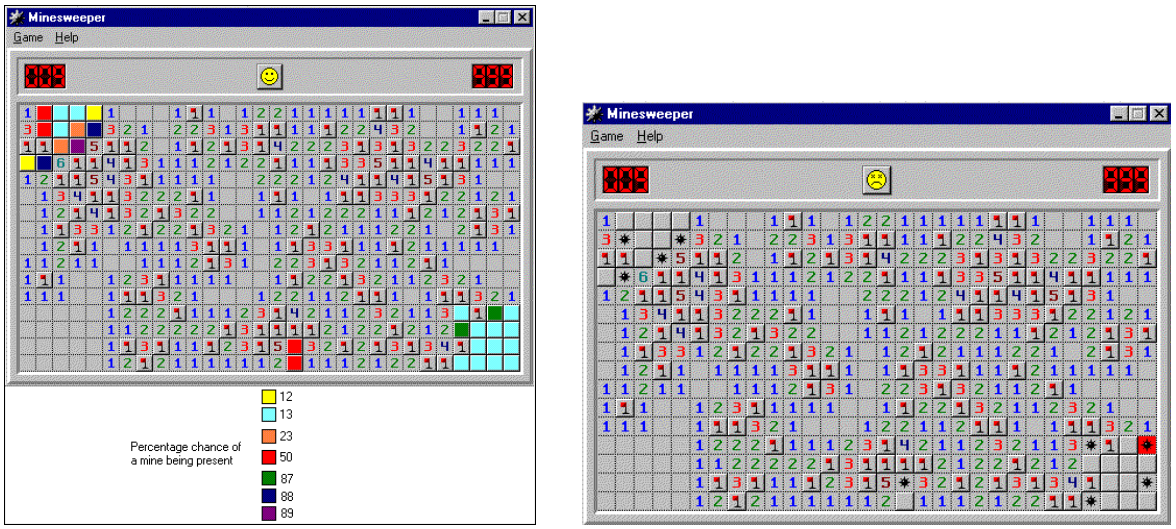
Sl. 1.4 i 1.5: 1-2-X uzorak (X je uvijek mina)

Slike 1.6-1.8 1-1-X (uzorak mora početi na rubu ploče ili kod otvorenih polja te je X sigurno prazno polje. Taj uzorak također može skretati kod uglova)

Slike 1.9 i 1.10 Broj se može smanjiti ako pored sebe ima polja koja su sigurno mine. Ponekad se trebaju koristiti prethodno navedeni uzorci kako bi se broj smanjio (obje slike su primjeri svođenja označenih brojeva na 1-2-1)

Uzorci i logika mogu se kombinirati u logičke lance i tako pronaći teže uočljive uzorke.

Međutim, ponekad je generirana ploča nerješiva jer ima više opcija gdje se mina može nalaziti bez da se zna sa 100 postotnom sigurnošću. U tom slučaju može se izračunati postotak da je polje mina (natjecatelji to ne rade jer zahtjeva previše vremena).



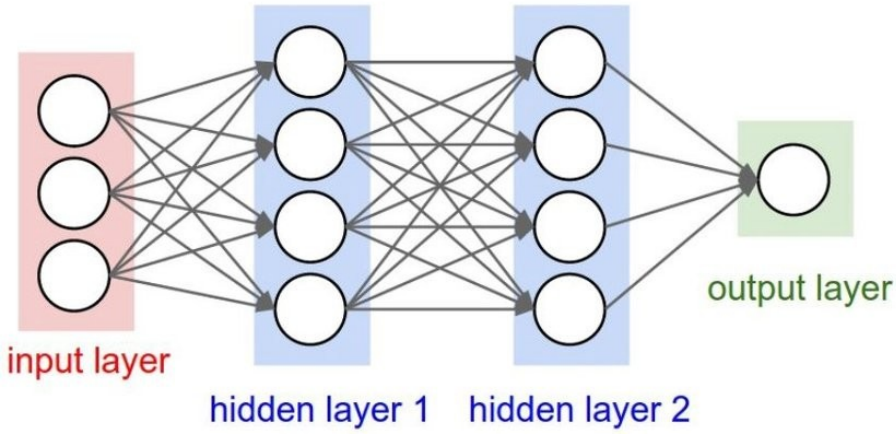
Sl. 1.11 i 1.12 Postotci da pojedino polje sadržava minu (lijevo) i stvarna lokacija mina (desno)

2. Neuronske mreže

Neuronske su mreže u današnje doba izrazito narasle u popularnosti. Premda je 1943. godine nastao pojam neuronskih mreža, njihova je popularnost znatno narasla izumom dubokog učenja (engl. *deeplearning*) [3]. Postoje razne vrste neuronskih mreža koje se upotrebljavaju za rješavanje raznih tipova problema.

2.1. ANN (*Artificial Neural Network*)

ANN je osnovna neuronska mreža inspirirana ljudskim mozgom. Sastoji se od neurona koji su raspoređeni u slojeve. Razlikujemo početni sloj za unos, skrivene slojeve za obradu te posljednji sloj za izlaz podataka. U gustim su slojevima (engl. *dense layers*) svi neuroni iz trenutnog sloja povezani sa neuronima iz sljedećeg. Neuronska mreža uči na način da množi ulaznu vrijednost s težinom na vezi između dva neurona. Rezultati se međusobno zbroje te se dodaje pristranost (engl. *bias*). Pristranost služi za bolje treniranje dinamičnih podataka koji su česti u stvarnom svijetu. Pri inicijalizaciji mreže, težine su postavljene nasumično.

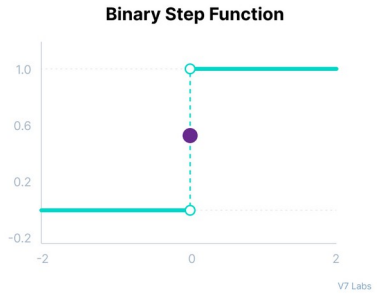


Slika 2.1 Prikaz neurona i međusobne povezanosti u odnosu na slojeve (gusti slojevi)

2.1.1. Aktivacijske funkcije

Nakon što se izračuna nova vrijednost u neuronu, na dobiveni rezultat primjenjuje se aktivacijska funkcija (engl. *activation function*). Ako se koristi linearna aktivacijska funkcija, model će moći opisivati samo linearne odnose. Korištenjem drugačijih aktivacijskih funkcija mogu se opisati nelinearni odnosi i funkcije što je korisnije u primjeni neuronskih mreža. Neke od aktivacijskih funkcija su sljedeće:

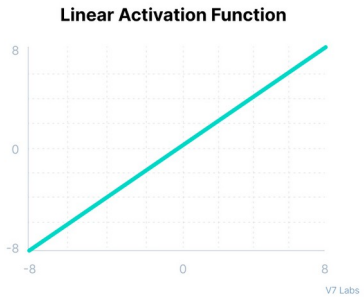
- Binarna step aktivacijska funkcija (danas se rijetko koristi)



$$y = 1, \quad x > 0$$
$$y = 0, \quad x \leq 0$$

Sl. 2.2 Graf linearne step aktivacijske funkcije

- Linearna aktivacijska funkcija (u pravilu se koristi u zadnjem sloju za regresijski model)

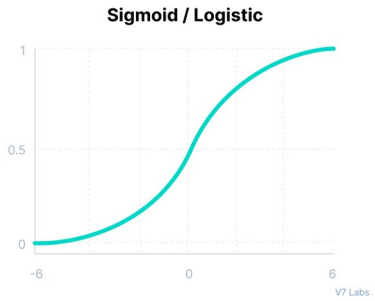


$$y = x$$

Slika 2.3 Graf linearne aktivacijske funkcije

- Sigmoidna aktivacijska funkcija (uvodi nelinearnost te se danas koristi u izlaznom sloju)

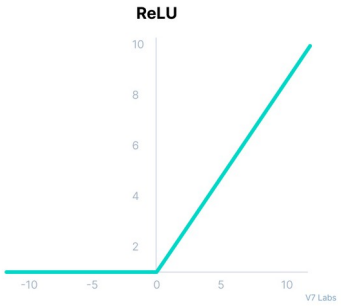
$$y = \frac{1}{1+e^{-x}}$$



Sl. 2.4 Graf sigmoid aktivacijske funkcije

- *Rectified* linearna aktivacijska funkcija, ReLU (najčešća aktivacijska funkcija zbog uvođenja nelinearnosti i brzog računanja)

$$y = x, \quad x > 0$$
$$y = 0, \quad x \leq 0$$



Sl. 2.5 Graf ReLU aktivacijske funkcije

- *Softmax* aktivacijska funkcija (služi za klasifikaciju)

$$S_{i,j} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,j}}}$$

2.1.2. Funkcije gubitka

Funkcija gubitka (engl. *loss function*) je mjera koja prikazuje koliko je model netočan. Služi za podešavanje težina i *bias*-a za vrijeme treniranja. Idealan *loss* je 0. Kategorički *cross entropy loss* eksplicitno se koristi za usporedbu istine i predikcija te je često korišten uz *softmax* aktivacijsku funkciju u zadnjem sloju mreže. Formula za kategorički *cross entropy loss* je sljedeća (y označava istinu, a ŷ predikciju):

$$L_i = -\sum y_{i,j} \log\left(\hat{y}_{i,j}\right)$$

Uz sigmoidnu aktivacijsku funkciju, u zadnjem se sloju koristi binarni *cross entropy loss*. Formula je sljedeća:

$$L_i = \frac{1}{J}\sum_j \left(-y_{i,j}\log\left(\hat{y}_{i,j}\right)-\left(1-y_{i,j}\right)\log\left(1-\hat{y}_{i,j}\right)\right)$$

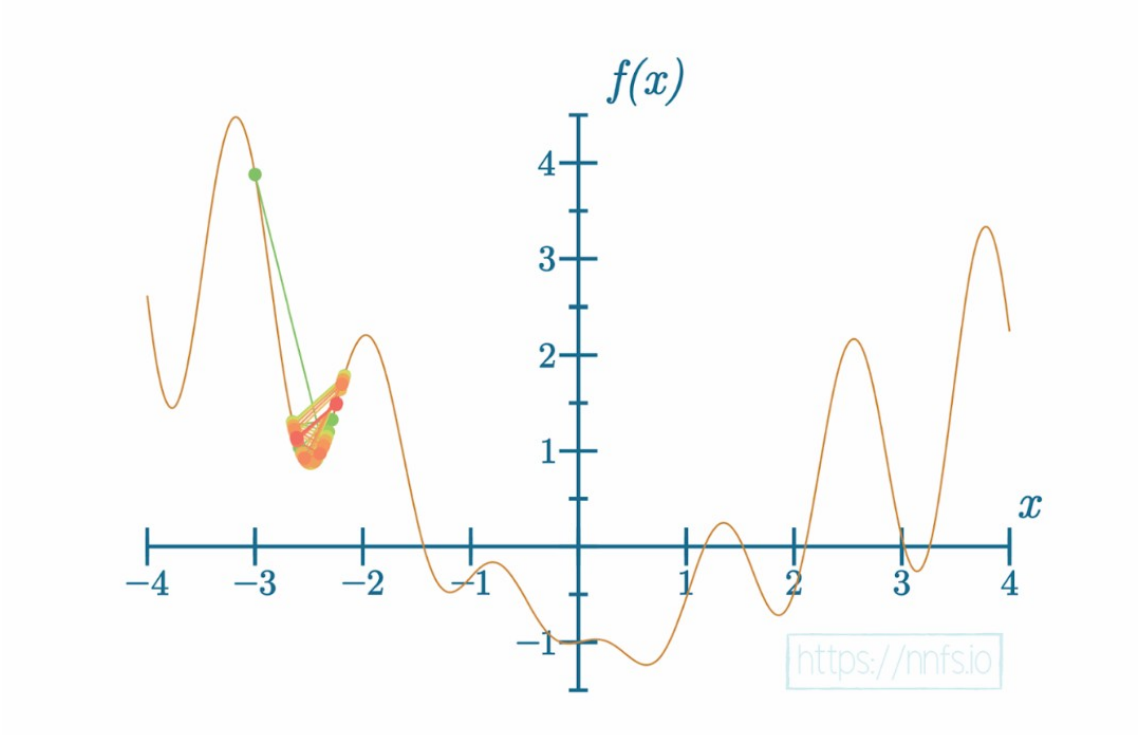
2.1.3. Metode optimizacije

Metode optimizacije služe za uštimavanje težina i *bias*-a s ciljem smanjenja *loss*-a. Većina su varijante stohastičkog gradijentnog spusta (SGD). On *fit*-a jedan po jedan primjerak. *Batch* gradijentni spust *fit*-a cijeli set podataka odjednom, dok mini-batch gradijentni spust *fit*-a dijelove potpunog skupa podataka.

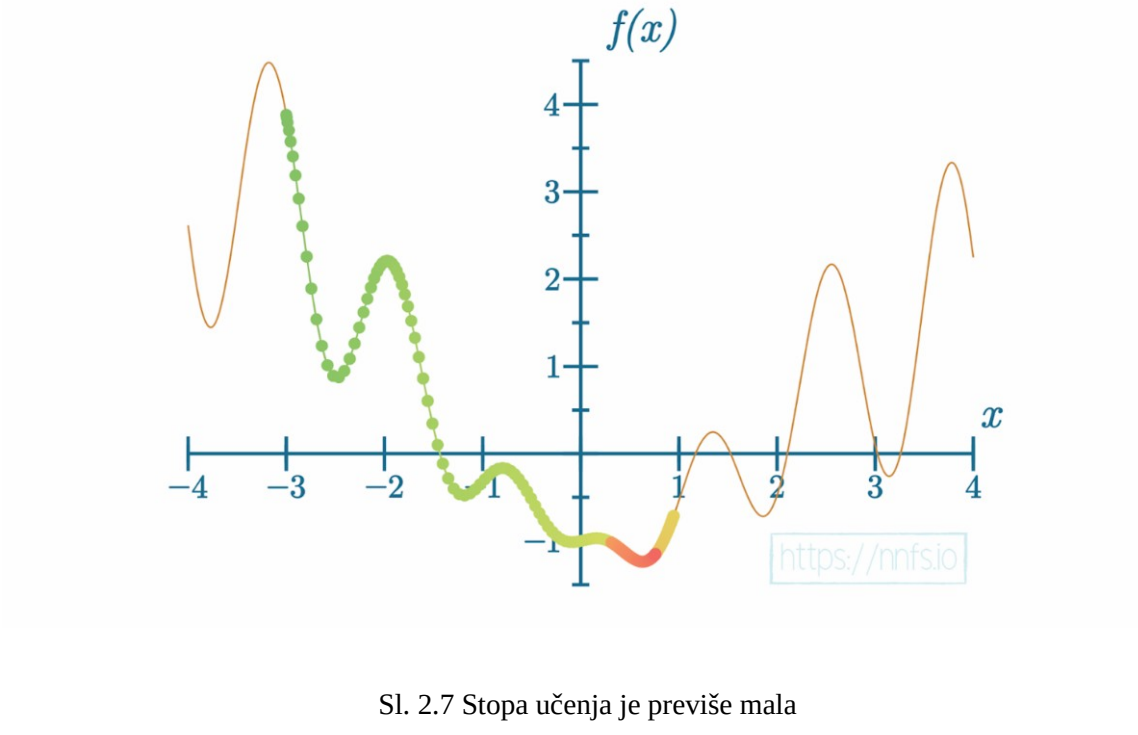
Svaki se prolaz kroz podatke za treniranje za vrijeme *fit*-anja zove epoha.

Stopa učenja (engl. *learning rate*) je promjenjiva vrijednost koja označava veličinu koraka kod svake iteracije pri traženju minimuma funkcije gubitka. Funkcije gubitka imaju više od jednog minimuma te je zato važno da se algoritam ne zaglavi u neoptimalnom lokalnom minimumu.

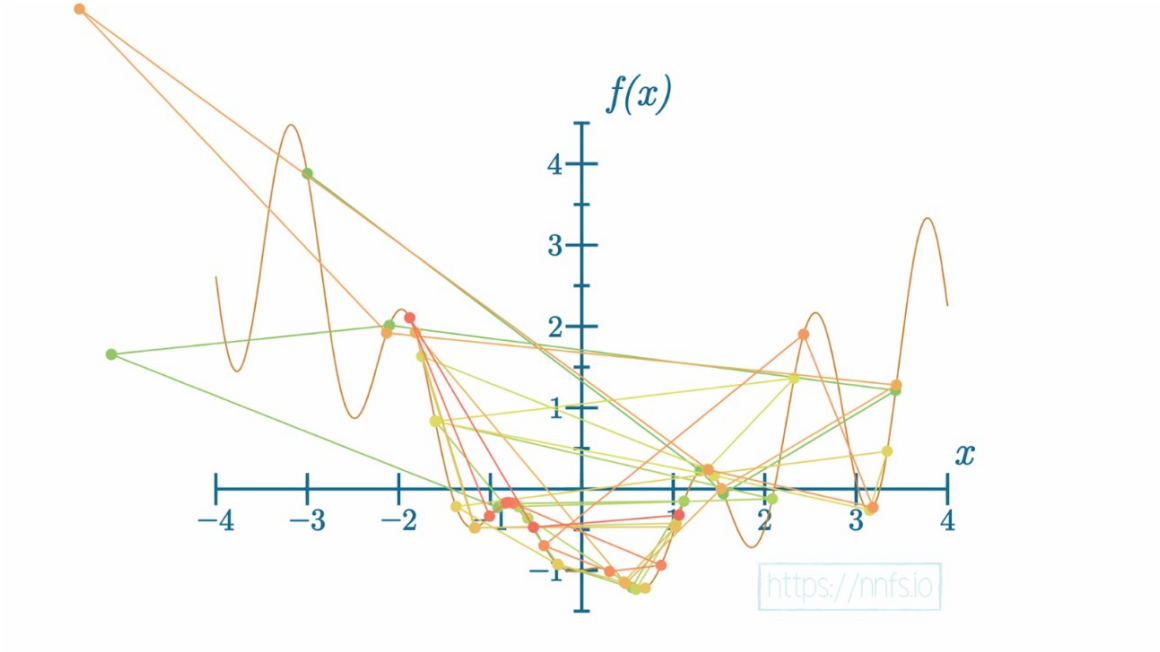
Metodi optimizacije se dodaje zamah koji gradijentu dodaje takozvanu inerciju.



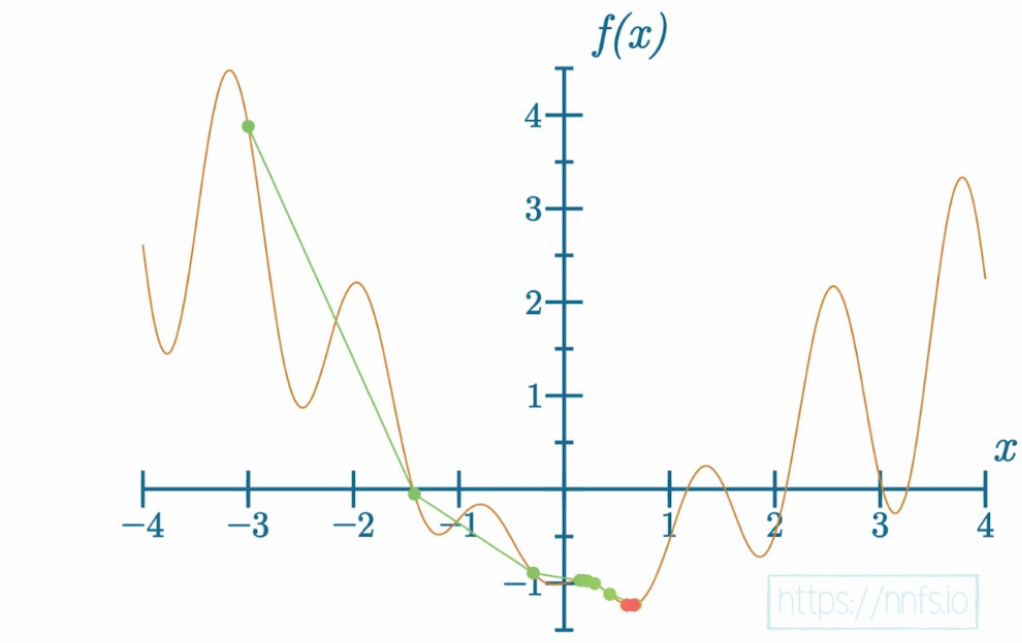
Sl. 2.6 Zaglavljenje u lokalnom minimumu



Sl. 2.7 Stopa učenja je previše mala



Sl. 2.8 Stopa učenja je previše velika



Sl. 2.9 Dobra stopa učenja

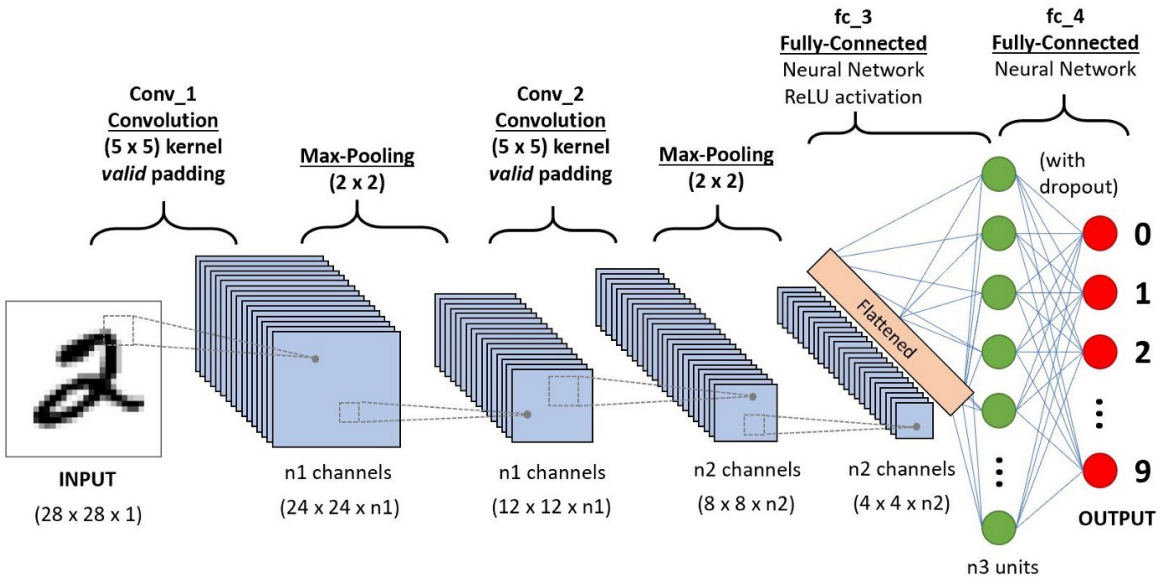
Način na koji se postiže dobro nalaženje minimuma je upotrebom opadanja stope učenja. Započinje se s velikom stopom učenja koja se za vrijeme treniranja smanjuje. Tako se uspješno pronalazi globalni minimum i osigurava stabilnost modela.

Najčešće korištena metoda optimizacije je Adam što je kratko za *Adaptive Momentum*. U mnogim slučajevima donosi najbolje rezultate, ali postoje i iznimke. Sve ovisi o tipu problema koji neuronska mreža pokušava riješiti [4].

2.2. CNN (*Convolutional Neural Network*)

CNN, to jest konvolucijske neuronske mreže su mreže čiji slojevi su pogodni za podatke koji se svode na matrični oblik. Najpoznatiji primjer su slike gdje su pikseli raspoređeni u mrežasti oblik te svaki ima određeni broj kanala koji predstavljaju boju i transparentnost.

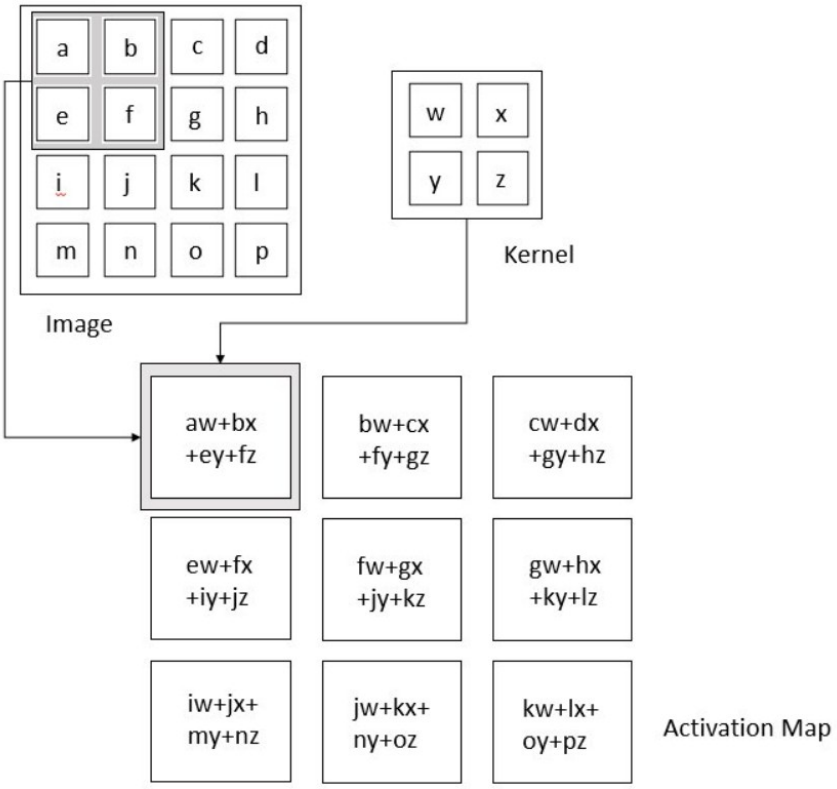
CNN koje se koriste za slike tipično imaju tri osnovna sloja: konvolucijski sloj, *pooling* sloj te potpuno povezani sloj [5].



Sl. 2.10 Primjer konvolucijske neuronske mreže

2.2.1. Konvolucijski sloj

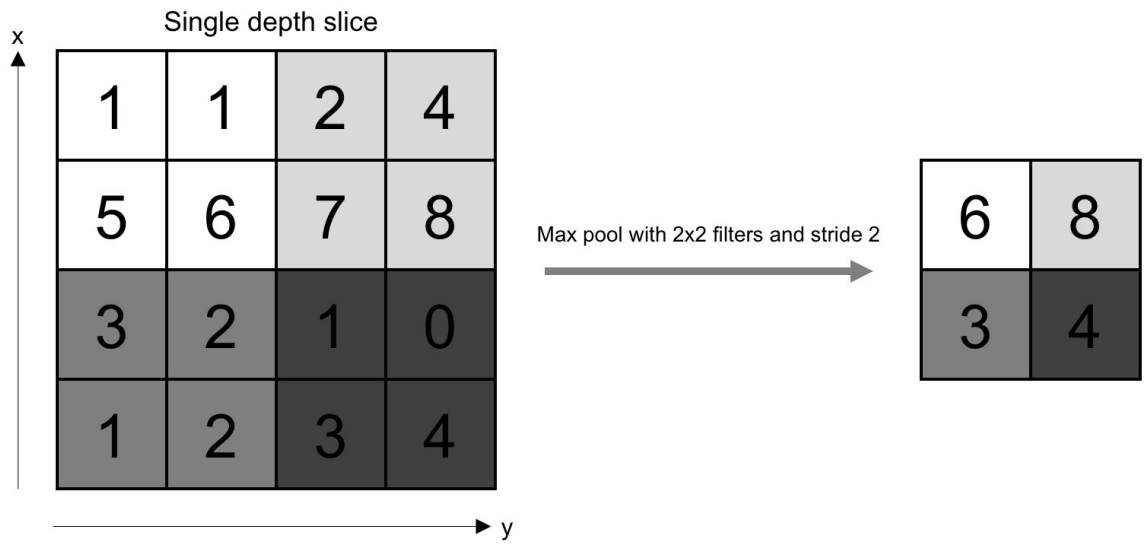
Konvolucijski sloj radi skalarni umnožak dviju matrica. Jedna je zvana kernel te se njene vrijednosti mogu učiti dok druga predstavlja ograničeno područje ulazne slike. Kernel se pomiče po originalnoj slici određenim skokovima. Tim operacijama se gradi aktivacijska mapa.



Sl. 2.11 Operacija konvolucije

2.2.2. *Pooling* sloj

Pooling sloj zamjenjuje izlazne vrijednosti prethodnog sloja različitim *pooling* funkcijama. Rezultat je sloj smanjenih prostornih dimenzija. Neke od *pooling* funkcija su srednja vrijednost, ponderirana srednja vrijednost, L2 norma i *max pooling*.



Sl. 2.12 *Max pooling* operacija

2.2.3. Potpuno povezani sloj

To je gusti sloj opisan u ANN poglavlju. Svi neuroni su međusobno povezani.

3. *Mineweeper* igrač

Igrač za igru Mineweeper pisan je u programskom jeziku Python 3.8.10 te su korištene sljedeće biblioteke:

- JupyterLab
- Keras
- Matplotlib
- NumPy
- pandas

Neuronska mreža trenirana je na računalu s procesorom Intel® Core™ i9-9900K.

3.1. Kako se napao problem

Trebao se razviti automatski igrač koji će bez poznavanja pravila naučiti igrati *Minesweeper*. Jedino što igrač zna je kako otvoriti ćelije. Neuronska mreža ima ulogu mozga. Ona s iskustvom uči koje ćelije sadrže minu s obzirom na otkriveni sadržaj ploče.

Neuronska mreža prima trenutni izgled igre te na osnovu dobivenih informacija predviđa za koju ćeliju je koliki postotak da se na njoj nalazi mina. Otvara zatvorenu ćeliju s najmanjom vjerojatnošću pozicije mine.

Treniranje mreže događa se nad skupom unaprijed određenog broja poteza. Nove igre se generiraju svaki put kada trenutna završi pobjedom ili gubitkom. Na taj način dobiva se velik broj jedinstvenih podataka za treniranje.

S vremenom neuronska mreža uči prepoznavati međusobnu ovisnost ćelija s obzirom na njihov sadržaj i tako počinje pobjeđivati.

3.2. Dizajn

3.2.1. board.py

Datoteka koja sadrži sve informacije za ploču *Minesweeper*-a i metode koje su potrebne za inicijalizaciju te dohvaćanje podataka.

Ploča je opisana NumPy *array*-jem dimenzija visina x širina x 3 kanala koji bilježe je li ćelija otvorena (2 predstavlja otvoreno, a 0 zatvoreno), na kojim pozicijama se nalazi mina te brojevni hintovi od 0 do 8.

```
self.board = np.zeros((difficulty.dim1_height,
difficulty.dim2_width, 3), dtype=int)
```

Također se pamti NumPy *array* u obliku potrebnim za model neuronske mreže. Dimenzije visine i širine su kao prethodne, ali je razlika u broju kanala. Ima sveukupno 11 kanala. Brojevne vrijednosti u *array*-ju su ili 0 ili 1. Prvi kanal označava koje su pozicije otkrivene (1 je otkriveno, 0 je neotkriveno). Drugi kanal je cijeli napunjen s jedinicama, to služi za bolju detekciju granica ploče. Treći do jedanaestog kanala služi za brojčane hintove. Svaki kanal predstavlja jednu vrijednost te jedinice na pozicijama označavaju da je određeni brojčani hint prisutan, a nule da nije.

```
self.data = np.zeros((difficulty.dim1_height,
difficulty.dim2_width, 11), dtype=int)
```

Ploča se inicijalizira prvim klikom na praznu, pokrivenu ćeliju. Na taj se način osigura da prva odabrana ćelija te njeni susjedi ne sadržavaju minu. Ćelije se automatski otvaraju (propadaju) dok se ne dođe do broja različitog od nula. Ne postoji opcija markiranja ćelija zastavicom jer bi to bilo previše poteza za neuronsku mrežu, a igra se zapravo pobjeđuje otvaranjem svih sigurnih ćelija. Zastavice su više pomoćna opcija za ljude.

```
def place_mines(self, y, x):
      mines_counter = 0
      neighbours = self.get_neighbours(y, x)

      while mines_counter < self.number_of_mines:
            random_number = randint(0, self.board_height *
            self.board_width - 1)
            y_mine = random_number // self.board_width
            x_mine = random_number % self.board_width

            if (y, x) != (y_mine, x_mine):
                  i = 0
                  for neighbour in neighbours:
                        if (y_mine, x_mine) != neighbour:
                              i += 1
                  if i == len(neighbours) and not
                  self.is_mine(y_mine, x_mine):
                        self.set_mine(y_mine, x_mine)
                        mines_counter += 1
```

Kod 3.1 – Inicijalno postavljanje mina

Rukovanje statusima pokrenute igre (poput zastavice traje li igra, je li pobijeđena, broj otvorenih polja, itd.) događa se u *game.py* datoteci.

3.2.2. models.py

Datoteka u kojoj se nalaze funkcije u kojima se stvaraju modeli neuronske mreže koristeći biblioteku Keras.

Također postoji klasa ModelAdapter koja služi za adaptiranje ulaza i izlaza ovisno o tipu neuronske mreže koja se koristi. CNN kao ulaz primaju 2D ploču s 11 kanala. Dohvaća se trenutno stanje podataka ploče za treniranje mreže:

```
x_new = board.data
self.x_data[sample_index] = x_new
```

Također se kreira *array* s jednim kanalom gdje se pohranjuju preokrenute vrijednosti kanala koji bilježi je li ćelija otkrivena:

```
x2_new = flipped_revealed(x_new)
self.x2_data[sample_index] = x2_new
```

Te su informacije potrebne za *fit*-anje modela. x2_data množi se sa završnim konvolucijskim slojem modela kako bi se vjerojatnost nalaženja mine na već otkrivenoj ćeliji svela na nula.

Zatim se pomoću metode od modela zvane *predict*, predviđa kolika je vjerojatnost da se na ćeliji nalazi mina za svaku pojedinu ćeliju. Kao sljedeći korak svaka se vjerojatnost zbraja s otkrivenim vrijednostima iz originalnog *array*-ja ploče (otkriveno je 2). Na taj se način osigurava od odabira već otvorene ćelije (čak u slučaju kad model predviđa da su sve vjerojatnosti osim otvorenih jednake 1, zbrajanje s 2 umjesto s 1 nikad neće dopustiti odabir već otvorene ćelije). Zatim se bira ćelija s najmanjom vjerojatnosti da se na njoj nalazi mina. To je njen sljedeći potez te ga uz originalno predviđanje vraća kao povratnu vrijednost:

```
inp = [np.array([x_new]), np.array([x2_new])]
out = self.model.predict(inp)
mine_prob = out.flatten() + get_layer(board.board,
0).flatten()
```

ANN koja koristi guste slojeve treba imati drugačiji ulaz te tako i drugačiji adapter. Svi kanali za pojedinu ćeliju se stapaju u jednu vrijednost. Brojčane vrijednost hintova se skaliraju na decimalnu vrijednost između 0 i 1. To je ulaz za mrežu s gustim slojevima:

```
x_new = hint_layer.copy()
x_new[revealed_layer == 0] = -8
x_new = x_new.flatten()
x_new = (x_new + 8) / 16.0
```

```
def conv2d_relu_sigmoid_binarycrosse_adam1(difficulty):
      input_shape = (difficulty.dim1_height,
      difficulty.dim2_width, 11)  # 11 channels

      in1 = kl.Input(shape=input_shape)
      conv = kl.Conv2D(64, (3, 3), padding='same',
      activation='relu', use_bias=True)(in1)
      conv = kl.Conv2D(64, (3, 3), padding='same',
      activation='relu', use_bias=True)(conv)
      conv = kl.Conv2D(64, (3, 3), padding='same',
      activation='relu', use_bias=True)(conv)
      conv = kl.Conv2D(64, (3, 3), padding='same',
      activation='relu', use_bias=True)(conv)
      conv = kl.Conv2D(1, (1, 1), padding='same',
      activation='sigmoid', use_bias=True)(conv)

      in2 = kl.Input(shape=(difficulty.dim1_height,
      difficulty.dim2_width, 1))
      out = kl.Multiply()([conv, in2])

      model = Model(inputs=[in1, in2], outputs=out)
      model.compile(loss='binary_crossentropy',
      optimizer='adam')

      return model, ModelConv2DAdapter()
```

Kod 3.2 – Primjer Keras modela koji koristi konvolucijske slojeve te vraća model i adapter

3.2.3. brain.py

Ovdje se događa najvažniji dio, treniranje neuronske mreže. Pomoću predviđenih podataka iz *models.py* kreira se istina. Postoje tri funkcije za postavljanje istine.

Prva se fokusira samo na novootvorenu ćeliju. Ako je ćelija sadržavala minu, vrijednost se postavlja na 1. Ostale su vrijednosti kako je model predvidio.

```
truth[y, x, 0] = board.is_mine(y, x)
```

Druga funkcija kao istinu predaje cijelu ploču sa svim informacijama gdje su mine. Vrijednosti su ili 0 ili 1.

```
for by in range(board.board_height):
      for bx in range(board.board_width):
            truth[by, bx, 0] = board.is_mine(by, bx)
```

Treća je kombinacija prve i druge. Ona kao istinu postavlja vrijednost 1 gdje se nalaze mine na svim susjedima od otkrivenih ćelija. U suprotnom se stavlja 0. Nedotaknute vrijednosti su kao što je model predvidio.

```
for cy, cx in board.get_hidden_cells_near_revealed_cells():
      truth[cy, cx, 0] = board.is_mine(cy, cx)
```

Nakon svega toga poziva se *fit* funkcija. Ona se trenira na pojedinim klikovima čiji se broj odredi prilikom pokretanja programa.

```
fit_data = self.model_adapter.get_fit_data()
history = self.model.fit(fit_data, self.model_adapter.y_data,
batch_size=32, epochs=epochs, validation_split=0.3,
callbacks=[tensorboard])
```

Model se trenira po sesijama. To su skupovi od određenog broja odigranih poteza koji se koriste kao podatci za treniranje.

Također se podaci zapisuju u log datoteke. Zapisane vrijednosti su:

- loss (vrijednost funkcije gubitka od podataka za treniranje)
- val_loss (vrijednost funkcije gubitka od podataka za kros validaciju)
- broj ukupno odigranih igara
- broj ukupnih pobjeda
- broj igara po sesiji
- prosjek klikova za jednu igru po sesiji
- prosjek pobjeda po sesiji

- prosjek otkrivenih ćelija za jednu igru po sesiji

Podaci se koriste za izradu grafova.

Model se sprema svakih deset sesija.

```
if (i + 1) % 10 == 0:
      self.model.save(f'models/{self.name}')
```

U *brain.py* se također nalaze funkcije za testiranje istreniranog modela na određenom broju igara, te igranje igre s istreniranim modelom.

3.2.4. train.py

Datoteka u kojoj se postavljaju početni parametri za treniranje modela. Omogućeno je trenirati novi model ili nastaviti trenirati neki od postojećih. Traženi parametri u *train.py* datoteci su:

- ime funkcije koja kreira željeni model
- težina igre (mogu biti i proizvoljne dimenzije te broj mina)
- ime funkcije za dohvaćanje istine
- broj sesija
- broj klikova po sesiji
- broj epoha

Iz imena funkcije koja kreira model dobiva se adapter potreban za korigiranje ulaza i izlaza.

Ime modela je načinjeno od vremenske oznake, naziva težine i naziva funkcije koja stvara model neuronske mreže.

3.2.5. analiza.ipynb

Služi za kreiranje grafova s obzirom na log datoteke generirane za vrijeme treniranja neuronske mreže.

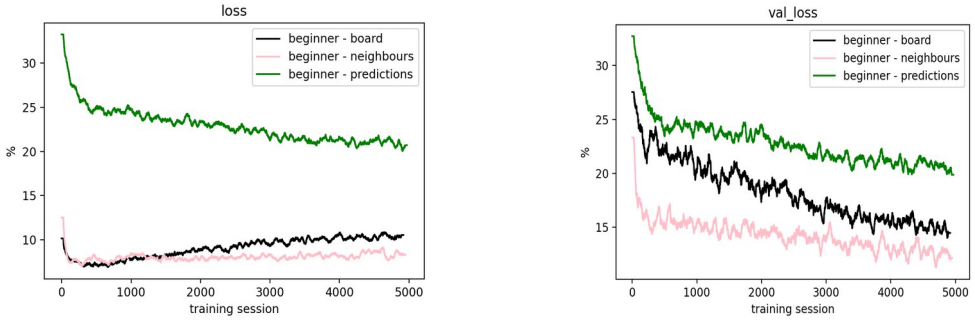
3.3. Rezultati

Predictions predstavljaju dobivanje istine jednu po jednu ćeliju, *board* dobivanje istine preko cijele ploče, a *neighbours* preko susjeda.

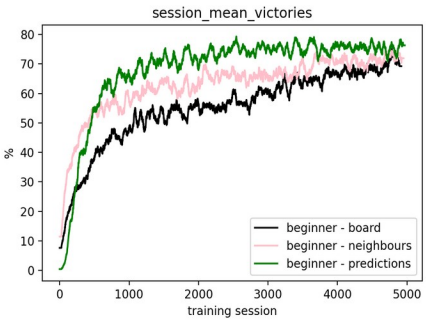
3.3.1. *Beginner* CNN

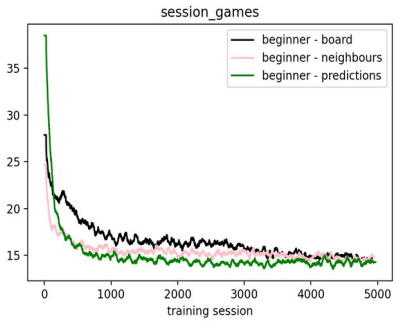
Izgled modela:

- Ulazni sloj
- Četiri 2D konvolucijska sloja sa 64 filtera, kernelom dimenzija 3x3, nepromijenjenim *padding*-om, aktivacijskom funkcijom ReLU i korištenim *bias*-ima
- Jedan 2D konvolucijski sloj s jednim filterom, kernelom dimenzija 1x1, nepromijenjenim *padding*-om, sigmoidnom aktivacijskom funkcijom i korištenim *bias*-ima
- Izlazni sloj se množi s preokrenutim vrijednostima kanala koji opisuje otkrivenost ploče
- Funkcija gubitka je binarni *cross entropy*, a optimizacijska metoda Adam (stopa učenja je 0.001, beta 1 je 0.9, beta 2 je 0.999, epsilon je 1e-7)

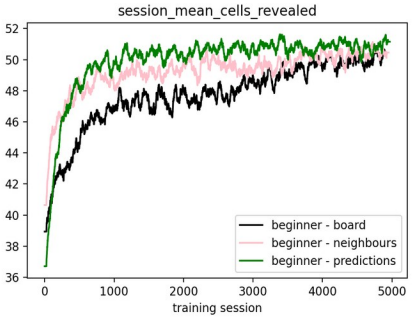


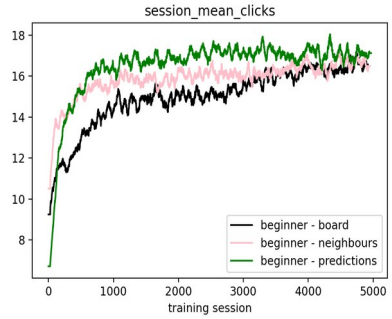
Sl. 3.1 i 3.2 Grafički prikaz *loss*-a (lijevo) i *val_loss*-a (desno) s obzirom na različite metode dobivanja istine





Sl. 3.3 i 3.4 Grafički prikaz srednje vrijednosti pobjeda po sesiji (lijevo) i broj igara po sesiji (desno) s obzirom na različite metode dobivanja istine





Sl. 3.5 i 3.6 Grafički prikaz srednje vrijednosti otkrivenih ćelija u jednoj igri po sesiji (lijevo) i srednje vrijednosti klikova u jednoj igri po sesiji (desno) s obzirom na različite metode dobivanja istine

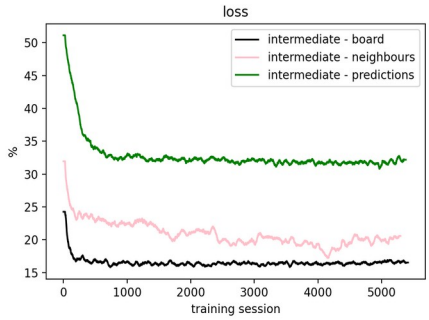
Težina je bila *Beginner* te se u jednoj sesiji dogodilo 250 klikova koji predstavljaju primjerke za *fit*-anje.

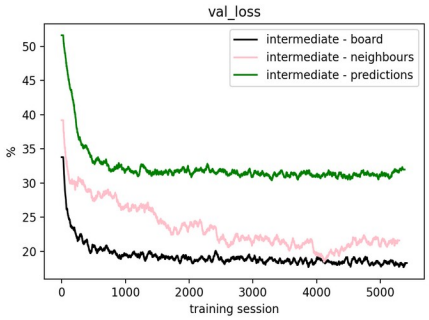
Iz grafova se može uočiti da iako su *board* i *neighbours* na početku imali više pobjeda, *predictions* je prestiglo druge metode te dostiglo u prosjeku 75% pobjeda. Pri 5000-toj sesiji, modeli od sve tri metode se okupljaju oko 70%.

Loss i *val_loss* su za *board* i *neighbours* znatno niži od preostale metode. To označava prenaučenost modela te možda možda čak pogrešno računanje *loss*-a za te metode. Također njihov gubitak raste s vremenom što znači da točnost opada.

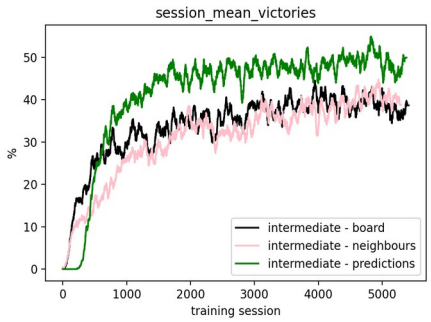
3.3.2. *Intermediate* CNN

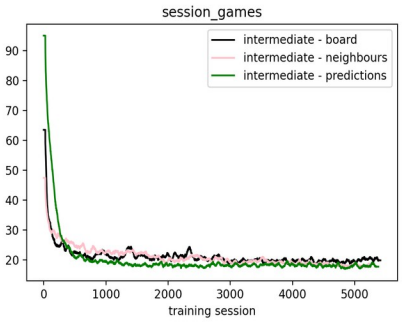
Model je identičan kao prethodni.



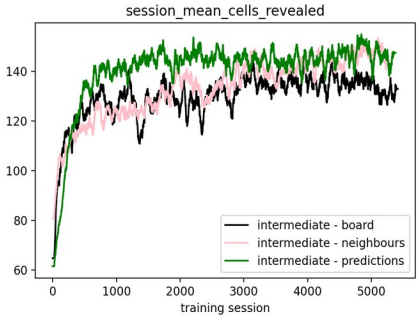


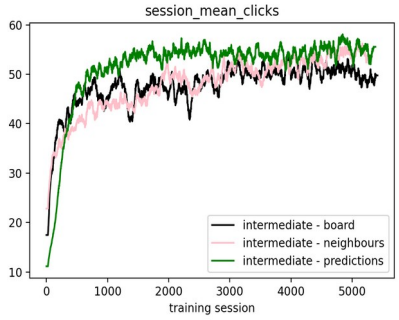
Sl. 3.7 i 3.8 Grafički prikaz *loss*-a (lijevo) i *val_loss*-a (desno) s obzirom na različite metode dobivanja istine





Sl. 3.9 i 3.10 Grafički prikaz srednje vrijednosti pobjeda po sesiji (lijevo) i broj igara po sesiji (desno) s obzirom na različite metode dobivanja istine





Sl. 3.11 i 3.12 Grafički prikaz srednje vrijednosti otkrivenih ćelija u jednoj igri po sesiji (lijevo) i srednje vrijednosti klikova u jednoj igri po sesiji (desno) s obzirom na različite metode dobivanja istine

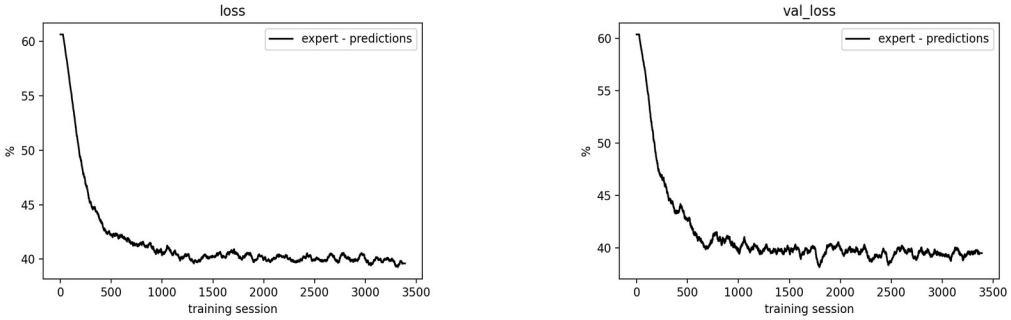
Težina je bila *Intermediate* te se u jednoj sesiji dogodilo 1000 klikova koji predstavljaju primjerke za *fit*-anje.

Kao kod *Beginner*-a, *predictions* postiže bolje rezultate od *board* i *neighbours*. Dostiže prosječno 45% pobjeda po sesiji. Gubitak za board i neighbours je opet puno manji. Iako bi to trebalo značiti da je točnost veća. Kao i kod *Beginner*-a, brže počnu pobjeđivati, ali to se znatno uspori s brojem sesija.

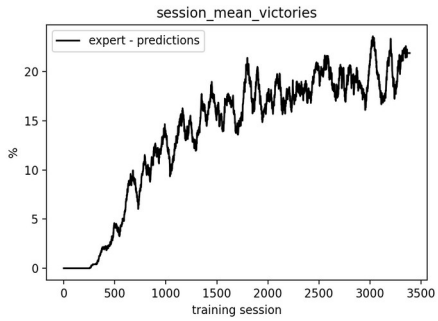
3.3.3. *Expert* CNN

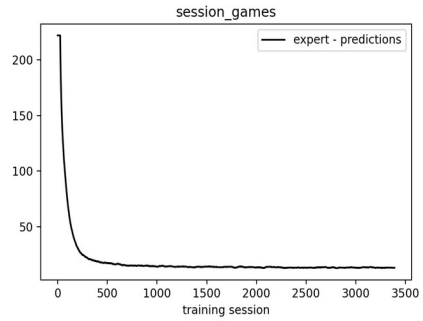
Izgled modela:

- Ulazni sloj
- Pet 2D konvolucijskih slojeva sa 128 filtera, kernelom dimenzija 3x3, nepromijenjenim *padding*-om, aktivacijskom funkcijom ReLU i korištenim *bias*-ima
- Jedan 2D konvolucijski sloj s jednim filterom, kernelom dimenzija 1x1, nepromijenjenim *padding*-om, sigmoidnom aktivacijskom funkcijom i korištenim *bias*-ima
- Izlazni sloj se množi s preokrenutim vrijednostima kanala koji opisuje otkrivenost ploče
- Funkcija gubitka je binarni *cross entropy*, a optimizacijska metoda Adam (stopa učenja je 0.001, beta 1 je 0.9, beta 2 je 0.999, epsilon je 1e-7)

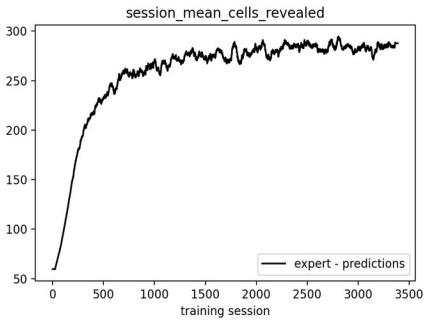


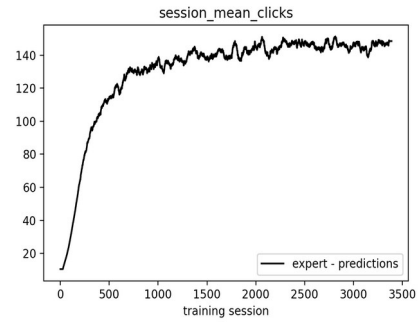
Sl. 3.13 i 3.14 Grafički prikaz *loss*-a (lijevo) i *val_loss*-a (desno) s obzirom na različite metode dobivanja istine





Sl. 3.15 i 3.16 Grafički prikaz srednje vrijednosti pobjeda po sesiji (lijevo) i broj igara po sesiji (desno) s obzirom na različite metode dobivanja istine





Sl. 3.17 i 3.18 Grafički prikaz srednje vrijednosti otkrivenih ćelija u jednoj igri po sesiji (lijevo) i srednje vrijednosti klikova u jednoj igri po sesiji (desno) s obzirom na različite metode dobivanja istine

Težina je bila *Expert* te se u jednoj sesiji dogodilo 2000 klikova koji predstavljaju primjerke za *fit*-anje.

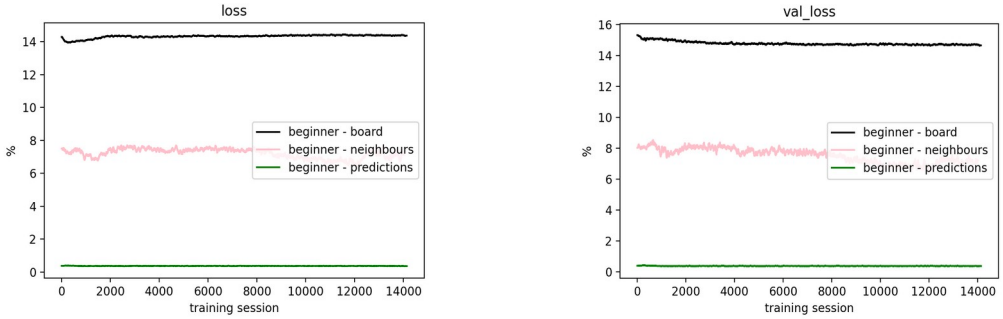
Za *predictions* dostiže prosječno 18% pobjeda po sesiji.

3.3.4. *Beginner* ANN

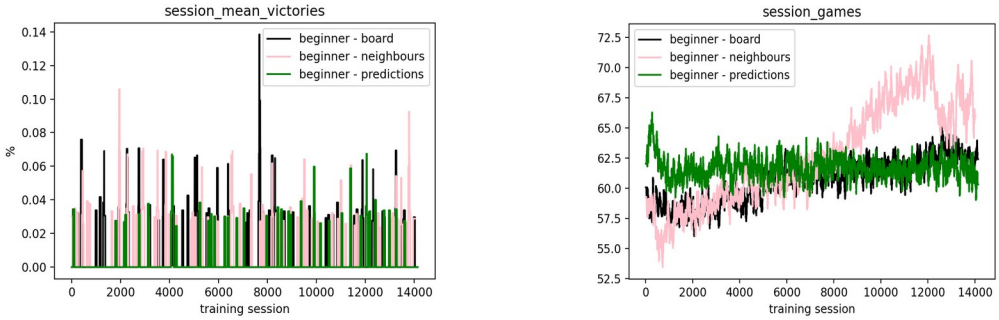
Izgled modela:

- Ulazni sloj
- Dva gusta sloja sa širina*visina*16 neurona, aktivacijskom funkcijom ReLU te korištenim *bias*-ima

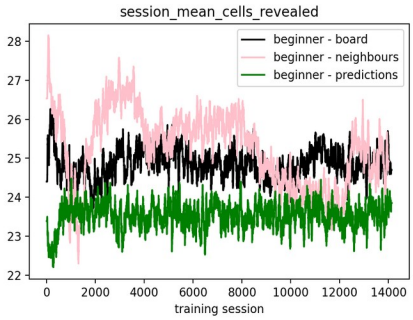
- Gusti sloj sa širina*visina*8 neurona, aktivacijskom funkcijom ReLU te korištenim *bias*-ima
- Gusti sloj sa širina*visina neurona, aktivacijskom funkcijom *softmax* te korištenim *bias*-ima
- Izlazni sloj se množi s preokrenutim vrijednostima kanala koji opisuje otkrivenost ploče
- Funkcija gubitka je *mean squared error* (MSE), a optimizacijska metoda je Adam (stopa učenja je 0.001, beta 1 je 0.9, beta 2 je 0.999, epsilon je 1e-7)

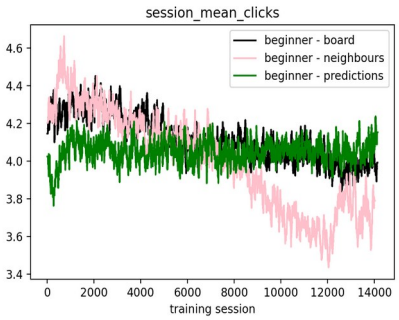


Sl. 3.13 i 3.14 Grafički prikaz *loss*-a (lijevo) i *val_loss*-a (desno) s obzirom na različite metode dobivanja istine



Sl. 3.15 i 3.16 Grafički prikaz srednje vrijednosti pobjeda po sesiji (lijevo) i broj igara po sesiji (desno) s obzirom na različite metode dobivanja istine





Sl. 3.17 i 3.18 Grafički prikaz srednje vrijednosti otkrivenih ćelija u jednoj igri po sesiji (lijevo) i srednje vrijednosti klikova u jednoj igri po sesiji (desno) s obzirom na različite metode dobivanja istine

Težina je bila *Beginner* te se u jednoj sesiji dogodilo 250 klikova koji predstavljaju primjerke za *fit*-anje.

Gusti slojevi i jednostavna ANN mreža nisu napravljene za učenje *Minesweeper*-a. Rezultati ukazuju na potpuno nasumično biranje poteza.

Za *predictions* se broj igara po sesiji te broj klikova nije promijenio. Za *neighbours* izgleda kao da počinje učiti, ali zatim se dogodi iznenadni skok na staru vrijednost. Također je gubitak za *predictions* sumnjivo mali, što ukazuje na loš model.

4. Prijedlog za poboljšanje

Prethodne su neuronske mreže bile trenirane s nadziranim učenjem. Dobivale su skup podataka koji je označavao istinu. Način na koji bi se povećala točnost i vjerojatnost pobjede je korištenje pojačanog učenja (engl. *reinforcement learning*).

Pojačano učenje radi na način da se dobro ponašanje nagrađuje, a loše kažnjava. Agent koji traži optimalno rješenje gleda dugoročnu i ukupno maksimalnu nagradu kako se ne bi zaglavio u lažnim optimumima. S vremenom se počnu izbjegavati loši koraci i aktivno tražiti pozitivni.

Za razliku od Q učenja, u dubokim Q neuronskim mrežama (DQN) nije potrebna Q tablica koja inače predstavlja Q vrijednosti za svaki mogući potez i za svako stanje. Neuronske mreže preuzimaju ulogu Q tablice koja bi za kompleksne slučaje zauzimala previše memorije. Neuronske mreže mogu postići veliku kompleksnost bez zauzimanja toliko memorije. To je velika prednost DQN-a [6].

Igrač za Minesweeper bi na ovaj način puno više naučio za vrijeme treniranja, nego što to pruža nadzirano učenje.

Zaključak

Konvolucijske neuronske mreže su ispale najbolja opcija za igru *Minesweeper* jer su u stanju vidjeti veze susjednih polja što je zapravo srž igranja te igre. Dobivanje istine jedno po jedno polje se s vremenom pokazalo isplativije, nego otkrivanja većeg dijela ploče za istinu kod treniranja. Gusti slojevi koji čine osnovnu ANN su se pokazali kao neuspješan način za treniranje igrača. Povećanjem složenosti igre, opada postotak pobjeda po sesiji. Igrač bi se mogao poboljšati pojačanim učenjem.

Literatura

Slike

[1.1] How To Play Minesweeper, Poveznica: <https://minesweepergame.com/strategy/how-to-play-minesweeper.php>

[1.2-1.3] Minesweeper Game Downloads, Poveznica: <https://minesweepergame.com/download/game.php?id=2>

[1.4-1.10] Minesweeper Strategy – Patterns, Poveznica: <https://minesweepergame.com/strategy/patterns.php>

[1.11-1.12] Barrett, S.,1999., Minesweeper: Advanced Tactics, Poveznica: <https://nothings.org/games/minesweeper/>

[2.1] Amherd, F., Rodriguez, E., siječanj 2021., Poveznica: <https://www.researchgate.net/figure/Dense-Neural-Network_fig5_348402885>

[2.2-2.5] 12 Types of Neural Network Activation Functions: How to Choose?, Baheti, P., 24. 5. 2022., Poveznica: <https://www.v7labs.com/blog/neural-networks-activation-functions#:~:text=An%20Activation%20Function%20decides%20whether,prediction%20using%20simpler%20mathematical%20operations.>

[2.6] Learning Rate Local Minimum, Poveznica: <https://nnfs.io/and/>

[2.7] Learning Rate Too Small, 200 Steps, Poveznica: <https://nnfs.io/pog/>

[2.8] Learning Rate Too Big, Poveznica: <https://nnfs.io/log/>

[2.9] Good Learning Rate, Poveznica: <https://nnfs.io/rog>

[2.10] Saha, S., A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, 15.12.2018., Poveznica: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

[2.11-2.12] Mishra, M., Convolutional Neural Networks, Explained, 26.8.2020., Poveznica: <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939#:~:text=A%20CNN%20typically%20has%20three,and%20a%20fully%20connected%20layer.>

Pisana djela

[1] Damien Moore, 15.9.2021., Windows Entertainment Pack, <https://minesweepergame.com/history/windows-entertainment-pack.php>

[2] Minesweeper World Records, <https://minesweepergame.com/world-records.php>

[3] Clabaugh, C., Myszewski, D., Pang, J., 2000., <https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html>

[4] Kinsley, H., Kukieła, D., Neural Networks from Scratch in Python, 2020.

[5] Mishra, M., Convolutional Neural Networks, Explained, 26.8.2020., Poveznica: <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939#:~:text=A%20CNN%20typically%20has%20three,and%20a%20fully%20connected%20layer>

[6] Kinsley, H., Reinforcement Learning Basics (Q and Deep Q Learning), 2019., Poveznica: <https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/>

Sažetak

Automatska izrada igrača za igru *Minesweeper* korištenjem neuronskih mreža

Rad se bavi izradom igrača za igru *Minesweeper* korištenjem neuronskih mreža. Proučena je teorija neuronskih mreža te su proučeni pojmovi korišteni kod objašnjenja rezultata. Objašnjena je programska implementacija igrača te su trenirane različite neuronske mreže na različitim težinama igre Minesweeper.

Ključne riječi: *Minesweeper*, programski igrač, neuronske mreže, konvolucijske neuronske mreže, funkcije aktivacije, metode optimizacije, funkcije gubitka, pojačano učenje

Summary

Automated player design for the *Minesweeper* game using neural networks

This thesis deals with the creation of a player for the game *Minesweeper* using neural networks. The theory of neural networks is studied and the studied terms are used in result analysis. The code implementation is explained and various neural networks are trained on different difficulties of the game *Minesweeper*.

Keywords: *Minesweeper*, program player, neural networks, convolutional neural networks, activation functions, optimization methods, loss functions, reinforced learning