Peta domača naloga

Tadej Petrič

June 20, 2023

Na vseh grafih je x os število dni (od začetka predvidovanja). Kadar sta na grafu dve krivulji je y os vrednost napovedi (oranžna napoved, modra točen rezultat). Kadar je na grafu samo ena krivulja je y os L_1 napaka v določenem dnevu. Prvih N korakov je vedno točnih, kasnejše napoveduje algoritem.

1 Prva

Za napovedovanje sem poskusil različne algoritme: rekurenčne nevronske mreže, LSTM in navadne globoke nevronske mreže. Pri rekurenčnih in LSTM mrežah je število slojev enako parametru N (zgodovini primerov), za izhodni parameter pa sem vedno izbral 1. Pri globoki nevronski mreži pa je velikost vhodne plasti enaka N, izhodna plast pa prav tako vedno 1, ima pa še dva vmesna skrita sloja. Da sem iz tega razvil model, ki napoveduje za M=7 ali M=30 sem rezultat napovedi uporabil kot vhod zadnjega sloja nevronske mreže (preostanek vhodov pa zamaknil). Alternativni pristop bi bil, da bi takoj zadnji sloj napovedoval več členov (7 ali 30), vendar se je z mojimi meritvami bolj odnesel trenutni pristop. Zaradi hitrosti sem uporabil batch size 16.

Arhitektura modelov

Modela za RNN in LSTM imata samo decoder in RNN/LSTM sloj. Uporabil nisem dropouta, nelinearnost je tanh. Skriti je na začetku ničelni, izhod RNN/LSTM sloja pošljemo v dekodirnik (linearna plast) in vrnemo.

Model za globoko linearno sprejme N=10 vhodov in 20 izhodov, pošlje skozi relu in dropout $p = 0.15$. Obe skriti plasti sprejmeta 20 vhodov in vrneta 20 izhodov, prav tako pa imata relu in dropout $p = 0.15$. Zadnja plast sprejme 20 vhodov in vrne en izhod.

Postopki učenja in testiranja

Nevronsko mrežo sem učil na vsakem mestu (ki ni mestna občina) posebaj ter napovedoval naslednji dan za to mesto. Podatke sem pripravil tako, da sem najprej normaliziral vse stolpce. Nato sem za vsako mesto poiskal vsa podzaporedja dolžine N ter ustrezni člen, ki ga želimo napovedati. Nato sem naključno

model	napaka
RNN N=5, M=7	0.0294
RNN N=5, M=30	0.0942
LSTM N=5, M=7	0.0299
LSTM N=5, M=30	0.0712
DNN N=10, M=7	0.0100
DNN N=10, M=30	0.0691
RNN N=10, M=7	0.0035
RNN N=10, M=30	0.0470
LSTM N=10, M=7	0.0059
LSTM N=10, M=30	0.0700

preuredil podatke. Učil sem z majhnim learning rateom (okoli 0.002) in weight decay (0.005) na optimizatorju AdamW.

Testiral sem na stolpcih mestnih občin. Tam sem podobno poiskal vsa podzaporedja in ustrezen naslednji člen zaporedja ter primerjal odstopanje ($L_1(x, y) = |x - y|$) od napovedi.

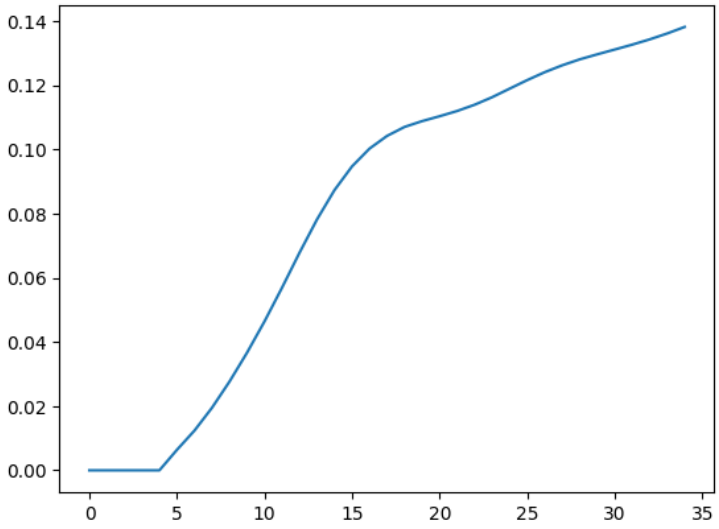

Figure 1: Napaka RNN z N=5

Vidimo, da če napovedujemo za dlje v prihodnost tudi napaka postaja večja. Napaka nekaterih modelov narašča hitreje, kot napaka drugih modelov (npr. napaka pri DNN narašča zelo hitro na začetku, a se ustali. Napaka za RNN pa narašča počasneje, ampak je slabša na dolgi rok). Modeli dobijo kar nizko napako, bi se jo pa dalo verjetno še izboljšati z bolj natančno izbiro hiperparametrov.

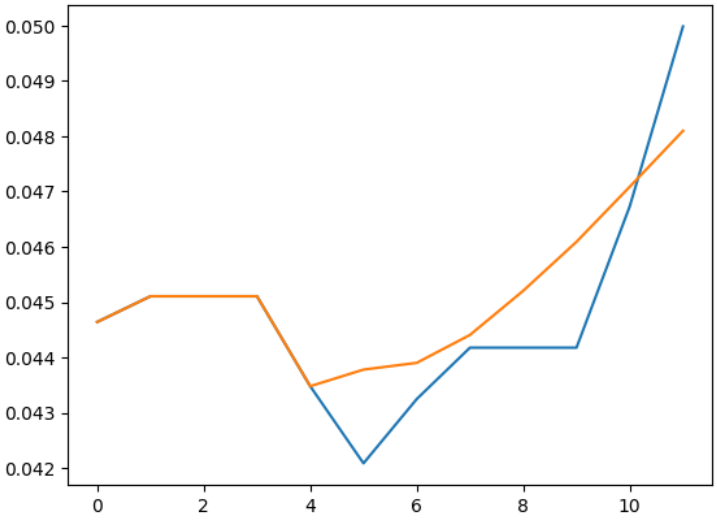

Figure 2: Najboljše prileganje RNN z N=5

2 Druga

Prvega načina sem se lotil tako, da sem uporabil trenutno občino (za katero napovedujemo) ter njene sosede. Ker ne moremo uporabiti vseh sosedov (saj je podatek nestrukturiran), izberemo le nekaj naključnih sosedov (recimo 3 sosede). Nato v nevronsko mrežo poleg zgodovine okužb trenutnega mesta podamo še zgodovino okužb izbranih sosednjih občin. Drugega načina, kjer uporabimo vse podatke in ne le striktnih sosedov, sem se lotil na podoben način kot node2vec: poleg samo sosedov občine, za katero napovedujemo, lahko uporabimo še sosede sosedov. Torej, v prvem koraku izberemo naključnega soseda občine (recimo mu X). V drugem koraku izberemo naključnega soseda ali občine za katero napovedujemo, ali od soseda X. V naslednjem koraku lahko izbiramo še od sosedov izbrane občine iz prejšnega koraka (poleg sosedov originalne občine in sosedov X), in tako naprej. Za dovolj velike hiperparametre lahko tako preiščemo tudi cel graf, kljub temu pa imamo konstantno število stolpcev v podatkovju in upoštevamo strukturo grafa. Sosede, katere lahko izbiram, dam v seznam (in ne množico) in izbiram z zamenjavo, saj tako dobim boljšo utežitev bližnjih elementov ter preprečim nekaj tehničnih težav (recimo če uporabljamo 5 sosedov, vendar ima mesto samo 3 sosede).

Saj zaradi dodatnih dimenzij (če imamo k sosedov, v mrežo vnesemo k-krat več podatkov) algoritem postane počasen, uporabljam le navadne globoke nevronske mreže. Če napovedujem glede na zgodovino 10 dni in k sosedov ima torej nova nevronska mreža $10(k+1)$ vhodov v vhodni plasti. Ko sem zgeneriral vsa zaporedja vhodov sem jih tudi naključno permutiral. Pri drugem pristopu sem tudi vsak epoch izbral druge sosede zato, da dobimo boljšo predstavitev

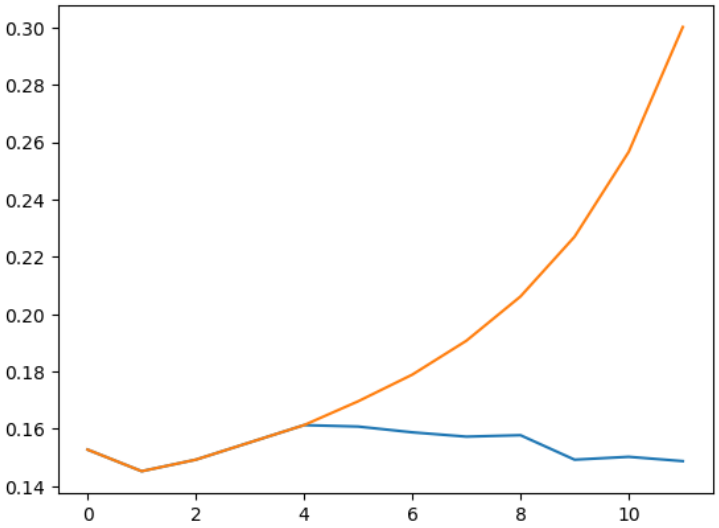

Figure 3: najslabše prileganje RNN z N=5

model	napaka
GNN M=7	0.0312
GNN M=30	0.0549
GNN2 M=7	0.0334
GNN2 M=30	0.0465

grafa. Pri učenju sem izpustil primere, kjer napovedujem mestne občine.

Spet sem napovedoval le za en dan v prihodnost in to uporabil za generiranje modela z M=7 in M=30. Tukaj sem podatke ostalih občin kopiral od pravih podatkov, z svojim izhodom sem spreminjal le podatke trenutne občine.

Testiral sem na vseh mestnih občinah.

Opazimo, da je način, kjer upoštevamo sosednosti boljši od prvotnega načina (kjer upoštevamo samo trenutni položaj). Drugi način poda še dodatno izboljšavo (za M=30); delno je verjetno zato, ker se podatki za občino, katero napovedujemo, lahko podvojijo (in tako dodatno utežijo pomembnost). Z tem pristopom dobim tudi najboljše rezultate od vseh ostalih pristopov. Slaba stran pa je, da tudi izvajanje traja najdlje (bi pa lahko pohitril, če bi izvajal v batchih).

Tako opazimo, da je sosednost občin koristna za napovedovanje. To je verjetno zato, ker se okužbe širijo iz ene občine v drugo. Če naraščanje okužb prepoznamo v sosednji občini, bo verjetno sledilo tudi naraščanje okužb v trenutni občini.

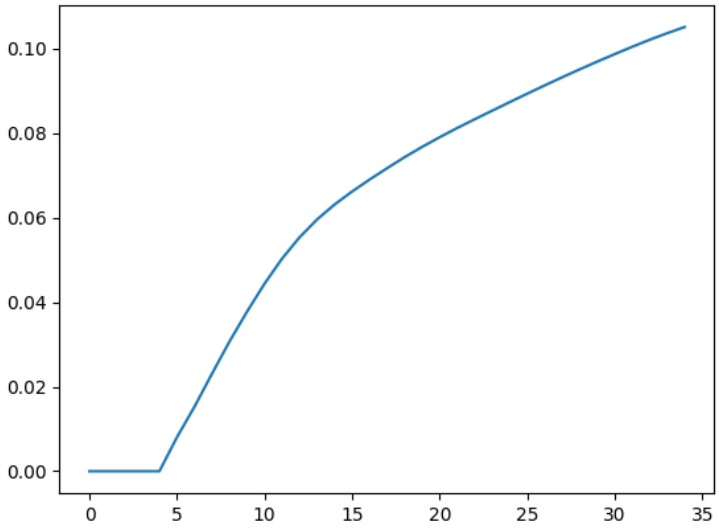

Figure 4: Napaka lstm z N=5

3 Opis datotek

V datoteki `naloga.ipynb` preberem podatke, vrednosti NaN zamenjam z 0 ter stolpce normiram. Nato ločim testne podatke od učnih. Potem za vsak algoritem definiram hiperparametre in ga naučim, potem pa izrišem grafe ter izpišem napake. V drugem delu naloge spet preberem podatke in jih očistim in normiram, prav tako pa preberem še datoteko, ki opisuje graf sosednosti. To preberem v tenzor, ki opiše kateri indeks je soseden kateremu indeksu (kjer indeks i predstavlja $i+1$ stolpec), nato pa naredim graf neusmerjen. Poiščem še indekse testne množice, potem pa učim modele in izpisujem rezultate podobno kot v prvem delu.

V datoteki `models.py` definiram modele, ki jih uporabim.

V datoteki `utility.py` definiram razne funkcije, ki mi pomagajo pri generiranju podatkov in učenju.

V datoteki `utility_geom.py` definiram funkcije za branje grafa, za iskanje sosedov in za generiranje podatkov, ki upoštevajo strukturo grafa. Funkcije za pripravo podatkov, ki imajo v imenu `montedfs` so različice funkcij, ki upoštevajo strukturo celotnega grafa, tiste brez pa upoštevajo samo sosede. Obe vrste so samo analog navadnih funkcij iz `utility.py`

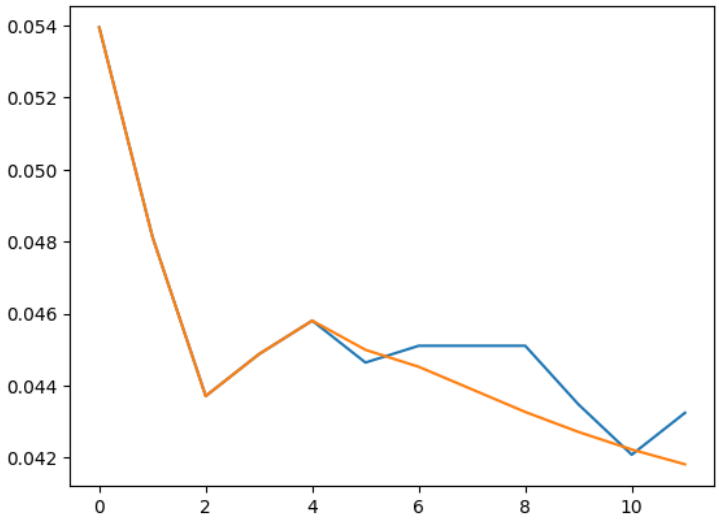

Figure 5: Najboljše prileganje lstm z N=5

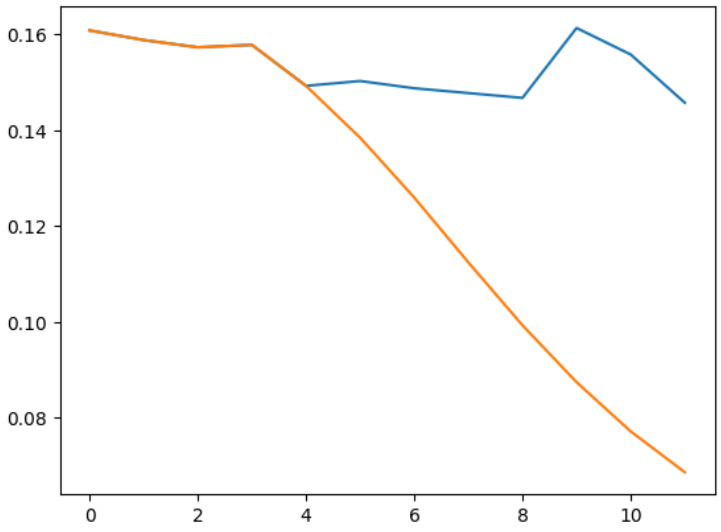

Figure 6: najslabše prileganje lstm z N=5

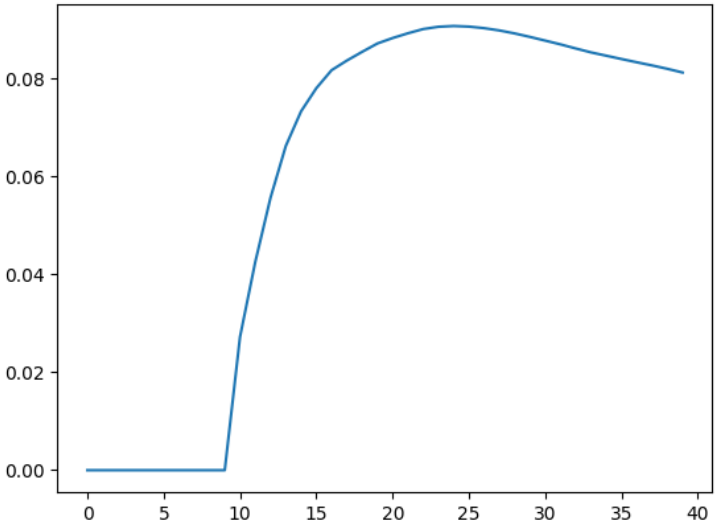

Figure 7: Napaka dnn z N=10

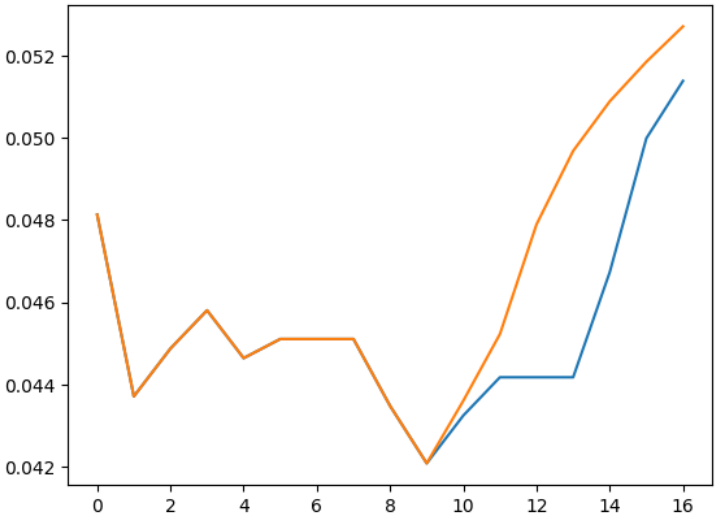

Figure 8: Najboljše prileganje dnn z N=10

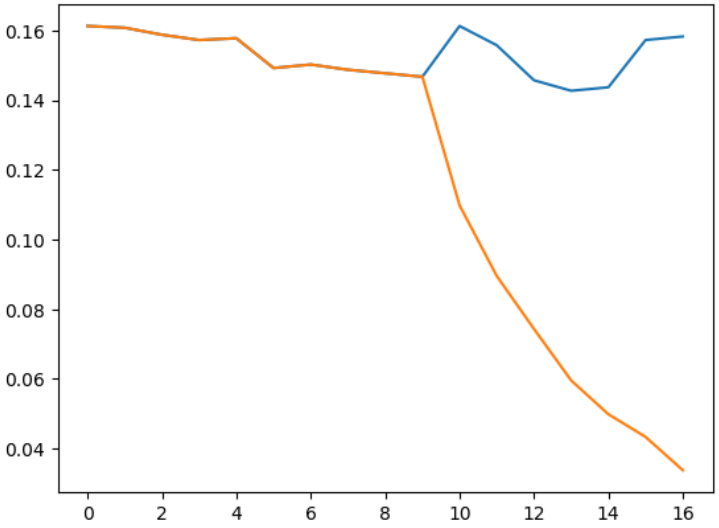

Figure 9: najslabše prileganje dnn z N=10

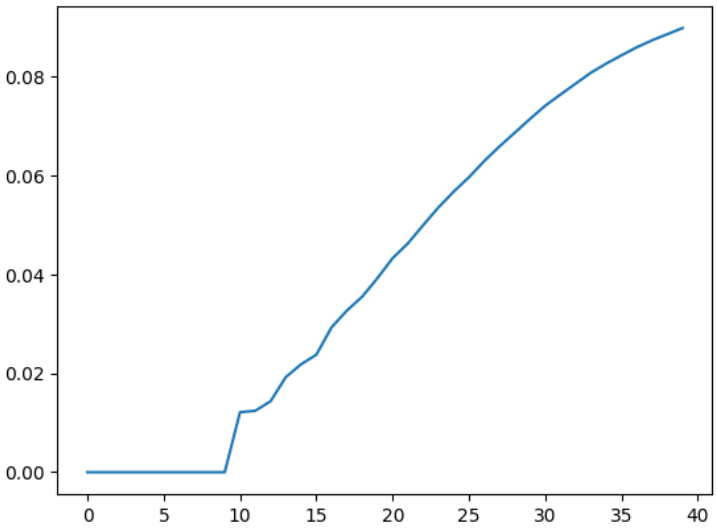

Figure 10: Napaka RNN z N=10

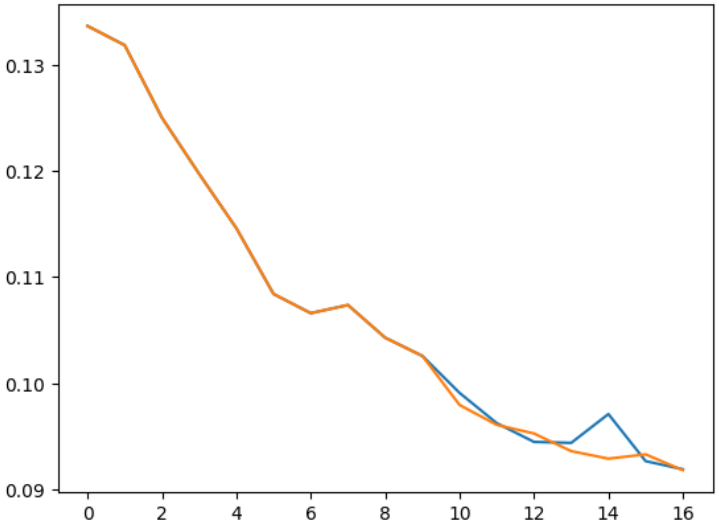

Figure 11: Najboljše prileganje RNN z N=10

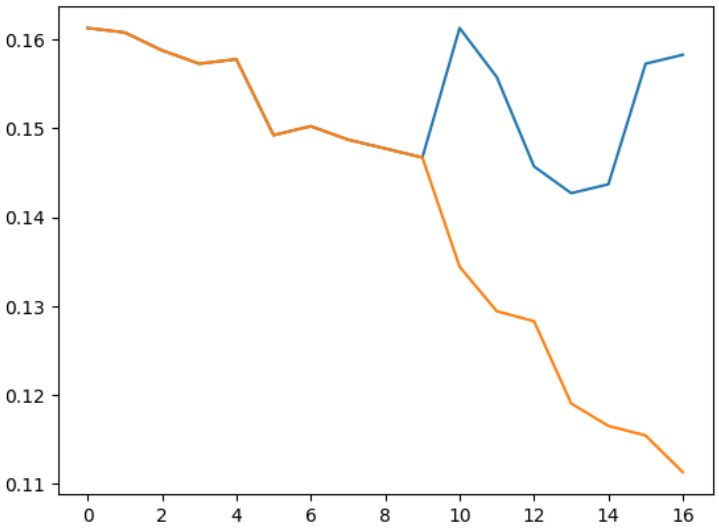

Figure 12: najslabše prileganje RNN z N=10

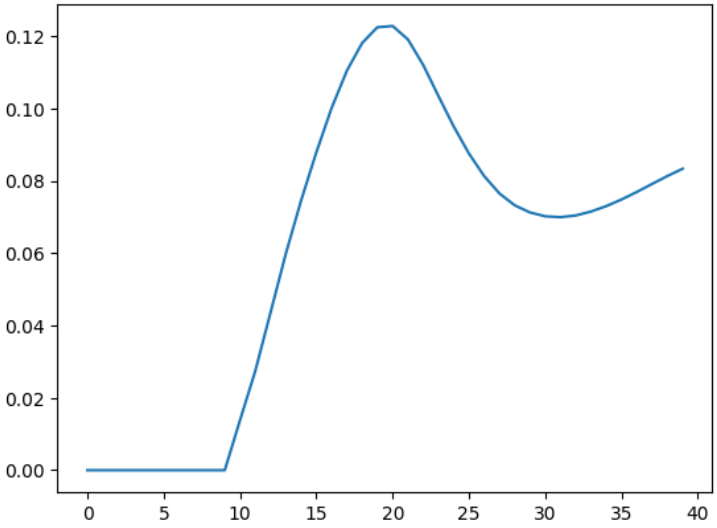

Figure 13: Napaka lstm z N=10

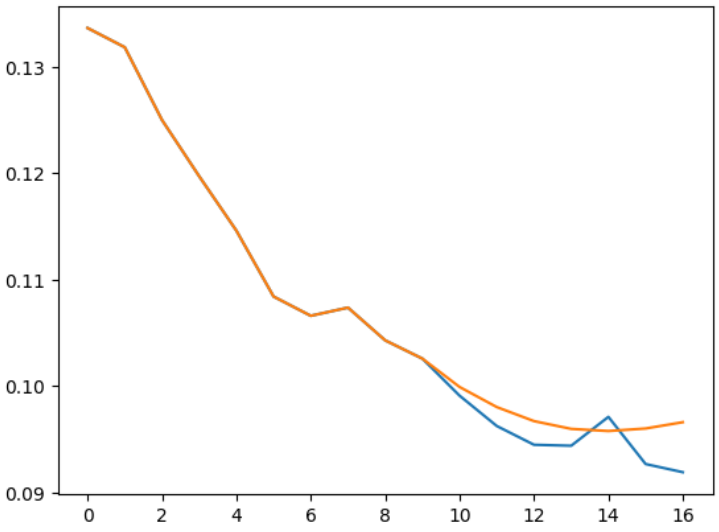

Figure 14: Najboljše prileganje lstm z N=10

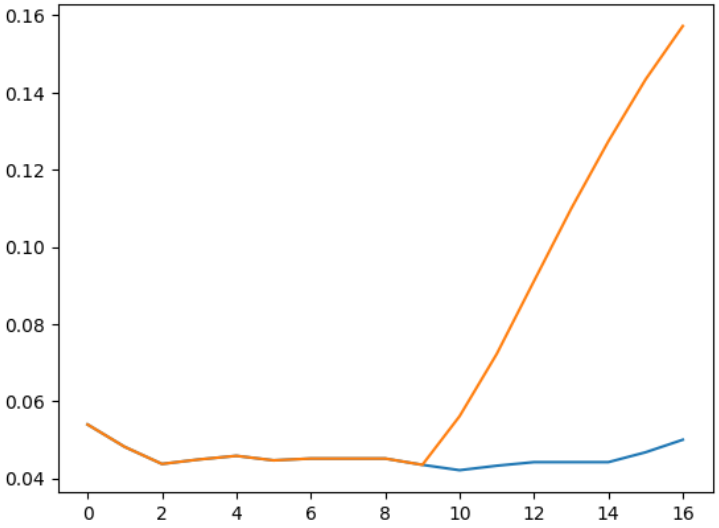

Figure 15: najslabše prileganje lstm z N=10

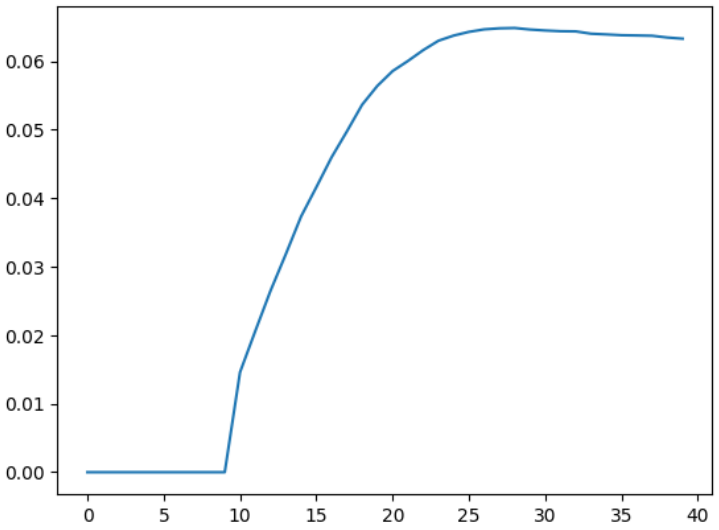

Figure 16: napake za prvi način z grafi

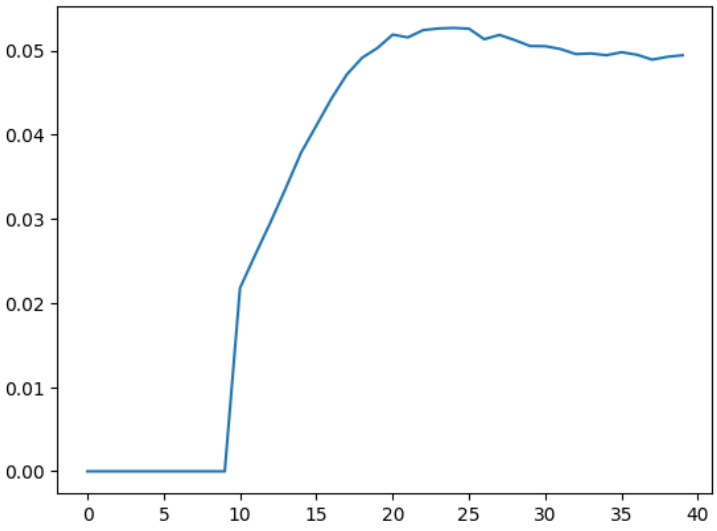

Figure 17: napake za drugi način z grafi