Predmet: "Vještačka inteligencija"
Laboratorijska vježba 4: Neuronske mreže (2/4)
Odgovorna nastavnica: Vanr. prof. dr Amila Akagić

Sadržaj vježbe:

1 Cilj vježbe

Cilj vježbe je upoznavanje sa naprednijim konceptima tekstualne klasifikacije. Kroz vježbu, studenti se upoznaju sa pojmovima tokenizacije i vektorizacije. U zadacima studenti koriste naučene koncepte kako bi rješavali problem klasifikacije tekstualnih podataka, pri čemu podaci nisu tabelarno struktuirani.

2 Neuronske mreže za problem klasifikacije (nastavak)

U prethodnoj vježbi su pokazane neke osnovne tehnike rješavanja problema klasifikacije, međutim za nešto složenije probleme ove tehnike nisu dovoljne. Stoga, u nastavku će biti pokazane dvije tehnike koje se vrlo često koriste. Prva od njih je t**okenizacija i vektorizacija teksta** koja se koristi u sklopu predprocesiranja tekstualnih podataka koji nisu tako striktno struktuirani kao što su bili u prošloj vježbi. Druga tehnika se koristi tokom iterativnog procesa treniranja i prilagođavanja hiperparametara, a naziva se **validacija**. Ona je ukratko spomenuta u prošloj vježbi, a u ovoj je nešto detaljnije prikazana.

2.1 Tokenizacija i vektorizacija teksta

Pri radu sa neuronskim mrežama sa tekstualnim podacima, potrebno je izvršiti neki (en)kodiranja teksta u brojčane vrijednosti s obzirom da to neuroni u mreži očekuju. Ukoliko su podaci u vidu rečenica, odnosno u prirodnom jeziku, tada obrada podataka spada u širu oblast koja se naziva procesiranje prirodnog jezika (eng. *Natural Language Processing - NLP*). Dodjela brojčanih vrijednosti ovakvim podacima se radi kroz dva sekvencijalna koraka koji se nazivaju tokenizacija i vektorizacija. **Tokenizacija** predstavlja dijeljenje teksta u riječi i na ovaj način se formira indeks/rječnik koji predstavlja vokabular skupa podataka i daje informaciju o relativnoj frekvenciji pojavljivanja svake od riječi u tekstu koji se tokenizira. Na primjer, ukoliko se tokenizira tekst koji se sastoji od sljedećih rečenica: *'Pop kopa bob. Ko pokopa popu bob? Pop pokopa popu bob.'*. Rezultat ovoga bi bio sljedeći rječnik:

```
{'bob': 1, 'pop': 2, 'pokopa': 3, 'popu': 4, 'kopa': 5, 'ko': 6}
```

Riječ *bob* se ponavlja najviše puta te joj je dodijeljen broj 1, sljedeći najveći broj ponavljanja ima riječ *pop* te je njoj dodijeljen broj 2, i tako dalje. Ovaj korak se može izvršiti, koristeći Keras, sljedećim kodom:

```
from keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
train_data_seq = tokenizer.texts_to_sequences(X_train)
test_data_seq = tokenizer.texts_to_sequences(X_test)
```

Prva linija kreira objekat `Tokenizer`, a druga vrši kreiranje rječnika na osnovu teksta. Naredne dvije linije vrše pretvaranje teksta u nizove cijelih brojeva na način da se svaka riječ zamijeni odgovarajućim brojem iz prethodno generisanog rječnika. Ono što se često uzima kao podrazumijevano pri tokenizaciji je to da se znakovi interpunkcije ignorišu i to da tokenizacija nije osjetljiva na velika i mala slova. Ukoliko se želi vidjeti generisani rječnik ili vidjeti broj ponavljanja svake od riječi, to se može učiniti sljedećim pozivima:

```
print(tokenizer.word_index) # Ispisuje generisani rjecnik
print(tokenizer.word_counts) # Ispisuje broj ponavljanja svake od rijeci u tekstu
```

Drugi pomenuti korak je vektorizacija. U ovom koraku je cilj pronaći neki reprezentativniji numerički oblik teksta. Postoji više načina za ovo. Jedan od često korištenih načina je tzv. *one-hot encoding* gdje se svaki tekst predstvlja vektorom čiji elementi naznačavaju prisustvo ili odsustvo tokena (odnosno riječi) u tekstu. Ukoliko se u tekstu nalazi riječ kojoj je pridružena vrijednost jedan u rječniku iz koraka tokenizacije, onda će prvi element vektora kojim predstavljamo ovaj tekst imati vrijednost jedan, a u suprotnom će imati vrijednost nula. Za ovu svrhu, sljedećim kodom, možemo definisati funkciju koja vrši vektorizaciju podataka:

```
1 def vectorize_sequences(sequences, dimension=4000):
2     results = zeros((len(sequences), dimension))
3     for i, sequence in enumerate(sequences):
4         results[i, sequence] = 1.
5     return results
```

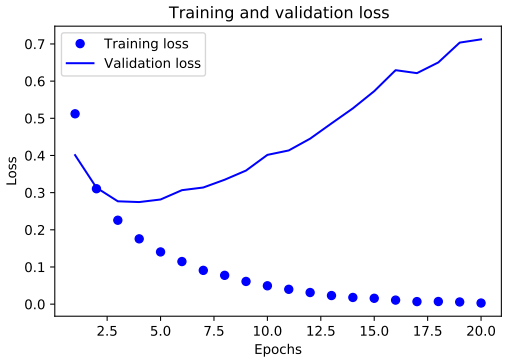
Ova funkcija prvo inicijalizira vektor `results` na vektor nula u liniji 2, a zatim pobrojava tokenizirane sekvence i svaku takvu sekvencu indeksira po njenim elementima i postavlja odgovarajuće elemente vektora na vrijednost jedan. Slična ovoj funkciji je i funkcija za vektorizaciju labela. Za slučaj binarne klasifikacije ona bi mogla izgledati ovako:

```
1 def vectorize_labels(labels):
2     results = zeros(len(labels))
3     for i, label in enumerate(labels):
4         if (label.lower() == 'abnormal'):
5             results[i] = 1
6     return results
```

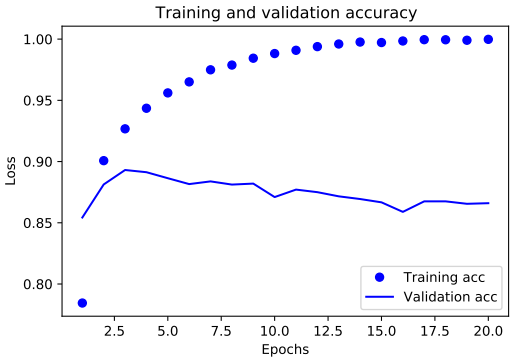
Pored navedenih tehnika predprocesiranja koje su spomenute u prethodnoj i ovoj vježbi, ukoliko se tokom analize skupa podataka uoče neke karakteristike koje se ponašaju kao šum, poželjno ih je ukloniti i prije nego se krene sa skaliranjem ili tokenizacijom i vektorizacijom. Primjer toga je ukoliko se podaci nalaze u html formatu, može se pokazati korisnim uklanjanje HTML tagova iz razloga što oni, za većinu primjena, ne nose nikakve korisne informacije.

2.2 Validacija

U prethodnoj vježbi je validacija spomenuta kao neobavezan korak u procesu nadziranog učenja, međutim ona se koristi gotovo u svim ozbiljnijim projektima. Služi kako bi se stekla indikacija o tome kako bi se model ponašao nad testnim podacima, a kasnije i u realnoj primjeni. U svrhu validacije se u općem slučaju izdvaja određeni podskup skupa podataka za treniranje i on se naziva validacijskim skupom. Taj podskup se ne podvrgava treniranju, već se koristi samo kako bi se stekao uvid o tome kako će se model ponašati nad podacima koje do sada nije vidio. Ovim se može prepoznati situacija koja se naziva *overfitting*, odnosno da je model previše prilagođen skupu za treniranje te da radi znatno lošije za podatke koji nisu dovoljno bliski skupu za treniranje. Kao primjer *overfitting*-a može poslužiti sljedeća slika sa prethodne vježbe:



(a) Prikaz vrijednosti funkcije gubitka.



(b) Prikaz tačnosti modela.

Slika 1: Prikaz rezultata nakon treniranja. Ovo je klasičan primjer *overfitting*-a jer se vrijednosti metrika i funkcije gubitka značajno počinju razlikovati nad validacijskom i trening skupu.

Na slici 1a se vidi da nakon četvrte epohe vrijednost funkcije gubitka nad validacijskim skupom počinje značajno rasti, dok nad skupom za treniranje i dalje nastavlja opadati. Slična situacija je i na slici 1b gdje tačnost modela nad validacijskim skupom podataka počinje degradirati nakon četvrte epohe, dok nad trening skupom podataka nastavlja se poboljšavati. Ovo je klasičan primjer pomenutog *overfitting*-a. Da nije bila vršena validacija, moglo bi se doći do pogrešnog zaključka da model ima odlične performanse i da je spreman za korištenje u praksi. Nakon što se sprovede ova analiza vrijednosti tačnosti i funkcije gubitka nad validacijskim skupom, potrebno je suočiti se sa ovim problemom. Jedan jednostavan način je da se prekine proces treniranja nakon četvrte epohe što bi dalo neko lokalno optimalno rješenje. Pored ovoga postoje neke naprednije tehnike rješavanja problema *overfitting*-a, no to je van opsega ove vježbe.

Jedan od načina vršenja validacije u Keras-u je dat primjerom koda:

```
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split
    =0.2)
```

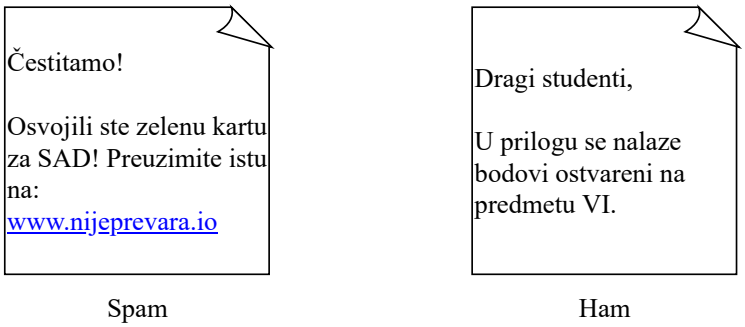
Prva 4 parametra su od ranije poznata, a posljednji proslijeđeni parametar `validation_split` predstavlja procenat koji se izdvaja iz trening skupa da se koristi za validaciju. Na ovaj način će se vršiti validacija nakon svake epohe. Ovo je samo jedan od načina vršenja validacije, a postoje i naprednije i obuhvatnije tehnike validiranja modela poput kros-validacije i njenih modifikacija.

3 Zadaci za rad u laboratoriji

Predviđeno je da se svi zadaci u nastavku rade u sklopu Jupyter Notebook okruženja. Svaki podzadatak treba biti zasebna Jupyter ćelija.

Zadatak 1 - Binarna klasifikacija - Spam filter

U ovom zadatku nastavljamo sa binarnom klasifikacijom, no ovdje ćemo se upoznati sa radom na skupu podataka koji zahtijeva procesiranje prirodnog jezika (eng. *Natural Language Processing - NLP*). Ovo podrazumijeva korištenje iznad objašnjene tehnike tokenizacije i vektorizacije podataka. Problem nad kojim će se ovo primjenjivati je klasificiranje poruka kao *spam* ili *ham*. *Spam* predstavljaju neželjene poruke koje su u nekim slučajevima i generisane te se šalju velikom broju korisnika bilo radi reklame, pokušaja prevare, namjernog opterećivanja inboxa korisnika i tako dalje. Sa druge strane, *ham* poruke su korisne poruke koje korisnik zaista treba i želi da primi. Primjer ovakvih poruka je prikazan na slici 2. Mail servisi koriste upravo *spam* klasifikatore kako bi zaštitili svoje korisnike od prevara, virusa ili bilo kakvog neželjenog maila. Skup podataka koji ćemo koristiti se sastoji od 2100 poruka koje imaju pridruženu labelu *Spam* ili *Ham*.



Slika 2: Razlika između *spam* (beskorisne/štetne) i *ham* (korisne) poruke.

a) Učitati skup podataka iz priloga vježbe *'SpamDetectionData.txt'* te prikazati prva 3 podatka kako bi se upoznali sa formatom skupa podataka. Koje su kolone u ovom skupu podataka?

b) Iz skupa podataka izdvojiti X i y pri čemu je X skup poruka, a y pridružene labele. Zatim ukloniti iz poruka html tagove `<p>` i `</p>` s obzirom da se oni nalaze u svakoj poruci. Koliko slova ima prva, a koliko druga rečenica iz skupa podataka?

c) Podijeliti skup podataka na dio za treniranje i testiranje pri čemu 10% ukupnog skupa se treba uzeti kao testni set;

d) Izvršiti tokenizaciju teksta korištenjem `Tokenizer` objekta kao što je opisano u vježbi. Vokabular generisati na osnovu trening podataka. Nakon toga, na osnovu generisanog rječnika pretvoriti sve poruke (i iz trening i test skupa) iz teksta u niz cijelih brojeva. Koje su tri najčešće riječi u tekstu? Kako izgleda prva rečenica iz trening skupa podataka, a kako izgleda formirani niz cijelih brojeva za nju?

e) Kao što smo se mogli uvjeriti u zadatku *b)*, nemaju sve rečenice istu dužinu. To se može riješiti vektorizacijom. Definisati funkcije `vectorize_sequences(sequences, dimension)` i `vectorize_labels(labels)`. Prva funkcija treba da vrši vektorizaciju ulaznih podataka i prima kao prvi parametar nizove cijelih brojeva koji su rezultat prethodnog podzadatka. Kao drugi parametar treba da prima broj na koju dužinu treba vektorizovati te nizove. Druga funkcija, `vectorize_labels`, treba da vrši vektorizaciju labela pri čemu labeli '*spam*' dodijeliti vrijednost 1, a klasi '*ham*' vrijednost 0. Pozvati ove funkcije nad vrijednostima dobijenim pod *c)* pri čemu vektorizaciju ulaznih podataka vršiti na vektore od 4000 elemenata;

f) Definisati sekvencijalni Keras model koji prima ulaz oblika `(4000,)`. Prva dva skrivena sloja trebaju biti `Dense` i imati 8 neurona sa aktivacijskom funkcijom `relu`. Izlazni sloj treba imati jedan neuron i imati `sigmoid` aktivacijsku funkciju;

g) Kompajlirati model tako da koristi `rmsprop` optimizator, za funkciju gubitka koristiti `binary_crossentropy`, te `accuracy` kao metriku;

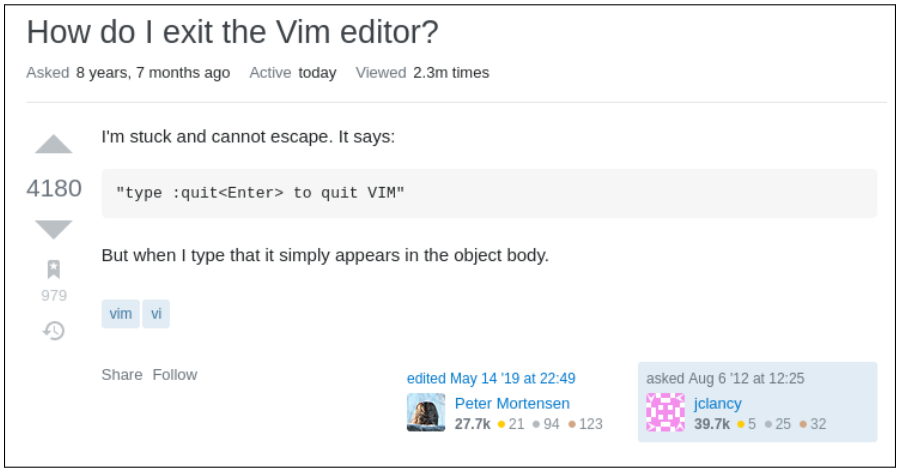
h) Istrenirati model na 5 epoha sa veličinom *batcha* od 128. 30% skupa za treniranje koristiti za validaciju. Kolika je postignuta tačnost i vrijednost funkcije gubitka? Grafički prikazati;

i) Izvršiti evaluaciju modela nad testnim skupom podataka. Kolika je tačnost nad ovim skupom?

j) Definišite proizvoljnu poruku, pomoću tokenizera formirajte niz cijelih brojeva, vektorizujte ga i provjerite da li model ispravno klasificira tu poruku.

Zadatak 2 - Višeklasna klasifikacija - klasificiranje Stackoverflow pitanja

U ovom zadatku ćemo se upoznati sa višeklasnom klasifikacijom. Problem koji će se rješavati je klasifikacija pitanja sa Stack Overflow platforme (`https://stackoverflow.com/`). Skup podataka se sastoji od pitanja iz nekoliko programskih jezika. I ovdje se radi o NLP te će se koristiti tokenizacija. Cilj klasifikacije jeste na osnovu pitanja sa Stack Overflow odrediti na koji se programski jezik pitanje odnosi. Primjer tipičnog Stack Overflow pitanja je dat na slici 3.



Slika 3: Primjer Stack Overflow pitanja.

a) Učitati *'stackoverflow.csv'* skup podataka iz priloga vježbe i prikazati posljednja tri podatka;

b) Izdvojiti iz skupa podataka X i y, odnosno skup pitanja i skup odgovarajućih labela respektivno. Koliko ima jedinstvenih labela, odnosno iz koliko programskih jezika se nalaze pitanja u skupu podataka? Koji su to programski jezici?

c) Izvršiti *one-hot* enkodiranje labela - prvo tekstualne labele mapirati u cijeli broj pomoću `LabelEncoder` objekta iz `sklearn.preprocessing` modula. Izvršiti *one-hot* enkodiranje labela korištenjem `to_categorical` funkcije iz `keras.utils modula`. Ispisati dobijeni niz labela;

d) Podijeliti skup podataka na dio za treniranje i testiranje pri čemu 10% ukupnog skupa se treba uzeti kao testni set;

e) Tokenizirati i vektorizovati tekst (pitanja) slično kao u prethodnom zadatku. Prilikom tokenizacije uzimati u obzir samo 500 najčešćih riječi (ovo se može definisati pri samom formiranju `Tokenizer` objekta pomoću jednog od parametara). Prema ovome prilagoditi i parametar `dimensions` pri vektorizaciji;

f) Definisati sekvencijalni Keras model sa 3 `Dense` sloja. Prvi treba imati 32 neurona, drugi 8 neurona, a posljednji, koji je i izlazni treba imati onoliko neurona koliko ima klasa u ovom problemu. Aktivacijske funkcije prva dva sloja postaviti na `relu`, a posljednjeg sloja na `softmax`;

g) Kompajlirati model tako da se koristi `adam` optimizator, `categorical_crossentropy` funkcija gubitka i `accuracy` metrika. Prikazati sažetak (eng. *summary*) modela;

h) Istrenirati model na 8 epoha sa veličinom batcha 8. Izdvojiti 25% trening skupa da se koristi za validaciju. Kolika je postignuta tačnost modela I kolika je vrijednost funkcije gubitka? Grafički prikazati;

i) Izvršiti evaluaciju modela nad testnim skupom. Kolika je tačnost nad ovim skupom?