UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,

RAČUNALNIŠTVO IN INFORMATIKO

Nejc Planer

NAPOVEDOVANJE ZMAGOVALCA NOGOMETNE TEKME Z REKURENTNO NEVRONSKO MREŽO LSTM

Diplomsko delo

Maribor, julij 2019

NAPOVEDOVANJE ZMAGOVALCA NOGOMETNE TEKME Z REKURENTNO NEVRONSKO MREŽO LSTM

Diplomsko delo

Študent:	Nejc Planer
Študijski program:	Univerzitetni študijski program Računalništvo in informacijske tehnologije
Mentor:	doc. dr. Danilo Korže, univ. dipl. inž. rač. in inf.
Somentor:	asist. Mladen Borovič, mag. inž. rač. in inf. tehnol.

Zahvala

Zahvaljujem se mentorju doc. dr. Danilu Koržetu in somentorju asist. Mladenu Boroviču za nasvete in pomoč pri opravljanju diplomskega dela.

Zahvala gre tudi moji družini za podporo pri študiju.

Napovedovanje zmagovalca nogometne tekme z rekurentno nevronsko mrežo LSTM

Ključne besede: napovedovanje, rekurentne nevornske mreže, celica LSTM, nogomet, umetna inteligenca

UDK: 004.032.26(043.2)

Povzetek

V diplomskem delu so predstavljene rekurentne nevronske mreže in primer njihove uporabe. V prvem delu je razloženo njihovo delovanje in vrsti nevronov, od katerih se kasneje uporabi celica LSTM (dolgo-kratko ročna spominska celica). *To je aplicirano tudi na primerih napovedovanja zmagovalca, ali pade več kot 1,5 ali 2,5 gola na tekmo in ali obe ekipi zadeneta. Napovedljivost zmagovalca ligaških tekem je od 61 do 72 odstotkov, zmagovalca nogometnih turnirjev pa od 65 do 70 odstotkov. Uporabljene so angleška, francoska, italijanska, nemška, španska in slovenska liga ter tekmovanji Copa America in svetovno prvenstvo.*

Predicting football match winner with LSTM recurrent neural network

Keywords: prediction, recurrent neural network, LSTM cell, football, artificial intelligence

UDC: 004.032.26(043.2)

Abstract

This diploma thesis presents recurrent neural networks and example of their usage. In the first part it is explained how they work and their types of neurons of which LSTM cell is later used. This is applied on the examples of predicting football match winner, whether there are more than 1,5 or 2,5 goals per game and whether both teams score. The predictability of football league match winner is from 61 to 72 percent and football tournament winner from 65 to 70 percent. Used leagues are from England, France, Germany, Spain and Slovenia and tournaments Copa America and World Cup.

KAZALO

KAZALO SLIK

KAZALO TABEL

UPORABLJENE KRATICE

Celica LSTM – dolgo-kratko ročna spominska celica (Long short-term memory)

Celica GRU – vratna rekurentna celica (Gated recurrent unit)

CSV – vrednosti, ločene z vejico (Comma separated values)

1 UVOD

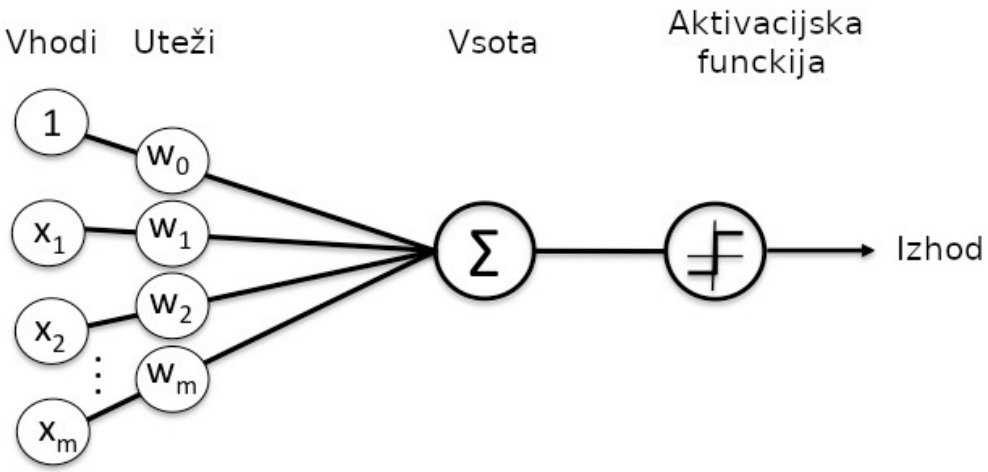
Besedi statistika in umetna inteligenca lahko dandanes večkrat zasledimo omenjeni skupaj. Prva je ključen del športa in jo preučujejo strokovnjaki iz veliko področij, iz nje pa ugotavljajo trende in izboljšave. O drugi pa je govora na najrazličnejših področjih. V zadnjih letih je ta domena računalništva postala zelo popularna, zaradi prebojev na področjih prepoznave govora, računalniškega vida in tudi v procesiranju teksta. Vendar se lahko umetna inteligenca aplicira na mnogo področij, dokler je le dovolj smiselnih podatkov za učenje modela. Tudi internet je poln dobre dokumentacije in forumov. Pojem umetna inteligenca pokriva vse, od odločitvenih dreves do bolj kompleksnih nevronskih mrež, ki smo jih prav tako uporabili v tej diplomski nalogi. In zato se zdi, da sta statistika in umetna inteligenca zelo povezani, saj prva zbira informacije o preteklosti, druga pa se lahko na podlagi teh nauči določenih vzorcev in dokaj dobro napove prihodnost.

Namen diplomske naloge je ugotoviti, ali lahko z nogometno statistiko in nevronskimi mrežami uspešno napovemo, kaj se bo zgodilo v prihodnjih tekmah. Za to je potrebno podatke, ki so na voljo, ustrezno preoblikovati in dodati nove, da si izboljšamo natančnost. Najti pa je treba tudi primerno mrežo za učenje, saj se mreže razlikujejo po številu plasti in nevronov ter ostalih parametrih, vse pa ne dosegajo iste natančnosti.

V prvem delu diplomskega dela bomo spoznali, kako delujejo rekurentne nevronske mreže in kakšne tipe nevronov poznamo ter njihovo notranje delovanje. V drugem delu pa bomo pogledali, kako se nevronska mreža implementira v programskem jeziku Python in kje lahko najdemo zgodovinske nogometne podatke ter kako jih lahko obdelamo. Na koncu bomo pregledali rezultate, ki smo jih uspeli doseči v našem delu in ugotovili, kaj ima dobro napovedljivost ter kako bi lahko izboljšali naš model.

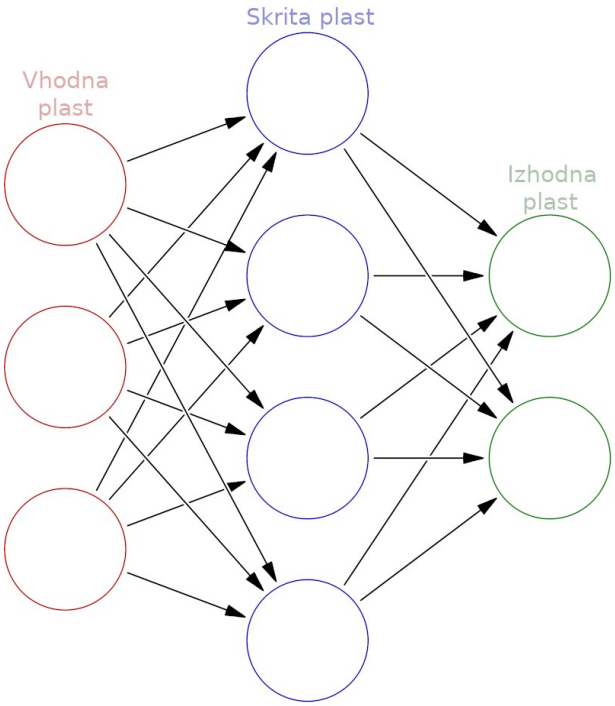
2 NEVRONSKE MREŽE

Nevronske mreže so skupek algoritmov, ki so modelirani po vzoru človeških možganov. Dobri so predvsem v prepoznavanju vzorcev. Razumejo numerične podatke, ki so predstavljeni v obliki vektorjev, zato je potrebno vhodne podatke preoblikovati v takšno obliko, da jih analizirajo in združijo v skupine ali pa jih klasificirajo. Torej v veliki količini podatkov ugotavljajo podobnosti in vzorce, ki pripeljejo do nekega stanja.



Slika 2.1: Diagram nevrona [1]

Osnovna enota nevronske mreže je nevron, ki je lahko poimenovan tudi vozlišče ali celica, katerega diagram lahko vidimo na sliki 2.1. Ti so med sabo povezani z utežmi, ki določene povezave zavirajo. Vhodi so spremenjeni glede na uteži in na koncu sešteti. Aktivacijska funkcija pa poskrbi za končen rezultat, ki je običajno med 0 in 1, lahko pa tudi med -1 in 1.



Slika 2.2: Diagram plasti [2]

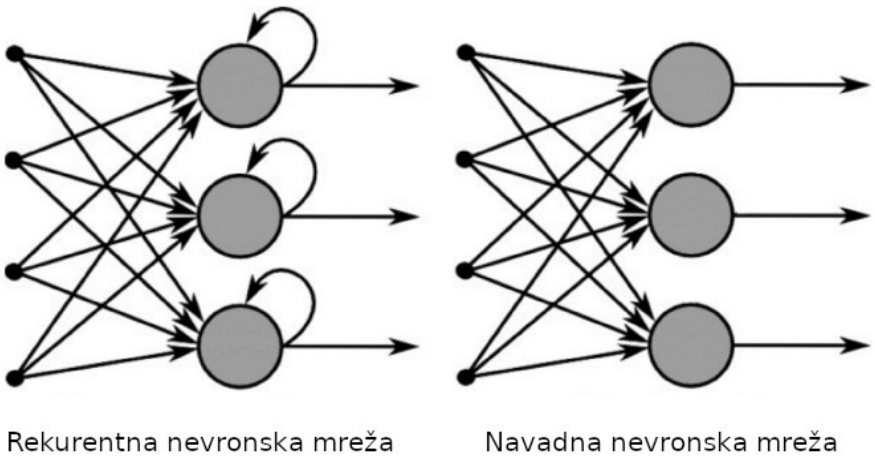
Plast mreže je ena skupina nevronov, ki delujejo kot stikala in se vklopijo oziroma izklopijo, ko gredo vhodni podatki skozi mrežo. Izhod vsake plasti je tako vhod naslednje plasti. Kot lahko vidimo na sliki 2.2, ima vsaka mreža vhodno in izhodno plast, vmesne skrite plasti pa niso nujne in jih je lahko poljubno število.

Poznamo različne tipe nevronskih mrež. Najprej imamo pragovno logično enoto (ang. Threshold logic unit – TLU) oziroma perceptron. To je nevron, ki se uporablja pri binarni klasifikaciji, kar pomeni, da vzorce razdeli na dva dela. Kadar pa je problem bolj kompleksen in ni linearno ločljiv, pa uporabljamo večplastne nevronske mreže, ki jim pogosto pravimo večplastni perceptron (ang. multilayer perceptron), oziroma nevronska mreža z veriženjem naprej (ang. feedforward neural network). Te učimo z algoritmom vzratnega razširjanja (ang. back-propagation algorithm) [3]. Za prepoznavanje slik pa uporabljamo konvolucijske nevronske mreže (ang. Convolutional Neural Network). Te vzamejo sliko in z utežmi določijo pomembnost objektov na sliki [4]. Na koncu imamo še sekvenčne podatke, za katere se najbolje izkažejo rekurentne nevronske mreže. Inovacije na področju strojnega učenja lahko najdemo na [5].

2.1 Rekurentne nevronske mreže

Rekurentne nevronske mreže za razliko od navadnih, kot vhod ne dobijo le vhodnega vektorja, temveč tudi stanje iz preteklosti. Odločitev, ki jo mreža sprejme pri časovnem koraku *t-1* direktno vpliva na časovni korak *t*. Tako imajo dva vira vhodov, sedanjega in preteklega, ki skupaj določita izhod novih podatkov.

Ker zelo dobre rezultate dosegajo s sekvenčnimi rezulatati, se veliko uporabljajo pri procesiranju besedil in govora. Tekst se drugače procesira kot celota, ampak dobimo več semantičnih informacij, če procesiramo besedo po besedo v pravilnem vrstnem redu. Uporabljajo se za odgovarjanje na vprašanja, ugotavljanje naslednje besede v stavku, generiranje kar celotnega besedila, opisovanje slik in podobno. Še eno področje uporabe so biološki podatki, ki si sledijo v zaporedju, kot so amino kisline ali pa nukleotiti DNK [6].



Slika 2.3: Razlika med rekurentno in navadno nevronsko mrežo [7]

Kakor prikazuje slika 2.3, je glavna razlika v zanki pri samem nevronu, kar omogoča vpogled v pretekle odločitve pri rekurentnih nevronskih mrežah. Lahko rečemo, da imajo te mreže nekakšen spomin. To je predvsem smiselno, kadar se neka informacija lahko skriva v samem zaporedju dogodkov, ki jih pa z navadnimi mrežami ne moremo

ugotoviti. Ta informacija je ohranjena v skritem stanju mreže in poskuša najti korelacijo med stanji, ki jih ločuje veliko trenutkov in zato lahko rečemo, da imajo rekurentne nevronske mreže zraven navadnih uteži še uteži, ki si jih nevroni delijo skozi čas. Skrito stanje h_t lahko opišemo tudi z enačbo:

$$h_t = \phi\left(Wx_t + Uh_{t-1}\right) \qquad (2.1)$$

Tu je:

h_t – skrito stanje na trenutnem časovnem koraku

x_t – vhod na trenutnem časovnem koraku

W – matrika uteži

h_{t-1} – skrito stanje prejšnjega časovnega koraka

U – prehodna matrika uteži prejšnjih skritih stanj

Torej v (2.1) nam obe matriki uteži filtrirata pomembnost trenutnega in preteklega skritega stanja. Napaka, ki se pri tem ustvari, se vrne z algoritmom vzratnega razširjanja in popravi uteži, dokler se napaka ne more več znižati. Vsota vsega pa je še potem stisnjena v funkcijo ϕ, ki pa je ena izmed funkcij sigmoid ali tanh. Ker se zanka zgodi na vsakem časovnem koraku, skrito stanje ne vsebuje le del prejšnjega stanja, ampak tudi vse pred h_{t-1}, dokler spomin to omogoča.

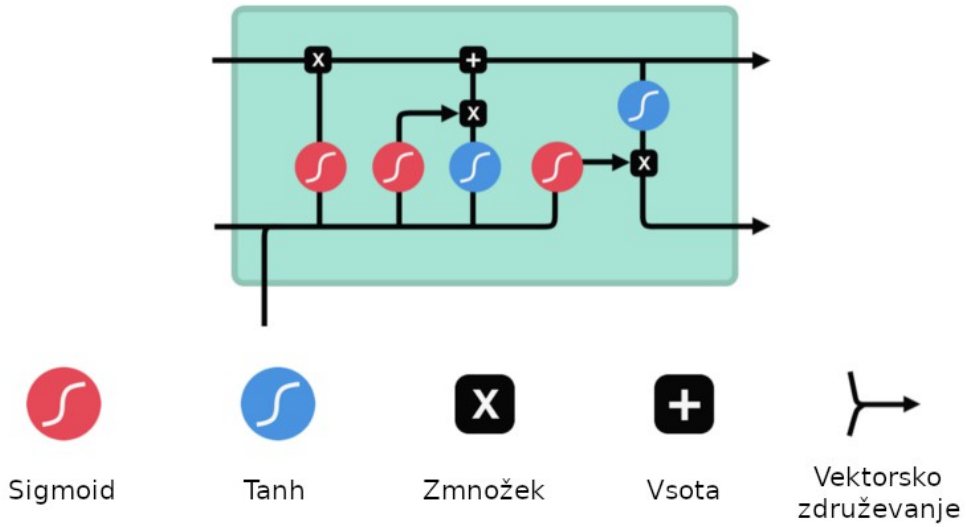
Za ta tip nevronskih mrež sta značilna dva tipa nevronov, to sta celica LSTM (ang. Long short-term memory, dolgo-kratko ročna spominska celica) in celica GRU (ang. Gated recurrent unit, vratna rekurenta celica).

2.2 Celica LSTM (Long short-term memory)

Težava z rekurentno nevronsko mrežo je kapaciteta spomina. Kot rešitev za to

uporabimo celico LSTM. Mreža še vedno deluje na enak način, razlika je le znotraj same celice.

Operacije iz slike 2.4 omogočajo celici, da določi, katere informacije so pomembne in katere ne. Najpomembnejši del celice so njena notranja vrata, ki regulirajo notranjo skrito stanje celice. Celica LSTM nam omogoča, da na trenuten rezultat lahko vplivajo tudi podatki iz začetka učenja in tako izniči samo kratkoročen spomin. Ko pa gredo novi podatki skozi celico, pa njena vrata odločajo, če trenutni korak vpliva na njeno stanje ali ne. Tudi vrata se učijo skozi fazo treniranja mreže.

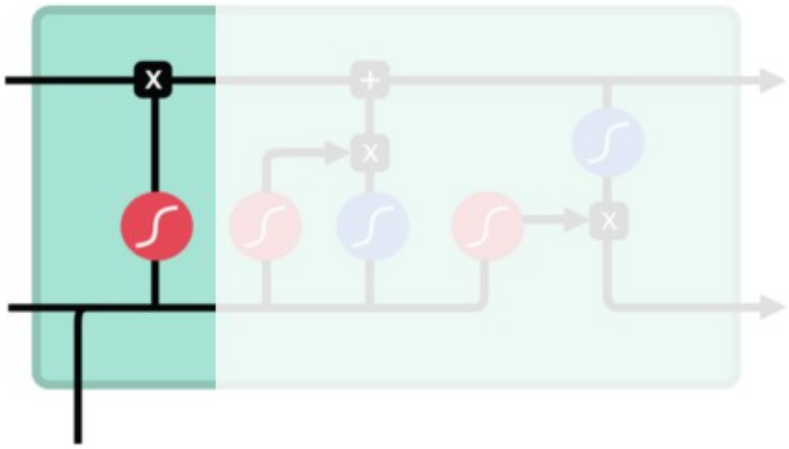


Slika 2.4: Struktura celice LSTM [8]

Tanh aktivacijska funkcija se uporablja za reguliranje vrednosti, ki potujejo skozi mrežo. Te vrednosti normalizira med -1 in 1. Vektorji, ki so vhodi mreže, se skozi le-to zaradi raznih matematičnih operacij veliko spreminjajo in bi zato lahko nekatere vrednosti zelo narasle zato je pomembna regulacija le-teh.

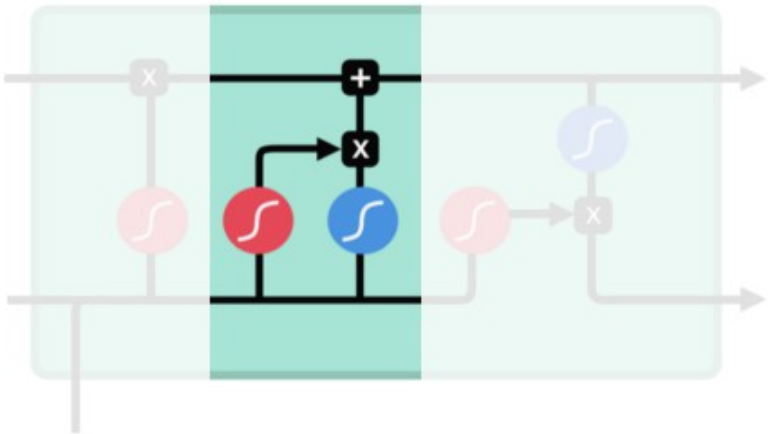
Sigmoid aktivacijska funkcija deluje podobno kot tanh, le z razliko, da vrednosti normalizira med 0 in 1. To služi za posodabljanje in pozabljanje določenih informacij, saj, če bo neka vrednost pomnožena z 0, bo le-ta zanemarjena. Če pa bo pomnožena z 1, pa bo ostala povsem enaka.

V celici imamo tri različna vrata za reguliranje toka podatkov. To so vrata za pozabljanje, vhodna vrata in izhodna vrata.



Slika 2.5: Vrata za pozabljanje [8]

Slika 2.5 prikazuje vrata za pozabljanje, ki odločajo, katere informacije naj bodo zavržene. Trenutna vhodna informacija se skupaj s prejšnjim skritim stanjem pošlje skozi sigmoid aktivacijsko funkcijo in vrne vrednost med 0 in 1. Če je bližje 0 pomeni, da informacija ni bila pomembna in se lahko pozabi, če pa je bližje 1, pa se naj obdrži.

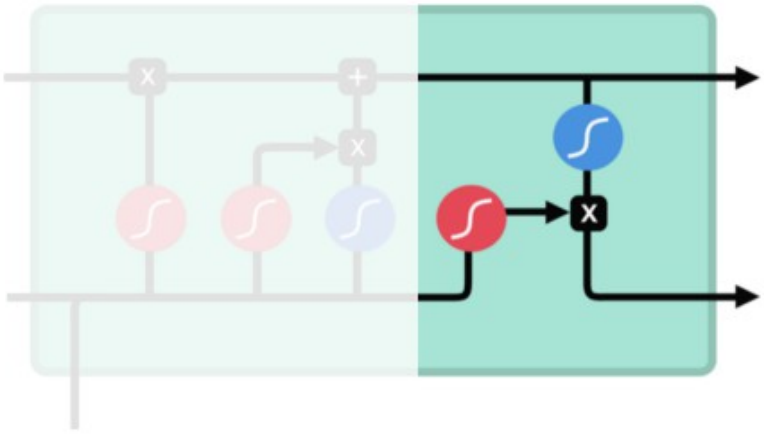


Slika 2.6: Vhodna vrata [8]

Za posodabljanje vrednosti celice imamo vhodna vrata, ki so prikazana na sliki 2.6.

Podobno kot pri vratih za pozabljanje pošljemo tudi tukaj trenutno vhodno informacijo skupaj s prejšnjim skritim stanjem skozi sigmoid aktivacijsko funkcijo. Ta se odloči, katere vrednosti bodo posodobljene, s tem da jih spet preslika v interval med 0 in 1. Isti dve informaciji pa se tukaj pošljeta tudi skozi tanh aktivacijsko funkcijo, ki pa postavi vrednosti med -1 in 1 za regulacijo mreže. Potem se izhoda sigmoid in tanh funkcij med seboj pomnožita in tako sigmoid funkcija pove, kateri normalizirani podatki iz tanh funkcije so pomembni. Na tej stopnji imamo tudi že dovolj podatkov za izračun naslednjega stanja celice. Najprej se pomnoži prejšnje stanje celice z izhodom vrat za pozabljanje, potem to vrednost seštejemo z izhodom vhodnih vrat in tako dobimo novo stanje celice.

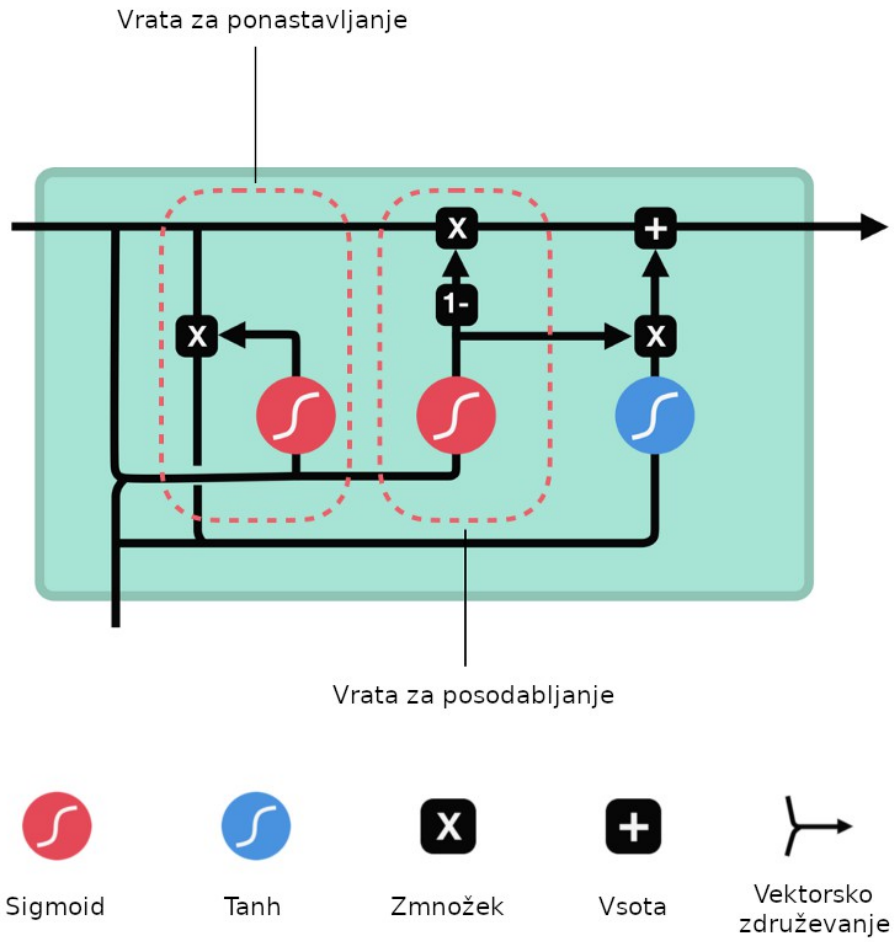
Na koncu imamo še izhodna vrata, prikazana na sliki 2.7. Ta odločajo, kakšno bo naslednje skrito stanje. Kot pri obeh prejšnjih vratih tudi tukaj pošljemo trenutno vhodno informacijo skupaj s prejšnjim skritim stanjem skozi sigmoid funkcijo. Novo stanje celice, ki se je izračunalo prej, pa pošljemo skozi tanh funkcijo in oba izhoda zmnožimo. Ta zmnožek je novo skrito stanje celice in se prenese na naslednji časovni korak.



Slika 2.7: Izhodna vrata [8]

2.3 Celica GRU (Gated recurrent unit)

Podobna celici LSTM je celica GRU. Za razliko od celice LSTM celica GRU nima stanja celice in informacije prenaša kar po skritem stanju. Ima tudi samo dvojna vrata, to so vrata za ponastavljanje (ang. reset gate) in pa vrata za posodabljanje (ang. update gate). Kot lahko vidimo na sliki 2.8, ima celica GRU znotraj manj operacij kot celica LSTM, zato je tudi hitrejša za učenje, kar pa ne pomeni, da je tudi boljša ali slabša. Velikokrat je odvisno od vhodnih podatkov, zato je potrebno preizkusiti oba tipa celice, da dobimo optimalne rezultate.



Slika 2.8: Struktura celice GRU [8]

Vrata za ponastavljanje določajo, koliko informacij je potrebno pozabiti.

Vrata za posodabljanje pa so v principu podobna pozabnim in vhodnim vratom celice LSTM. Določajo torej, katere informacije je potrebno zavreči in katere dodati.

Čeprav je celica GRU tesno povezana s poenostavitvijo celice LSTM, se ne sme obravnavati kot poseben tip celice LSTM, ampak vseeno kot enota za sebe. Obe celici imata podobno uspešnost. Celica GRU je preprostejša in zato lažja za implementacijo. Primernejša bi mogoče bila za manjše množice podatkov, zaradi slabšega spomina kot celica LSTM. Celica GRU je tudi manj testirana, zaradi starejše arhitekture in popularnosti celice LSTM, zato se običajno zdi celica LSTM kot varnejša izbira, še posebej, ko imamo večje množice podatkov [6].

3 IMPLEMENTACIJA

3.1 Nevronska mreža

Nevronsko mrežo smo implementirali v programskem jeziku Python [9], uporabili smo njegovo knjižnico za umetno inteligenco Keras [10]. Ta je narejena za hitro uporabo in eksperimentiranje z nevronskimi mrežami. Je zelo modularna in prijazna uporabniku, sama koda za nevronsko mrežo pa se napiše v samo nekaj vrsticah.

```
1   model = Sequential()
2
3   model.add(LSTM(32, input_shape=(None, x_train.shape[1:][1],),
4                  return_sequences=True))
5   model.add(Activation('sigmoid'))
6   model.add(Dropout(0.2))
7
8   model.add(LSTM(32, return_sequences=True))
9   model.add(Activation('sigmoid'))
10  model.add(Dropout(0.2))
11
12  model.add(LSTM(32, return_sequences=True))
13  model.add(Activation('sigmoid'))
14  model.add(Dropout(0.2))
15
16  model.add(LSTM(2))
17  model.add(Activation('sigmoid'))
18
```

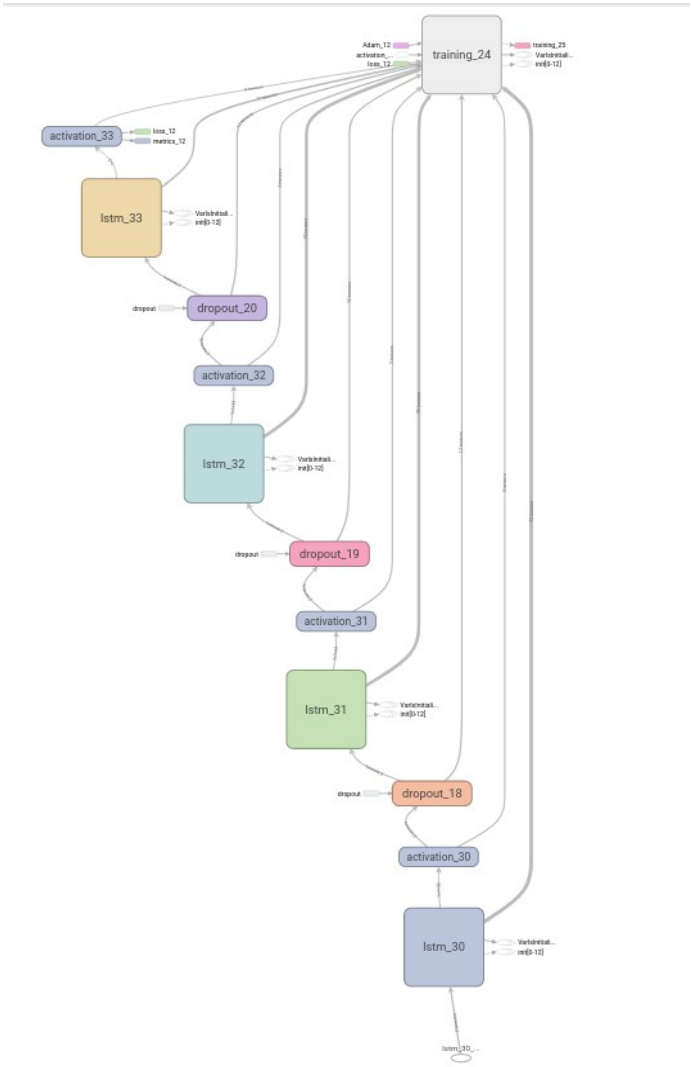
Slika 3.1: Implementacija rekurentne nevronske mreže LSTM v programskem jeziku Python s knjižnico Keras

Na sliki 3.1 vidimo primer implementacije nevronske mreže s knjižnico Keras. Z zelo kratko kodo smo ustvarili štiri plastno rekurentno nevronsko mrežo z celico LSTM. Najprej ustvarimo sekvenčni model, kar omogoča dodajanje plasti eno za drugo in potem samo dodamo želene plasti. Tukaj smo dodali štiri plasti, od tega imajo vhodna

in skrite plasti 32 nevronov, zadnja izhodna plast pa ima 2 nevrona, saj imamo tudi toliko izhodov. Na koncu vsake plasti je še aktivacijska funkcija sigmoid, ki nam še dodatno normalizira vrednosti. Za prvimi tremi plastmi lahko vidimo tudi t. i. Dropout funkcijo, ki poskrbi, da se določen delež naučenega pozabi in tako ne prihaja do prekomernega prileganja. Nevronsko mrežo zaženemo z vrstico kode, ki jo vidimo na sliki 3.2.

```
model.fit(x_train, y_train, epochs=1000,
          batch_size=32, validation_data=(x_test, y_test))
```

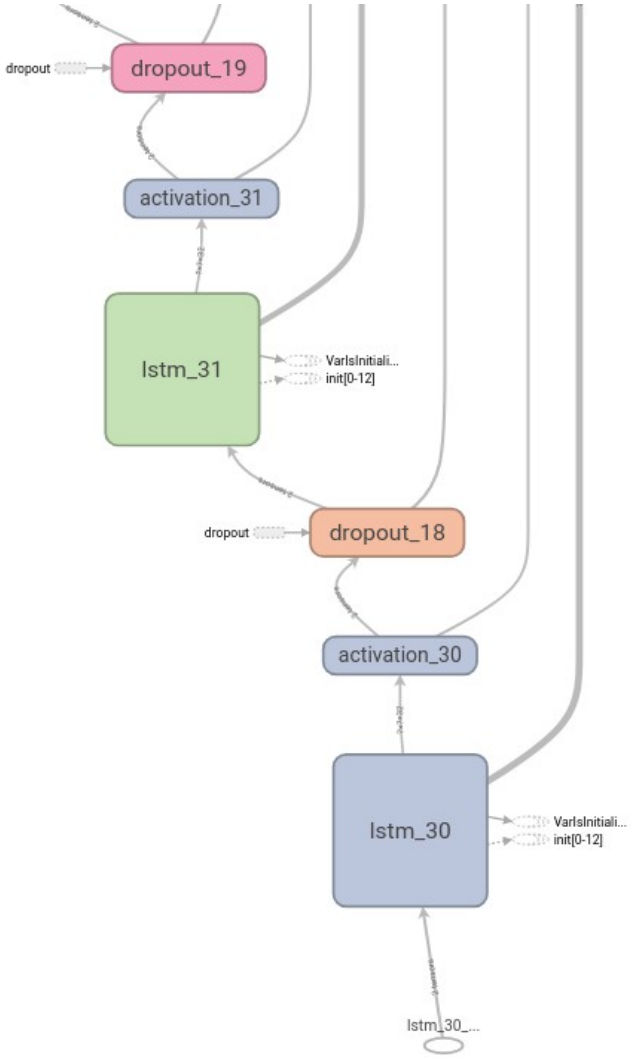
Slika 3.2: Koda za zagon nevronske mreže



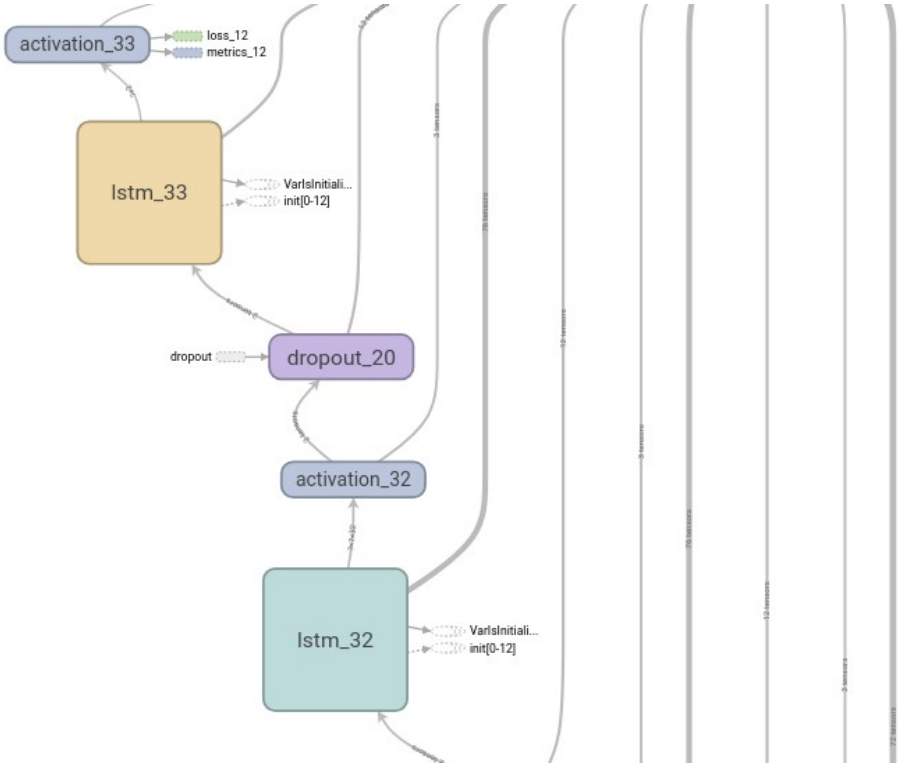
Slika 3.3: Vizualiziran model nevronske mreže

Keras nam omogoča tudi orodje za vizualiziranje modela mreže, imenovan TensorBoard. Ta nam tudi omogoča vpogled v grafe natančnosti in izgube za analiziranje rezultatov.

Na sliki 3.3 vidimo vizualiziran model programske kode iz slike 3.1. Na sliki 3.4 vidimo vhodno in prvo skrito plast celic LSTM. Podatki potujejo po mreži navzgor in potem naprej v naslednji dve plasti, ki jih vidimo na sliki 3.5. Na vsaki plasti pa se zgodi tudi evaluiranje podatkov, zato so vsi povezani na vozlišče za treniranje, ki ga vidimo na sliki 3.6.



Slika 3.4: Prvi dve plasti nevronske mreže



Slika 3.5: Zadnji dve plasti nevronske mreže

Slika 3.6: Treniranje nevronske mreže

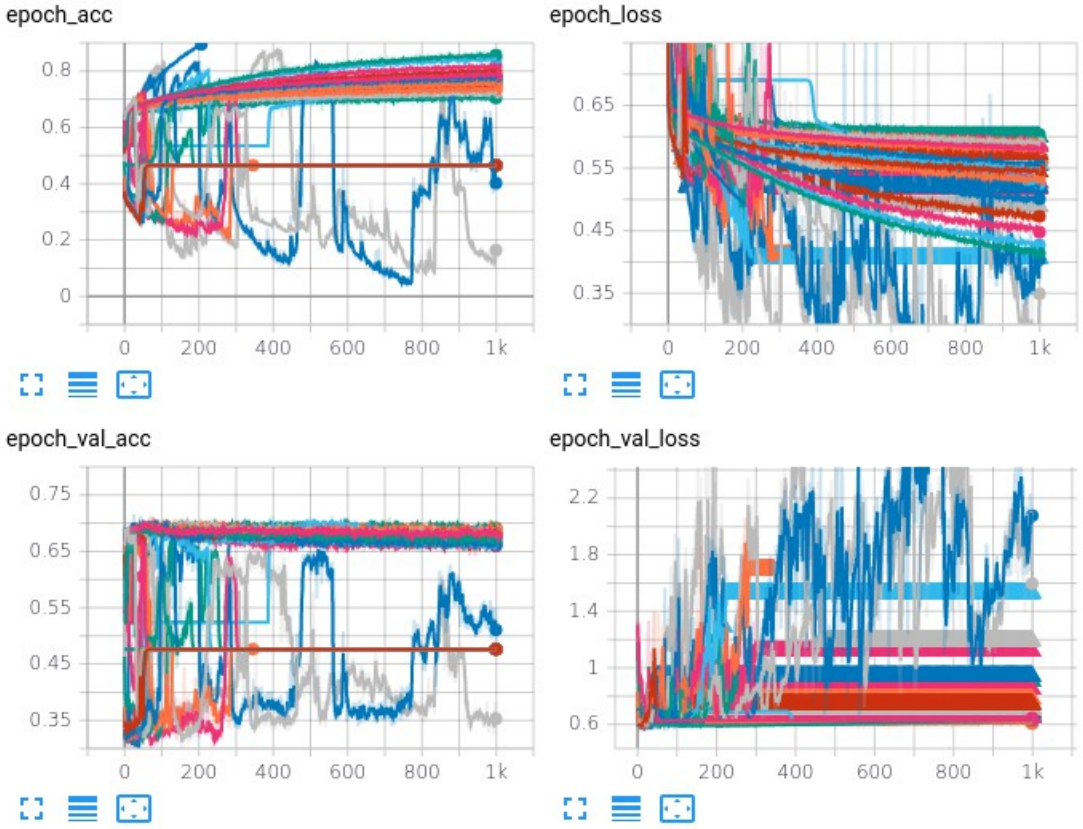
Izbiro oblike modela, torej število plasti in nevronov ter ostale parametre, smo določili s poskušanjem, saj je vsak nabor podatkov problem zase in ne obstaja neka univerzalna rešitev. Zato moramo preizkusiti veliko različnih modelov, da vidimo, kje je natančnost najboljša in izguba najmanjša ter da ne prihaja do prekomernega prileganja, torej, da se mreža dobro nauči le na primeru množice za učenje, v praksi pa potem dosega

slabše rezultate na testni množici. Testiranje tega bi samo po sebi vzelo veliko časa, zato smo napisali kratko skripto, ki jo vidimo na sliki 3.7. Tudi sama skripta porabi kar nekaj ur za izvajanje, da gre skozi vse možnosti, ampak je ne rabimo sproti spremljati in na koncu samo preučimo generirane grafe TensorBoard-a ter vzamemo najboljšega izmed vseh. Za testiranje smo si izbrali angleško ligo, več o podatkih pa v poglavju 3.2.

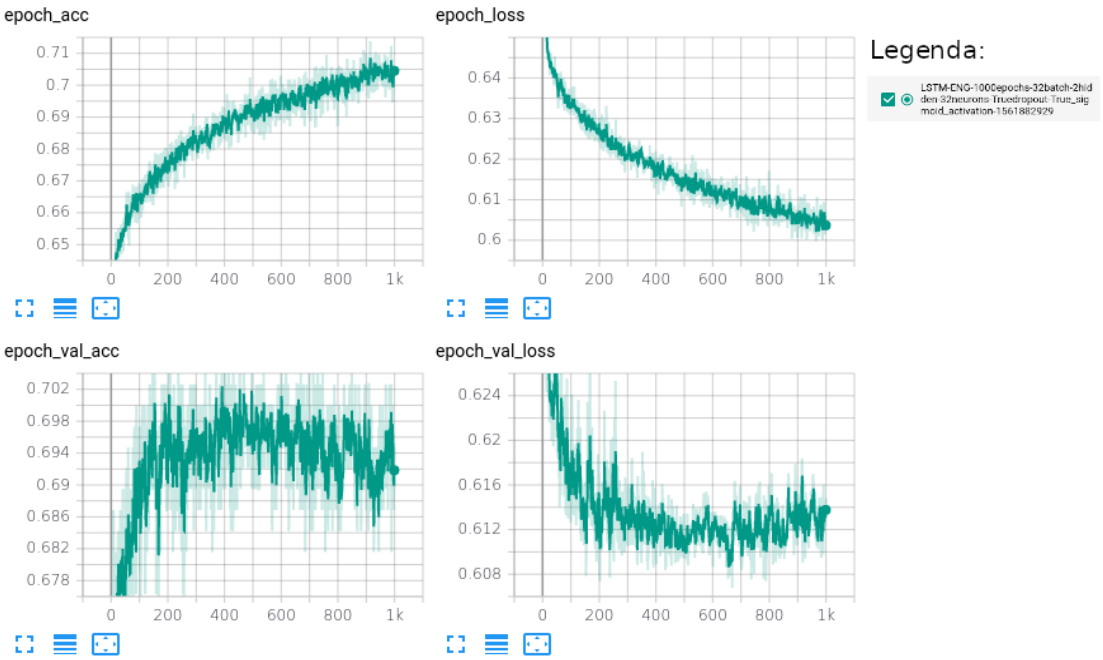
```
for epochs in [200, 400, 600, 800, 1000]:
    for num_hidden_layers in  [0, 1, 2]:
        for num_of_neurons in [32, x_train.shape[1:][1], 128]:
            for activation in ['sigmoid', 'tanh', 'softmax', 'relu']:
                LOG_NAME = "LSTM-ENG-{}epochs-{}batch-{}hidden-{}neurons-Truedropout-True_{}_activation-{}".format(
                        epochs, batch_size, num_hidden_layers, num_of_neurons, activation, int(time.time()))

                print(LOG_NAME)

                model = Sequential()

                model.add(LSTM(num_of_neurons,
                        input_shape=(None, x_train.shape[1:][1],),
                        return_sequences=True))
                model.add(Activation(activation))
                model.add(Dropout(0.2))

                for i in range(num_hidden_layers):
                    model.add(LSTM(num_of_neurons,
                            return_sequences=True))
                    model.add(Activation(activation))
                    model.add(Dropout(0.2))

                model.add(LSTM(2))
                model.add(Activation(activation))

                model.compile(loss="categorical_crossentropy",
                        optimizer="adam",
                        metrics=["accuracy"])

                callbacks = [TensorBoard(log_dir=loc + 'logs/{}'.format(LOG_NAME))]

                model.fit(x_train, y_train, epochs=epochs,
                        callbacks=callbacks, batch_size=32,
                        validation_data=(x_test, y_test))
```

Slika 3.7: Koda za ugotavljanje velikosti in parametrov nevronske mreže

Sedaj imamo generiranih veliko grafov, ki jih vidimo na sliki 3.8. Če jih podrobneje pogledamo, hitro ugotovimo, da vse aktivacijske funkcije niso primerne, najbolje sta se izkazali sigmoid in softmax. Najboljši graf skozi 1000 epoh je graf iz slike 3.9, katerega vizualizirano nevronsko mrežo lahko vidimo na sliki 3.3 in kodo na sliki 3.1.



Slika 3.8: Generirani Tensorboard grafi



Slika 3.9: Najboljši izmed generiranih grafov

3.2 Pridobivanje podatkov

Zgodovinske podatke o največjih svetovnih nogometnih ligah je možno najti dokaj hitro. Na spletni strani [11] najdemo CSV (ang. Comma separated values, vrednosti ločene z vejico) datoteke za vse sezone določene lige, vse od začetka 90. let. Te datoteke vsebujejo mnogo podatkov, večina teh, pa ne pride v poštev pri tej nalogi. Izmed vseh podatkov smo uporabili le imena ekip, končen rezultat in kvote iz določenih stavnic, medtem ko so še na voljo informacije o kotih, prepovedanih položajih, kartonih itd. Vzeli smo podatke o angleški, nemški, italijanski, francoski in španski prvi ligi.

Za slovensko prvo ligo pa je podatke v že shranjeni obliki žal težje najti, je pa celotna zgodovina javno dostopna na wikipediji [12] in uradni strani Prve slovenske nogometne lige [13]. Zato smo morali napisati program [14], ki je znal iz teh spletnih strani razbrati podatke, ki smo jih lahko potem aplicirali na naš model. Ta je iz spletne strani [13] ustvaril dve CSV datoteki. V eni so podatki o končni lestvici za vsako sezono, za katero kodo vidimo na sliki 3.10, v drugi pa naslednje informacije o vsaki tekmi:

- datum,
- ime domače ekipe,
- ime tuje ekipe,
- število zadetkov domače ekipe,
- število zadetkov tuje ekipe,
- oznaka za zmagovalca (domača ekipa = H, tuja ekipa = A, izenačen rezultat = D).

```python
def main():
    data = []
    URL = "https://www.prvaliga.si/tekmovanja/default.asp?action=lestvica&id_menu=102&id_sezone="
    from_year = 1992
    to_year = 2019

    seasons = pd.read_csv("output/slo_standings.csv")

    for i in range(from_year, to_year + 1):
        print("Parsing year: " + str(i))
        r = requests.get(URL + str(i))
        r.encoding = 'utf-8'
        soup = BeautifulSoup(r.text, "html.parser")

        # Select all teams
        league_table = soup.find("table", {"class": "Tabela1"}).find_all("tr")

        for j, row in enumerate(league_table):
            if j != 0:
                # Get team position
                position = row.findNext("td").get_text().strip()[:-1]

                # Get team name
                team_td = row.find("td", {"class": "title"})
                if team_td != None:
                    team = team_td.get_text().strip()
                    if team[0] == "*":
                        team = team[1:].strip()

                    team_name = teams_dict.get(team)
                    if team_name != None:
                        team = team_name

                    if not seasons.loc[seasons["Team"] == team].empty:
                        seasons.loc[seasons["Team"] == team, ((i-1) % 100)] = int(position)
                    else:
                        raise Exception("Not found team: " + team)

        # Save seasons standings
        seasons.to_csv("output/slo_standings.csv", index=None)
```

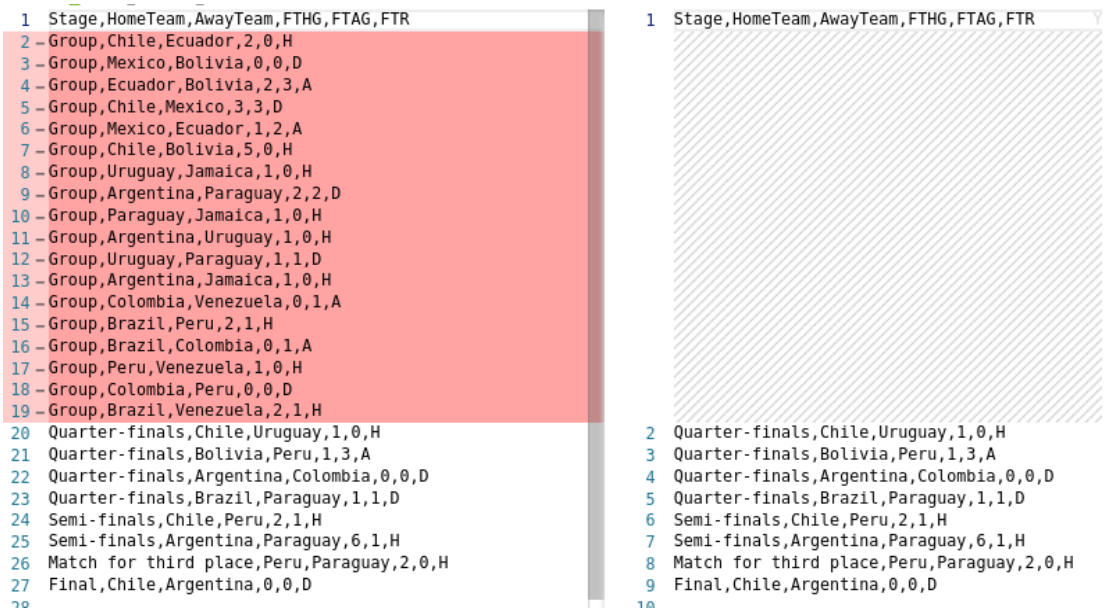
Slika 3.10: Koda za shranjevanje končne lestvice vsake sezone slovenske prve lige v CSV datoteko

Tabela 3.1: Primer spreminjanja imena nogometnega kluba Celje

Ime ekipe
CM Celje
MIK CM Celje
CMC Publikum
Publikum
Protonavto Publikum
Biostart Publikum

Največja težava pri tem je bila, da so se imena klubov skozi leta veliko spreminjala in je bilo potrebno kar nekaj ročnega dela, da smo povezali skupaj prava imena klubov. Navajamo primer za ekipo Celja, ki ga lahko vidimo v tabeli 3.1.

Za turnirski del podatkov pa smo vzeli svetovno prvenstvo in Copo Americo. CSV datoteka za svetovno prvenstvo je bila na voljo na spletu [15] in je potrebovala le minimalne popravke, medtem ko smo za Copo Americo zopet morali napisati svoj program [16], ki je viden na sliki 3.12. Ta je iz več strani wikipedije dobil podatke za skoraj vse tekme in jih vrnil v skoraj isti obliki kot program [14], z razliko, da smo datum zamenjali s fazo tekmovanja (skupinski del = Group, četrt finale = Quater-finals, pol finale = Semi-finals, tekma za tretje mesto = Match for third place, finale = Final). Določene tekme pa smo morali zaradi nekonsistentnosti spletne kode wikipedije vnesti sami, primer lahko vidimo na sliki 3.11.



Slika 3.11: Primer manjkajočih tekem tekmovanja Copa America

```python
def main():
    URL = "https://en.wikipedia.org/wiki/{}_Copa_Am%C3%A9rica"
    # Format [year, number of teams, normal page]
    years = [[1993, 12, True], [1995, 12, True], [1997, 12, True], [1999, 12, True], [2001, 12, True],
             [2004, 12, True], [2007, 12, True], [2011, 12, False], [2015, 12, False], [2016, 16, True]]

    for year in years:
        print("Parsing year: {}".format(year[0]))
        r = requests.get(URL.format(year[0]))
        r.encoding = 'utf-8'
        soup = BeautifulSoup(r.text, "html.parser")

        # Get group data
        games = soup.find_all("div", {"class": "footballbox"})
        if year[2]: # Some pages differ from others and are harder to parse
            num_of_group_games = year[1] * 6 / 4
        else:
            num_of_group_games = 0
        data = []
        for i, game in enumerate(games):
            if i < num_of_group_games:
                stage = "Group"
            elif i < num_of_group_games + 4:
                stage = "Quarter-finals"
            elif i < num_of_group_games + 4 + 2:
                stage = "Semi-finals"
            elif i < num_of_group_games + 4 + 2 + 1:
                stage = "Match for third place"
            else:
                stage = "Final"

            home_team = game.find('th', {'class': 'fhome'}).findNext('a').get_text()
            away_team = game.find('th', {'class': 'faway'}).findNext('a').get_text()

            goals = game.find('th', {'class': 'fscore'}).get_text().split('–')
            home_team_goals = re.findall('\d+', goals[0])[0]
            away_team_goals = re.findall('\d+', goals[1])[0]

            ftr = "D" if home_team_goals == away_team_goals else ("H" if home_team_goals > away_team_goals else "A")

            data.append([stage, home_team, away_team, home_team_goals, away_team_goals, ftr])

        # Save current year
        df = pd.DataFrame(data, columns = ["Stage","HomeTeam","AwayTeam","FTHG","FTAG","FTR"])
        df.to_csv("output/copa_america_" + str(year[0]) + ".csv", index=None)
```

Slika 3.12: Koda za shranjevanje tekem tekmovanja Copa America v CSV datoteko

Iz teh vhodnih podatkov smo potem v fazi predprocesiranja pridobili nove. Za vsako tekmo smo si izračunali število prejetih in danih golov (slika 3.13) ter točke iz prejšnjih tekem sezone (slika 3.14), formo za zadnjih 5 tekem (slika 3.15), kateri teden v ligi je in pa razlike med goli ter točkami obeh ekip. Na sliki 3.16 lahko vidimo grafični prikaz tega, v tabeli 3.2 pa je opis vseh podatkov, ki jih pošljemo v mrežo.

```
def getGoalsScored(season):
    # Create a dictionary with team names as keys
    teams = {}

    home_teams = list(season.groupby('HomeTeam').mean().T.columns)
    away_teams = list(season.groupby('AwayTeam').mean().T.columns)
    all_teams = home_teams + list(set(away_teams) - set(home_teams))

    for i in all_teams:
        teams[i] = []

    for i in range(len(season)):
        # Get home and away goals
        HTGS = season.iloc[i]['FTHG']
        ATGS = season.iloc[i]['FTAG']

        # Append to correct team
        teams[season.iloc[i].HomeTeam].append(HTGS)
        teams[season.iloc[i].AwayTeam].append(ATGS)

    # Make all teams go up to the same lenght
    max_len = 0
    for key, value in teams.items():
        if len(value) > max_len:
            max_len = len(value)

    for key,value in teams.items():
        if len(value) < max_len:
            for i in range(len(value), max_len):
                value.append(np.nan)

    # Create dataframe
    GoalsScored = pd.DataFrame(data=teams, index = [i for i in range(1,max_len + 1)]).T

    # Seasons starts at 0 goals
    GoalsScored[0] = 0

    # Get goals up to some point
    for i in range(2,max_len + 1):
        GoalsScored[i] = GoalsScored[i] + GoalsScored[i-1]

    return GoalsScored
```

Slika 3.13: Koda za izračun prejetih golov vsake tekme v sezoni

```python
def getPoints(result):
    if result == "W":
        return 3
    elif result == "D":
        return 1
    else:
        return 0


def getSeasonPoints(season, num_of_teams):
    matchres, lenght = getMatchResults(season)  # Get match results
    matchres_points = matchres.applymap(getPoints)

    for i in range(2, lenght + 1):
        matchres_points[i] = matchres_points[i] + matchres_points[i-1]

    matchres_points.insert(column=0, loc=0, value=[0*i for i in range(num_of_teams)])
    return matchres_points


def getMatchResults(season):
    teams = {}  # Create a dictionary with team names as keys

    home_teams = list(season.groupby('HomeTeam').mean().T.columns)
    away_teams = list(season.groupby('AwayTeam').mean().T.columns)
    all_teams = home_teams + list(set(away_teams) - set(home_teams))

    for i in all_teams:
        teams[i] = []

    for i in range(len(season)):
        if season.iloc[i].FTR == 'H':
            teams[season.iloc[i].HomeTeam].append('W')
            teams[season.iloc[i].AwayTeam].append('L')
        elif season.iloc[i].FTR == 'A':
            teams[season.iloc[i].AwayTeam].append('W')
            teams[season.iloc[i].HomeTeam].append('L')
        else:
            teams[season.iloc[i].AwayTeam].append('D')
            teams[season.iloc[i].HomeTeam].append('D')

    max_len = 0
    for key, value in teams.items():
        if len(value) > max_len:
            max_len = len(value)

    for key, value in teams.items():
        if len(value) < max_len:
            for i in range(len(value), max_len):
                value.append('N')

    return pd.DataFrame(data=teams, index = [i for i in range(1,max_len + 1)]).T, max_len
```

Slika 3.14: Koda za izračun točk ekip vsake tekme v sezoni

```python
def getForm(season, num):
    form, length = getMatchResults(season)
    form_final = form.copy()
    for i in range(num, length + 1):
        form_final[i] = ""
        j = 0
        while j < num:
            form_final[i] += form[i-j]
            j += 1

    return form_final


def addForm(season, num, games_per_week):
    form = getForm(season, num)

    # Since form is not available for first match week
    h = ["M" for i in range(num * games_per_week)]
    a = ["M" for i in range(num * games_per_week)]

    j = num
    for i in range((num * games_per_week), len(season)):
        ht = season.iloc[i].HomeTeam
        at = season.iloc[i].AwayTeam

        past = form.loc[ht][j]  # Get past n results
        h.append(past[num - 1])

        past = form.loc[at][j]
        a.append(past[num - 1])

        if ((i + 1) % games_per_week) == 0:
            j += 1

    season["HomeForm"+str(num)] = h[:season.shape[0]]
    season["AwayForm"+str(num)] = a[:season.shape[0]]

    return season
```

Slika 3.15: Koda za določanje forme za nekaj tekem nazaj

WHD	WHA	HTGS	ATGS	HTGC	ATGC	HTP	ATP	HW	HD
5.00	6.00	12	11	9	6	12	13	4	0
3.50	2.45	14	14	4	2	16	18	5	1
4.00	4.33	8	8	11	13	6	5	1	3
4.40	1.47	3	12	14	7	2	12	0	2
12.00	29.00	19	8	3	11	16	5	5	1
3.20	2.90	4	11	8	10	2	9	0	2
3.70	1.85	5	9	11	9	4	10	1	1
3.50	4.50	6	6	6	9	9	5	2	3
3.10	3.00	3	7	14	10	2	4	0	2
3.40	3.20	10	4	11	6	10	7	3	1
3.20	2.80	8	8	13	12	5	7	1	2
3.00	4.00	9	3	11	16	7	2	2	1
3.20	2.55	5	8	8	6	7	12	2	1
3.40	3.30	13	11	10	11	12	9	4	0

Slika 3.16: Vhodni podatki za tekme

Preden pošljemo podatke iz tabele 3.2 v nevronsko mrežo, moramo še spremeniti tekst v numerične vrednosti, saj mreža razume le takšno obliko podatkov. Zato uporabimo t.i. one-hot kodiranje, ki vsak tekst enolično določi. Najprej storimo to za ekipe, če ta ni igrala, je imela vrednost 0, če je igrala doma, je imela vrednost 0.5, če pa je igrala zunaj pa 1, primer lahko vidimo na sliki 3.17. Isto kodiranje uporabimo tudi za rezultat, če zmaga domača ekipa dobi stolpec H vrednost 1, stolpec NH pa vrednost 0, v primeru neodločenega izida ali poraza domače ekipa pa obratno. Enako naredimo tudi pri formi ekip z razliko, da imamo pri formi štiri vrednosti namesto dveh.

Leeds	Leicester	Liverpool	Man City	Man United	Middlesboro
0.0	0.0	0.5	1.0	0.0	0.0

Slika 3.17: Primer kodiranja ekip tekme Liverpool proti Manchester City

Tabela 3.2: Vhodni podatki

Vhodni podatek	**Razlaga**
Kvota stavnice Bet365	3 podatki: kvota za zmago domače ekipe, kvota za neodločen izid in kvota za zmago tuje ekipe
Število zadetih golov	Koliko golov je zadela vsaka ekipa do tega trenutka v sezoni
Število prejetih golov	Koliko golov je prejela vsaka ekipa do tega trenutka v sezoni
Točke	Koliko točk imata obe ekipi v tem trenutku sezone
Pregled statistike tekem	Število zmag, remijev in porazov obeh ekip do tega trenutka v sezoni
Forma za zadnjih 5 tekem	Za vsako izmed zadnjih 5 tekem ali je bila zmaga (W), remi (D), poraz (L) ali pa še ni bila odigrana (M)
Igralen teden	Kateri zaporedni teden sezone je
Niz zmag	Ali je ekipa zadnje 3 ali 5 tekem zmagala
Niz porazov	Ali je ekipa zadnje 3 ali 5 tekem izgubila
Razlika v golih	Število zadetih golov – število prejetih golov
Razlika v točkah	Točke domače ekipe – točke tuje ekipe
Razlika v lanskih položajih na lestvici	Lanski končni položaj prve ekipe – lanski končni položaj druge ekipe (če katera od ekip v prejšnji sezoni ni igrala v tej ligi, se ji dodeli pozicija 30, tako da je razlika od ostalih)
Ekipa	Ime ekipe
Rezultat	H – zmaga domače ekipe, NH – neodločeno ali poraz domače ekipe

Rezultati se lahko razlikujejo glede na to, kaj želimo napovedati, kar vidimo na sliki 3.18. Če napovedujemo zmagovalca, imamo stolpca H in NH. Če napovedujemo gole, ali bodo zadeli več kot je določeno število golov in je to res, ima stolpec M vrednost 1, drugače pa stolpec L. Če pa napovedujemo, ali obe ekipi zadeneta in je to res, ima stolpec T vrednost 1, drugače pa stolpec F.

a)

H	NH
1.0	0.0
1.0	0.0
0.0	1.0
0.0	1.0
0.0	1.0
1.0	0.0

b)

L	M
0.0	1.0
1.0	0.0
1.0	0.0
0.0	1.0
0.0	1.0
1.0	0.0

c)

F	T
0.0	1.0
1.0	0.0
1.0	0.0
1.0	0.0
0.0	1.0
1.0	0.0

Slika 3.18: Primeri rezultatov

4 REZULTATI

Rezultate bomo pogledali iz več perspektiv. Začnimo z ligaškim delom in primerjajmo uspešnost napovedovanja zmagovalca po različnih ligah, kasneje pa si še bomo pogledali turnirske načine igranja. Za napovedovanje smo uporabili nevronsko mrežo iz slike 3.1.

Ker bomo opazovali generirane Tensorboard grafe, moramo najprej obrazložiti določene kratice, ki so uporabljene:

- epoch_acc – uspešnost napovedovanja na učni množici (delež od 0 do 1, želimo čim bližje 1);
- epoch_loss – izguba pri napovedovanju učne množice (vrednosti večje od 0, želimo pa, da so čim bližje 0);
- epoch_val_acc – uspešnost napovedovanja na testni množici (delež od 0 do 1, želimo čim bližje 1);
- epoch_val_loss – izguba pri napovedovanju na testni množici (vrednosti večje od 0, želimo pa, da so čim bližje 0).

Čas napovedovanja v tabelah pomeni skupen čas za nalaganje datoteke, za dodatno preoblikovanje teh podatkov, da so primerni za vhod v mrežo in napovedovanje celotne sezone oziroma celotnega tekmovanja. Čas napovedovanja za eno samo tekmo pa je okoli 0,003 sekunde.

Za zaganjanje uporabljamo računalnik s procesorjem Intel i5-4750 3,6GHz, grafično kartico NVIDIA GeForce GTX 960 in 8 GB RAM pomnilnika.

4.1 Napovedovanje tekem ligaške sezone

Nevronsko mrežo bomo naučili na podatkih do zadnje sezone, zadnjo sezono pa bomo imeli kot validacijsko množico, da preverjamo natančnost napovedovanja za nove podatke. Čas merimo za 1000 epoh, ampak imamo nastavljen tudi limit, da se v primeru, če se kadarkoli v roku 200 epoh odstotek pravilno napovedanih ne izboljša, učenje ustavi. Uporabljeno nevronsko mrežo smo testirali tudi z manjšim številom plasti in z uporabo celice GRU namesto celice LSTM.

Lige se med seboj razlikujejo le po številu ekip, tako da so vhodni podatki pri vseh enaki, razen za slovensko ligo nimamo informacije o kvotah iz stavnic.

Tabela 4.1: Uspešnost napovedovanja zmagovalca tekme glede na različno število plasti

	Mreža			
	2 plasti, celica LSTM		**4 plasti, celica LSTM**	
Liga	**Uspešnost napovedi (v odstotkih)**	**Čas učenja (v sekundah)**	**Uspešnost napovedi (v odstotkih)**	**Čas učenja (v sekundah)**
Italijanska liga	70,97	88	**71,84**	233
Angleška liga	70,53	102	**70,79**	218
Nemška liga	67,65	62	**69,93**	210
Španska liga	65,26	72	**66,05**	123
Francoska liga	**65,53**	139	63,68	147
Slovenska liga	61,11	66	**61,67**	116

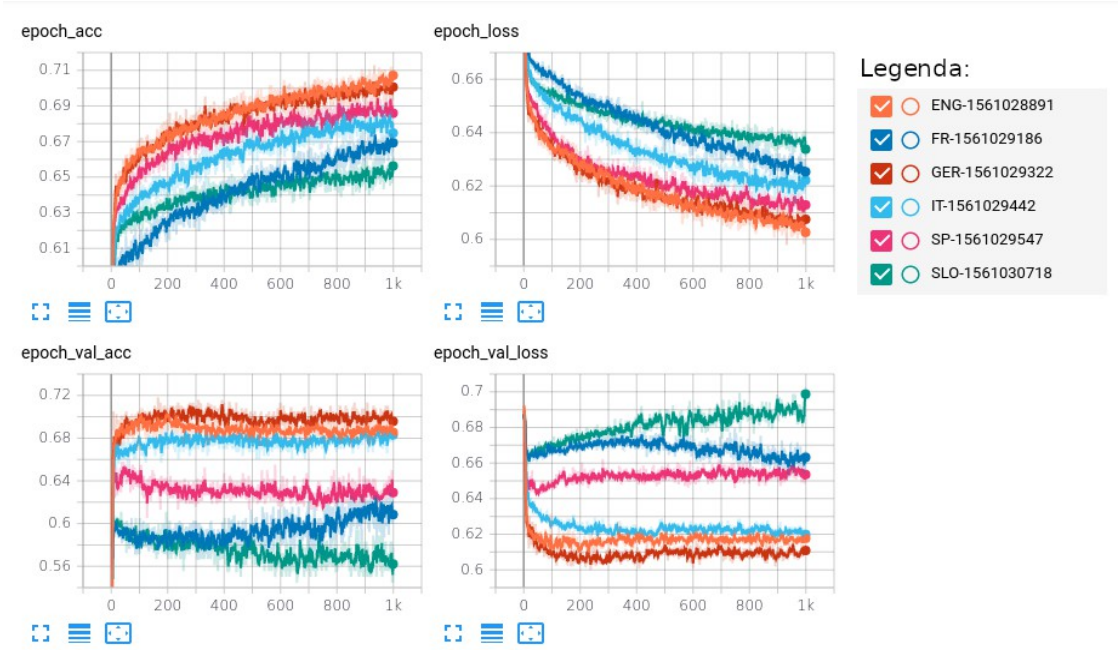
Tabela 4.2: Uspešnost napovedovanja zmagovalca tekme glede na različen tip celice

	Mreža			
	4 plasti, celica GRU		**4 plasti, celica LSTM**	
Liga	**Uspešnost napovedi (v odstotkih)**	**Čas učenja (v sekundah)**	**Uspešnost napovedi (v odstotkih)**	**Čas učenja (v sekundah)**
Italijanska liga	71,58	193	**71,84**	233
Angleška liga	70,53	118	**70,79**	218
Nemška liga	68,95	93	**69,93**	210
Španska liga	**66,32**	100	66,05	123
Francoska liga	**64,47**	223	63,68	147
Slovenska liga	**61,67**	94	**61,67**	116

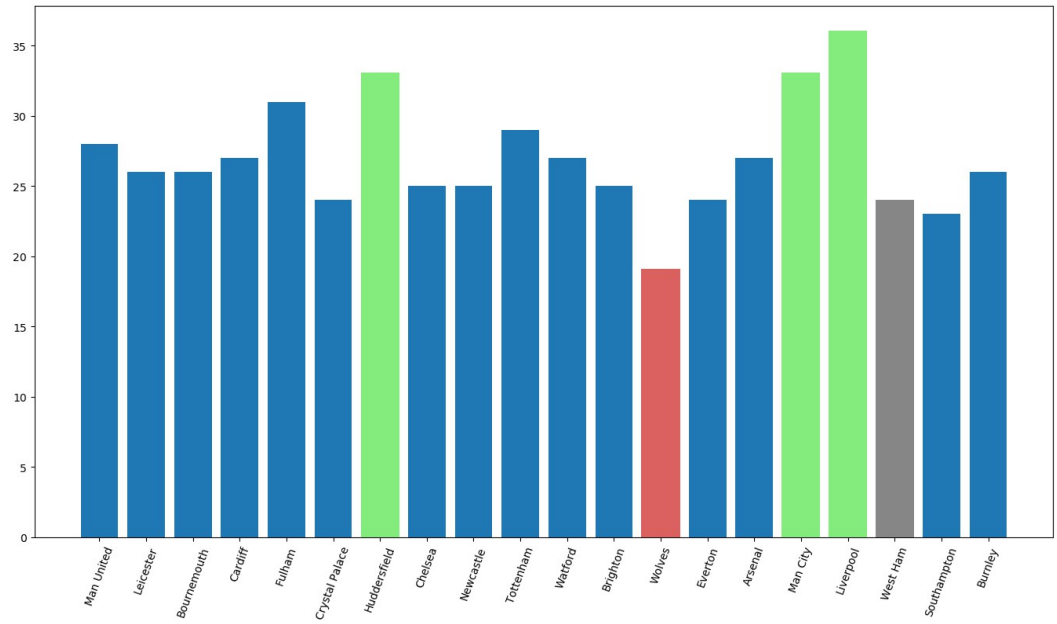
Rezultate skozi epohe vidimo na sliki 4.1, uspešnost pa v tabelah 4.1 in 4.2, kjer lahko vidimo za primerjavo tudi natančnost na mreži z dvema namesto štirimi plastmi in celice GRU namesto celice LSTM. Vidimo, da se nevronska mreža z manj plastmi slabše obnese, je pa manj časovno zahtevna. Med celico GRU in celico LSTM pa ne vidimo večjih razlik v napovedovanju, so pa razlike v času učenja, in sicer se zaradi manjše zapletenosti celica GRU načeloma uči hitreje. Ker za slovensko ligo nimamo podatkov za kvote na stavnicah in vidimo slabšo napovedljivost, lahko sklepamo, da so kvote pomemben faktor pri učenju nevronske mreže. Ostale lige pa imajo iste podatke na vhodu, kar pomeni, da sta italijanska in angleška liga manj naključni, kot pa francoska in španska, nemška pa je nekje vmes.

Prihaja pa tudi do razlik med ekipami, saj se rezultate določene ekipe lažje napove kot druge, kar vidimo na sliki 4.2. Najbolje se napoveduje Liverpool (36 od 38 pravilnih), Manchester City (33 od 38 pravilnih) in pa Huddersfield (33 od 38 pravilnih), ki so označeni z zeleno. Če pogledamo lestvico, so to prva, druga in pa zadnja ekipa v ligi, torej ji večje probleme delajo ekipe iz sredine lestvice, kar pa je tudi logično, saj so te manj konsistentne. Najslabše napovedljiv je Wolves, ki je označen z rdečo, in sicer le 19 od 38 tekem pravilnih. Če pa pogledamo ekipo iz sredine lestvice (10 mesto), West

Ham, ki je označen s sivo in je pravilno napovedanih 24 od 38 tekem.



Slika 4.1: Grafi natančnosti in izgub za napovedovanje zmagovalca tekme v ligaškem tekmovanju



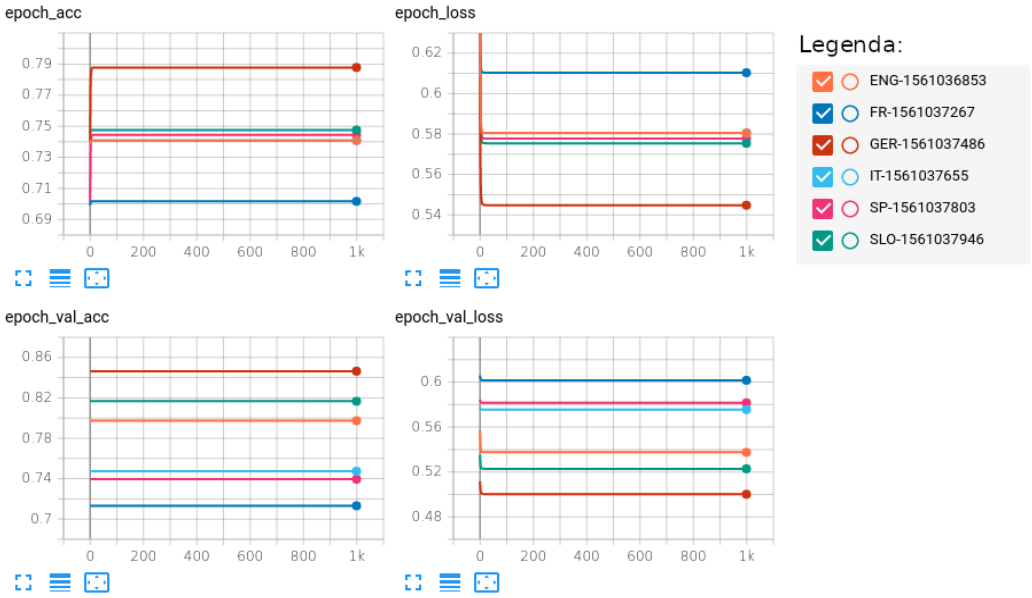
Slika 4.2: Uspešnost napovedovanja za posamezno ekipo znotraj zadnje sezone angleške lige

Če pogledamo vse skupaj iz perspektive stavnice, ugotovimo, da zmagovalec ni edina stvar, na katero lahko stavimo, zato poskušamo še nekatere druge. Skoraj vedno so prisotne tudi stave, ali pade na tekmo več kot 1,5 ali 2,5 gola in če obe ekipi zadeneta. Začnimo z goli, ali na tekmo pade več kot 1,5 gola. Kar se tiče vhoda v mrežo je enak, popraviti moramo le rezultat, in sicer tam ko smo prej imeli H in NH, sedaj zamenjamo z M, ko pade več in L, ko pade manj kot 1,5 gola, kar je razvidno iz slike 3.18.

Tabela 4.3: Uspešnost napovedovanja, ali pade več kot 1,5 gola na tekmo

Liga	**Uspešnost napovedi (v odstotkih)**	**Čas učenja (v sekundah)**	**Čas napovedovanja (v sekundah)**
Nemška liga	84,64	120	8
Slovenska liga	81,67	113	8
Angleška liga	79,74	109	8
Italijanska liga	74,74	108	8
Španska liga	73,95	113	8
Francoska liga	71,23	119	8

Napovedljivost tega je precej boljša kot pa napovedljivost zmagovalca, kar lahko vidimo v tabeli 4.3. Iz slike 4.3 lahko vidimo, da nevronska mreža v trenutku ugotovi napovedljivost in tako pride do nekega lokalnega minimuma ter se kasneje ne spreminja na bolje. Če pogledamo dejanske napovedi mreže, lahko vidimo, da v večini primerov pride do spoznanja, da skoraj vedno pade več kot 1,5 gola na tekmo. Torej so te natančnosti bolj ali manj enake dejanskemu odstotkovnemu številu tekem, ko pade več kot 1,5 gola na tekmo.



Slika 4.3: Grafi natančnosti in izgub za napovedovanje, ali pade več kot 1,5 gola na tekmo v ligaškem tekmovanju

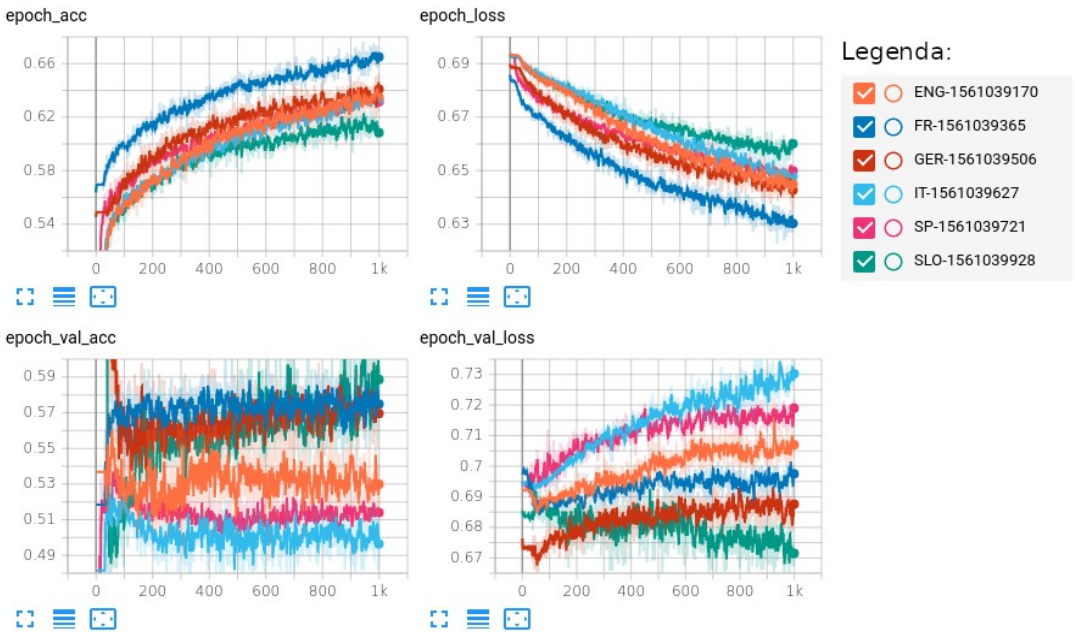
Sedaj pa primerjajmo te rezultate s tem, ali pade več kot 2,5 gola na tekmo. Vhod je isti kot pri prejšnjem primeru, le da moramo popraviti rezultate.

Tabela 4.4: Uspešnost napovedovanja, ali pade več kot 2,5 gola na tekmo

Liga	**Uspešnost napovedi (v odstotkih)**	**Čas učenja (v sekundah)**	**Čas napovedovanja (v sekundah)**
Nemška liga	63,73	119	8
Slovenska liga	62,78	110	8
Francoska liga	60,00	253	8
Angleška liga	57,63	148	8
Španska liga	55,00	170	8
Italijanska liga	53,68	146	8

Kot lahko vidimo na sliki 4.4 in tabeli 4.4 je napovedljivost slaba. Ni enak problem kot pri več kot 1,5 gola na tekmo, saj tu ne pride do rešitve, da bi naj to bilo vedno res, ampak iz teh vhodnih podatkov ne moremo razbrati nekega vzorca, kdaj bi temu bilo

tako.



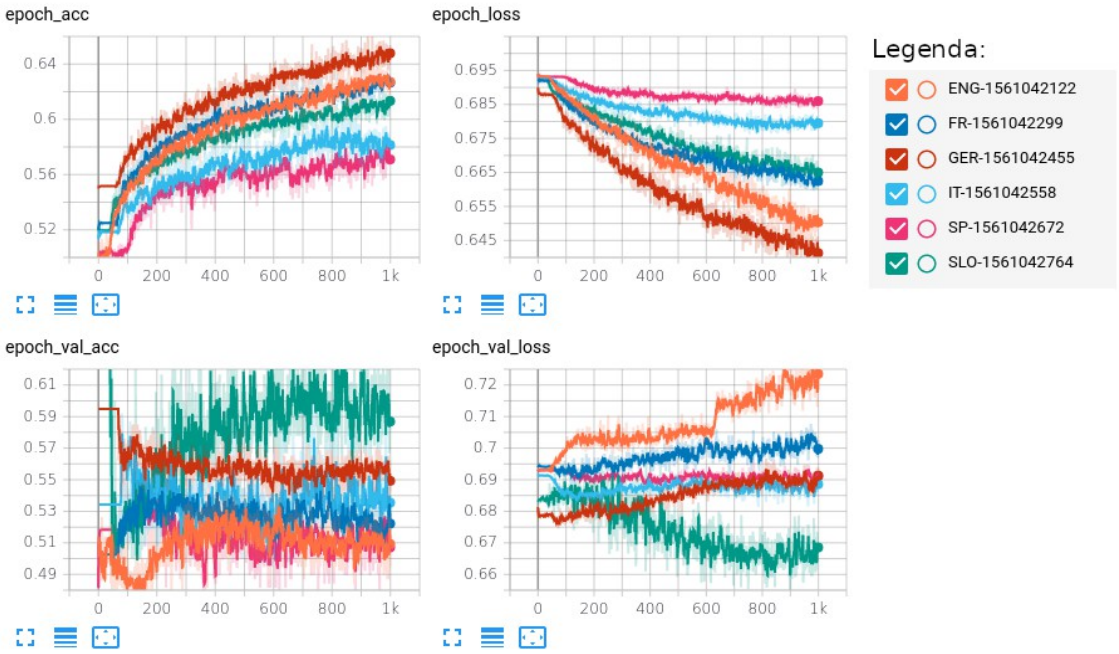
Slika 4.4: Grafi natančnosti in izgub za napovedovanje, ali pade več kot 2,5 gola na tekmo v ligaškem tekmovanju

Poglejmo še, ali lahko napovemo, če bosta obe ekipi zadeli vsaj 1 gol. Zopet isti vhod kot pri prejšnjem primeru, le ko obe ekipi zadeneta, damo kot rezultat T, ko pa ne pa F, kar je razvidno na sliki 3.18.

Tabela 4.5: Uspešnost napovedovanja, ali obe ekipi na tekmi zadeneta vsaj en gol

Liga	**Uspešnost napovedi (v odstotkih)**	**Čas učenja (v sekundah)**	**Čas napovedovanja (v sekundah)**
Slovenska liga	66,11	108	7
Italijanska liga	62.37	266	8
Nemška liga	59,48	142	8
Španska liga	57,89	168	8
Francoska liga	55,79	173	8
Angleška liga	54,74	202	8

Slika 4.5 prikazuje podobno zgodbo kot slika 4.4, tudi rezultati so si podobni, kar lahko vidimo v tabeli 4.5.



Slika 4.5: Grafi natančnosti in izgub za napovedovanje, ali obe ekipi zadeneta vsaj po en gol v ligaškem tekmovnaju

4.2 Napovedovanje tekem nogometnih turnirjev

Malo drugačen sistem igranja pa se pojavlja na večjih nogometnih turnirjih kot je recimo svetovno prvenstvo, saj se igranje deli na dva dela, to sta skupinski in izločevalni del. Tega problema bi se lahko lotili na dva načina, in sicer tako, da poskušamo skupinski in izločevalen del napovedati vsakega posebej in drugi, za katerega smo se tudi odločili, torej, da vzamemo vse podatke skupaj. Nevronsko mrežo bomo naučili vse do zadnjega turnirja in le-tega potem poskušali napovedati. Ker imamo pri tekmovanjih manj podatkov čas merimo za 10000 epoh in nastavljen imamo limit, če se kdarkoli v roku 2000 epoh odstotek pravilno napovedanih ne izboljša, se učenje ustavi.

Na svetovnem prvenstvu sodeluje 32 držav, ki so razporejene v 8 skupin. V skupini igra

vsaka država z vsako le enkrat. Iz vsake skupine napredujeta v izločevalni del dve najboljši ekipi. V tem delu, če izgubiš tekmo, izpadeš iz tekmovanja, če pa zmagaš, pa napreduješ naprej. Vendar pa pred letom 1998 temu ni bilo tako, saj je sodelovalo manjše število držav. Svetovno prvenstvo se igra na vsaka 4 leta, začelo pa se je leta 1930.

Copa America ima isti sistem tekmovanja, s tem da ima 3 ali pa 4 skupine. Če ima 4 skupine, gresta iz vsake skupine naprej dve državi, če pa ima 3 pa iz dveh skupin napredujejo 3, iz ene pa 2 državi. Začelo se je leta 1993 in od takrat dalje je potekalo 10-krat.

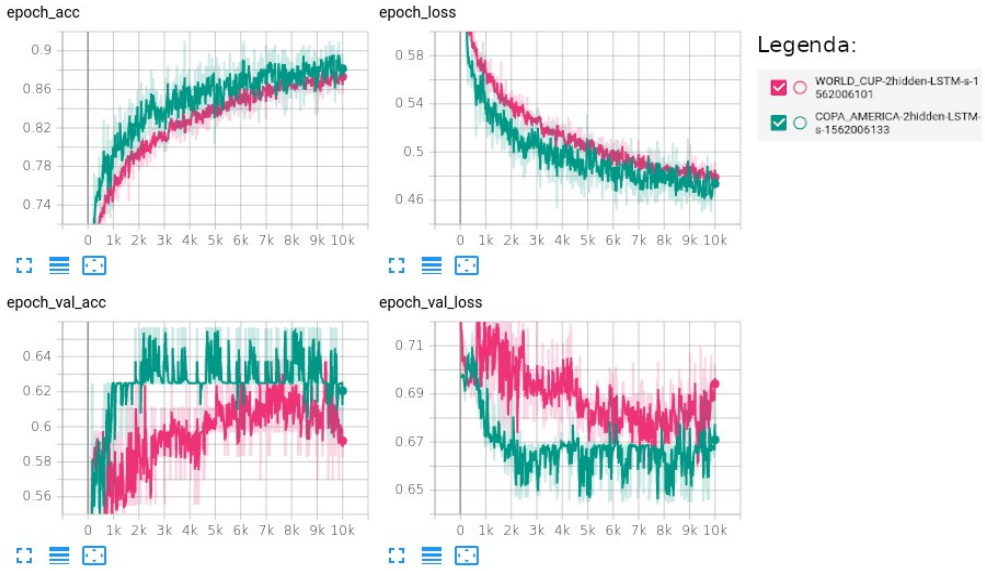
Tabela 4.6: Uspešnost napovedovanja zmagovalca tekme

Tekmovanje	**Uspešnost napovedi (v odstotkih)**	**Čas učenja (v sekundah)**	**Čas napovedovanja (v sekundah)**
Copa America	65,62	69	6
Svetovno prvenstvo	65,28	457	6

Na sliki 4.6 in tabeli 4.6 vidimo, da sta tekmovanji približno isto napovedljivi. Obe pa imata podoben trend, ki ga lahko vidimo v tabeli 4.7. To pomeni, da ima mreža večje probleme z napovedovanjem skupinskega dela turnirja, medtem ko izločilni del predvideva dobro, od 70 do 90 odstotkov natančno. Mogoče bi bilo bolje, da bi ta problem razdelili na učenje dveh nevronskih mrež, vsak del posebej.

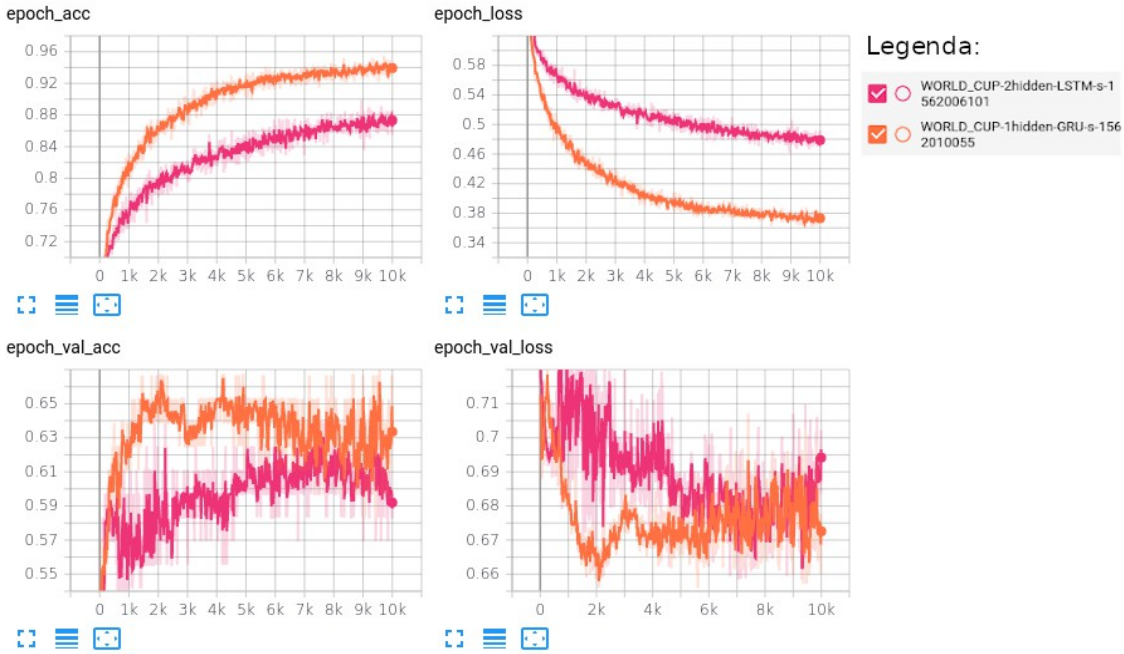
Tabela 4.7: Uspešnost napovedovanja tekmovanja skupinskega in izločevalnega dela

Tekmovanje	**Uspešnost napovedi skupinskega dela (v odstotkih)**	**Uspešnost napovedi izločevalnega dela (v odstotkih)**
Copa America	58,3	87,5
Svetovno prvenstvo	58,3	72,9



Slika 4.6: Grafi natančnosti in izgub za napovedovanje zmagovalca tekme v turnirskem tekmovanju

Ker imamo tukaj bistveno manj podatkov kot pri nogometnih ligah, smo poizkusili tudi nekaj različnih kombinacij nevronskih mrež, tako da smo spremenili število skritih plasti in pa celico LSTM v celico GRU. Za tekmovanje Copa America nismo uspeli doseči boljših rezultatov, medtem ko smo pri svetovnem prvenstvu prišli iz 65,28 na 69,44 odstotkov, kar je razvidno iz slike 4.7. Ta rezultat je bil dosežen pri celici GRU in eni skriti plasti namesto dveh.



Slika 4.7: Grafi natančnosti in izgub za napovedovanje zmagovalca svetovnega prvenstva na dveh različnih nevronskih mrežah

5 ZAKLJUČEK

Namen diplomske naloge je bilo ugotoviti, ali lahko z rekurentnimi nevronskimi mrežami uspešno napovemo zmagovalca nogometne tekme. Želeli smo zbrati dovolj dobre podatke in ustvariti uspešno nevronsko mrežo za dan problem.

Skozi delo smo ugotovili, da ligaški podatki o tekmah obstajajo za zadnja tri desetletja, o nogometnih turnirjih pa od samega začetka in da je beleženih veliko različnih stvari, čeprav jih je kar nekaj od tega neuporabnih in smo si morali na podlagi teh generirati nove podatke, ki nam dejansko pridejo prav. Veliko stvari smo morali tudi pridobiti iz spleta, tako da smo napisali skripte, ki so iz spletne strani razbrale prave podatke za našo uporabo. Preizkusili smo tudi veliko različnih sistemov nevronskih mrež, preden smo našli najboljšo verzijo.

Končni rezultati kažejo na napovedljivost zmagovalca ligaške tekme od 61 do 72 odstotkov. Za primerjavo je bila v delu [17] uspešnost napovedovanja zmagovalca največ 50 odstotkov. Dela se razlikujeta v tem, da smo želeli ugotoviti le, če domača ekipa zmaga in se tako rešiti najtežje napovedljivih neodločenih izidov in jih postaviti skupaj s porazom domače ekipe, medtem ko so v drugem diplomskem delu želeli pravilno napovedati tudi neodločene rezultate. Sistem dobro napove tudi izločevalni del nogometnih turnirjev, vse od 70 do 90 odstotkov, ima pa težave v skupinskem delu, in sicer je tam napovedljivost od 55 do 65 odstotkov. To bi mogoče lahko izboljšali s tem, da bi skupinski del gledali kot ločeni problem in bi ga napovedovali posebej. Nasplošno pa bi se vse verjetno dalo izboljšati z več vhodnimi podatki in podatki, ki segajo dlje v zgodovino. Poizkusili smo napovedati tudi druge priljubljene stvari na stavnih mestih, kot so na primer, ali pade na tekmo več kot 1,5 ali 2,5 gola in ali obe ekipi zadeneta gol. Pri napovedovanju, ali pade več kot 1,5 gola na tekmo smo imeli

uspešnost od 71 do 85 odstotkov, vendar je bila težava, da je bila takšna uspešnost, ker je mreža za vse napovedala, da pade več golov in se ni znala naučiti, kdaj temu ni tako. Pri napovedovanju, ali pade več kot 2,5 gola na tekmo pa smo dosegali že slabše rezultate, od 53 do 64 odstotkov. Približno enako slabe rezultate smo dobili pri napovedovanju, ali obe ekipi zadeneta, in sicer od 54 do 67 odstotkov.

Torej z rekurentnimi nevronskimi mrežami s celico LSTM in nogometno statistiko smo želeli podati napovedi za nadaljne tekme. Nadaljne delo in izboljšave bi lahko sledile iz pridobivanja večjega števila podatkov, tako za vsako tekmo, kot tudi za dlje časa nazaj v zgodovino. Morda bi nam pri tem lahko pomagal tudi spremenjen model nevronske mreže.

6 VIRI IN LITERATURA

[1] A Beginner's Guide to Neural Networks and Deep Learning. Dostopno na: https://skymind.ai/wiki/neural-network [17. 6. 2019]

[2] Artificial neural network. Dostopno na: https://en.wikipedia.org/wiki/Artificial_neural_network [17. 6. 2019]

[3] Guid N., Strnad D. *Umetna inteligenca, 1. izdaja*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru, 2015

[4] A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way. Dostopno na: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 [30. 6. 2019]

[5] Top 2018 Machine Learning Trends (And Our 2019 Preview!). Dostopno na: https://medium.com/the-official-integrate-ai-blog/top-2018-machine-learning-trends-and-our-2019-preview-9a6c82e6afba [2. 7. 2019]

[6] Aggarwal Charu C. *Neural networks and deep learning : a textbook*. Cham: Springer, 2018

[7] Recurrent neural networks and lstm. Dostopno na: https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5 [30. 5. 2019]

[8] Illustrated Guide to LSTM’s and GRU’s: A step by step explanation. Dostopno na: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 [17. 6. 2019]

[9] Python. Dostopno na: https://www.python.org/ [2. 7. 2019]

[10] Keras. Dostopno na: https://keras.io/ [29. 6. 2019]

[11] Football Betting | Football Results | Football Bets | Football Odds. Dostopno na: http://www.football-data.co.uk/ [19. 6. 2019]

[12] Slovenian PrvaLiga. Dostopno na: https://en.wikipedia.org/wiki/Slovenian_PrvaLiga [19. 6. 2019]

[13] PLTS - Prva Liga Telekoma Slovenije. Dostopno na: https://www.prvaliga.si/prvaliga/default.asp [19. 6. 2019]

[14] PrvaLigaWebScraper. Dostopno na: https://github.com/planeer/PrvaLigaWebScraper [19. 6. 2019]

[15] FIFA World Cup. Dostopno na: https://www.kaggle.com/abecklas/fifa-world-cup [19. 6. 2019]

[16] CopaAmericaWebScraper. Dostopno na: https://github.com/planeer/CopaAmericaWebScraper [19. 6. 2019]

[17] Korpar, Ž. *Predikcija športnih rezultatov z uporabo strojnega učenja*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko Univerze v Mariboru, 2018.