Zahvaljujem se mentoru prof. Jakobovića na prijateljskom pristupu, konstruktivnim idejama, vodstvu, motivaciji, i potpori kroz izradu ovog završnog rada. Također se zahvaljujem svojim roditeljima zbog njihove neprekidne potpore tijekom moga obrazovanja.

Sadržaj

Uvod

U današnje vrijeme dosta se priča o razvoju umjetne inteligencije, no što je umjetna inteligencija točno? Sljedeći citat nam može pobliže objasniti šta bi trebala biti umjetna inteligencija.

„Umjetna inteligencija – naziv za znanstvenu disciplinu koja se bavi izgradnjom računalnih sustava čije se ponašanje može tumačiti kao inteligentno“ - John McCarthy, (1956.).

Strojno učenje možemo nazvati primjenom umjetne inteligencije. Cilj nekog modela strojnog učenja je da uči i poboljšava se iz prethodno stečenih znanja i iskustava. Kao što smo strojno učenje nazvali primjenom umjetne inteligencije, tako možemo duboko učenje nazvati primjenom strojnog učenja. Duboko učenje pokušava imitirati način na koji ljudska inteligencija funkcionira da bi stekao određena znanja. Napretkom metoda strojnog učenja, u slučaju ovog završnog rada metoda dubokog učenja, pronalazimo njihove razne primjene u životu, primjerice kod autonomnih automobila, prepoznavanja objekata, prepoznavanja izraza lica, traženje uzorka korisnikovog ponašanja u svrhu plasiranja ciljanih reklami, i tako dalje. Tako će svrha i cilj ovog rada biti traženje uzorka kretanja i pokušaj predviđanja tržišta dionica koristeći model dubokog učenja, preciznije povratne neuronske mreže.

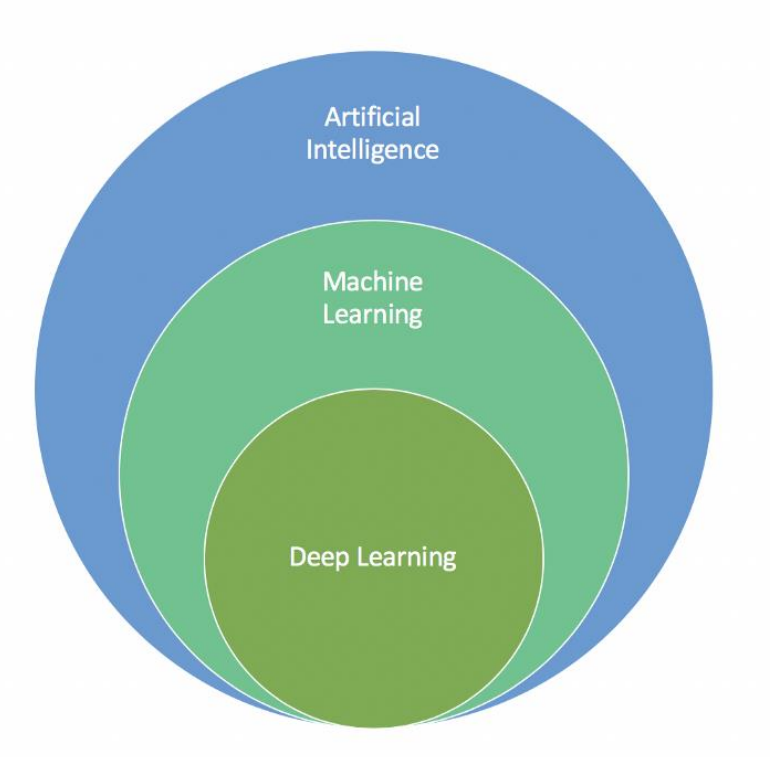
1. Strojno učenje

1.1. Motivacija i definicija

Strojno učenje definiramo kao primjenu umjetne inteligencije, to jest strojnim učenjem želimo postići da naš sustav preko prethodno stečenih znanja i iskustva odlučuje. Sljedećim citatom možemo dobro predstaviti strojno učenje.

„Ono što je zajedničko mnogim definicijama jest spoznaja da danas živimo u svijetu u kojem smo okruženi obiljem podataka, te da nam je interesantno razvijati programske sustave koji su sposobni iskorištavati te podatke, učiti iz njih i na temelju toga nuditi korisna ponašanja.“ – [1]

Kako je i u citatu navedeno, za provedbu odlučivanja i stjecanja znanja trebamo imati podatke ili iz samog kompleta podataka želimo uočiti uzorak i saznati nešto više o tim podacima. Raspoloživi podaci mogu biti numerički ili kategorički. Primjer numeričkih podataka su godine starosti čovjeka, cijene, broj ljudi na koncertu, dok su primjeri kategoričkih podataka brand odjeće, marka automobila, boja očiju, itd.



Slika 1.1 Odnos umjetne inteligencije, strojnog učenja i dubokog učenja [13]

1.2. Vrste strojnog učenja

Strojno učenje dijelimo na tri glavna područja, a to su:

- Nadzirano učenje
- Nenadzirano učenje
- Podržano učenje

1.2.1. Nadzirano učenje

Nadzirano učenje se definira skupom ulaznih podataka x koji se preslikavaju na izlaz y, to jest naš skup za učenje je oblika $D = \{(x_i, y_i)\}_{i=1}^{N}$, gdje x_i predstavlja podatke o primjerku, a y_i predstavlja ciljnu vrijednost koju pridružuje tom primjerku (engl. *target value*). Zapravo tražimo presliku $\hat{y} = f(x)$, gdje ako je y_i diskretna/nebrojčana vrijednost radimo klasifikaciju, a ako je y_i kontinuirana/brojčana vrijednost radimo regresiju. Umjetne neuronske mreže generalno spadaju u ovu vrstu strojnog učenja, iako postoje i nenadzirane vrste neuronskih mreža.

1.2.2. Nenadzirano učenje

Nenadzirano učenje definiramo samo skupom ulaznih podataka, skup učenja je onda oblika $D = \{(x_i)\}_{i=1}^{N}$, vidimo da naš skup onda ne prima y_i, odnosno ciljne vrijednosti uz ulazne. U postupke nenadziranog učenja spadaju postupci grupiranja (engl. *clustering*), postupci smanjena dimenzionalnost (engl. *dimensionality reduction*), postupak otkrivanja stršećih/novih vrijednosti (engl. *outlier/novelty detection*).

1.2.3. Podržano učenje

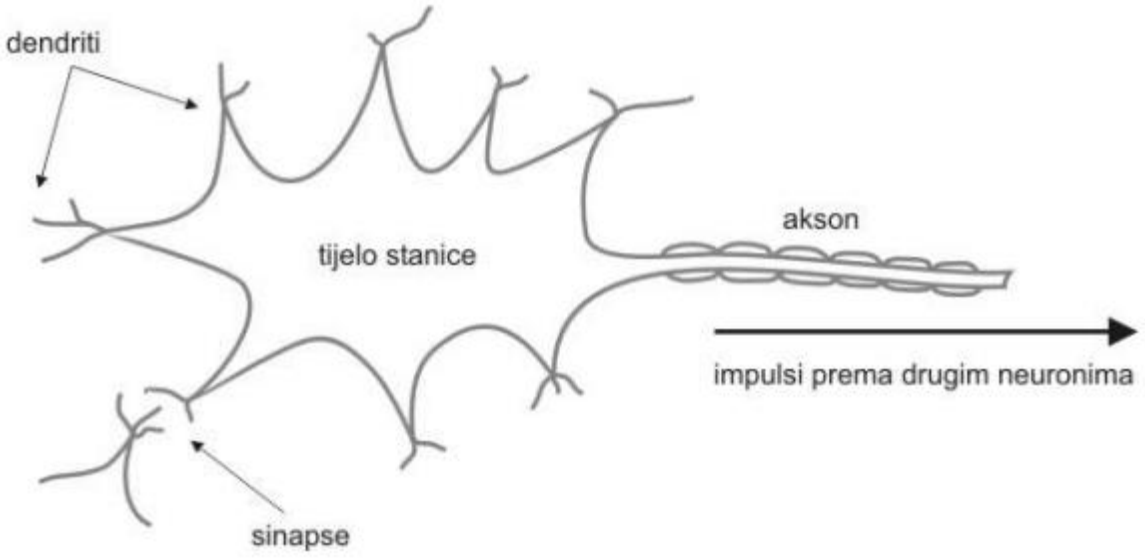
Podržano učenje se bavi optimizacijom ponašanja. Odnosno zadaća podržanog učenja je razviti optimalnu strategiju ponašanja na temelju pokušaja s odgođenom nagradom.

2. Umjetne neuronske mreže

2.1. Motivacija i definicija

Umjetne neuronske mreže ili skraćeno neuronske mreže, su model dubokog učenja inspiriran biološkim neuronima koji su sastavni dio mozga i živčanog sustava. U sljedećoj usporedbi možemo vidjeti biološku inspiraciju za model umjetnog neurona.

2.1.1. Biološki neuron

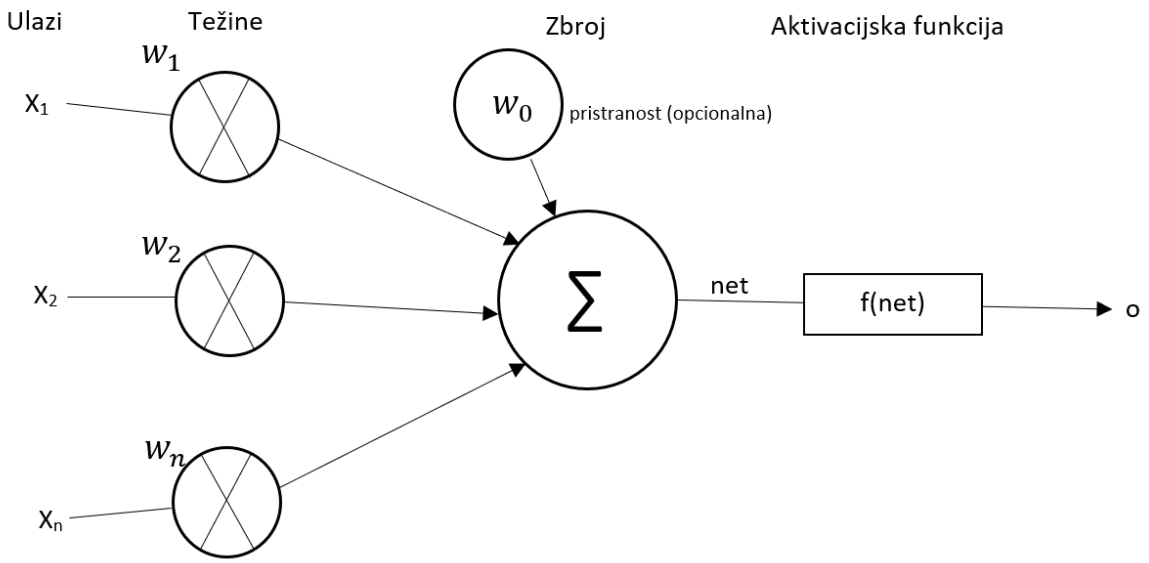


Slika 2.1 Biološki neuron [6]

Na slici 2.1 se nalazi biološki neuron, koji se sastoji od tijela u kojem se u središtu nalazi jezgra, a na rubovima tijela se nalaze dendriti koji primaju impulse od susjednih neurona. Prema desno se širi akson, na kraju kojeg se nalaze teledendroni sa završnim kvržicama koji šalju impulse dalje u susjedne neurone.

2.2. Umjetni neuron

Inspirirani biološkim modelom, 1943. godine Warren McCulloh i Walter Pitts definiraju matematički model umjetnog neurona prikazanog na slici 2.2.



Slika 2.2 Model umjetnog neurona (engl. perceptron)

Ulazi x_1, x_2, …, x_n predstavljaju dendrite biološkog neurona, te oni sadrže određene vrijednosti pobude prethodnog neurona. Težine w_1, w_2, …, w_n predstavljaju u kojoj jačini signal primljen kroz ulaz x_i utječe na neuron. Uz to postoji i težina koju predstavlja konstantan pomak (engl. *bias*) w_0.

$$net = \sum_{i=1}^{n} w_i x_i + w_0 \tag{2.1}$$

Tijelo stanice umjetnog neurona prema svojem izlazu generira sumu označenu sa *net* , te se ona računa prema izrazu 2.1. Konačni izlaz *o* ovisi o aktivacijskoj funkciji *o* = *f* (*net*).

2.2.1. Aktivacijske funkcije

Prilikom konstrukcije modela neuronske mreže možemo koristiti različite aktivacijske funkcije kao što su to funkcija skoka, funkcija identiteta, sigmoidna funkcija, hiperbolički tangens, *ReLU* funkcija. Kao primjer aktivacijske funkcije možemo uzeti *funkciju skoka*, neuron koji onda dobivamo poznat je kao *TLU-perceptron*, te funkciju onda definiramo kao

$$step(net) = \begin{cases} 0, & net < 0 \\ 1, & net \geq 0 \end{cases} \tag{2.2}$$

Aktivacijska funkcija koju većinski koristimo u povratnim neuronskim mrežama je hiperbolička tangens funkcija (*tanh*):

$$tanh(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}} \tag{2.3}$$

tanh aktivacijska funkcija vraća vrijednosti između -1 i 1. Osim *tanh* funkcije, LSTM model interno koristi i *sigmoidnu* funkciju oblika:

$$\sigma(net) = \frac{1}{1 - e^{-net}} \tag{2.4}$$

, σ funkcija vraća vrijednosti između 0 i 1. Aktivacijske funkcije *tanh*, σ i *ReLU* će nam biti bitne kod modela povratnih neuronskih mreža .

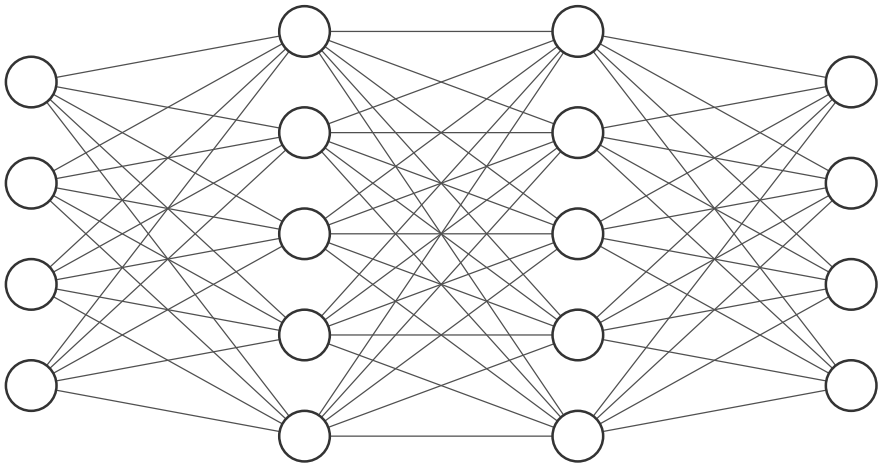
2.2.2. Vrste neuronskih mreža

Iako neuronske mreže možemo podijelite na dosta načina, ugrubo ih dijelimo na:

- Jednoslojne mreže bez povratnih veza (engl. *single-layer feedforward networks*)
- Višeslojne mreže bez povratnih veza (engl. *multi-layer feedforward networks*)
- Povratne neuronske mreže (engl. *recurrent neural networks - RNN*)
- Konvolucijske neuronske mreže (engl. *convolutional neural networks - CNN*)

2.3. Višeslojne mreže bez povratnih veza

Nakon definicije umjetnog neurona, samim time i jednoslojne mreže bez povratnih veza, možemo pričati o višeslojnom modelu umjetnih neuronskih mreža bez povratnih veza čija su oni glavna sastavnica.



Slika 2.3 Višeslojna neuronska mreža

Na slici 2.3 se nalazi jedna neuronska mreža s četiri sloja. Ulazni sloj čine četiri neurona i kroz njih mreži predajemo ulazne podatke. Zatim ga slijede dva skrivena sloja, svaki po pet neurona. Te kao zadnji sloj imamo izlazni sloj s četiri neurona iz kojih izvlačimo rezultate. Između svakog sloja mreže imamo težine s kojima su čvorovi povezani s onima iz prethodnog i sljedećeg sloja. Skriveni slojevi nisu vidljivi korisniku, jedini koji su u interakciji sa korisnikom su ulazni i izlazni.

Kao ulaz, neuronskoj mreži predočavamo podatke iz naših skupova podataka, te iz njih pokušavamo dobiti izlaze koji će odgovarati ciljanoj regresiji ili klasifikaciji.

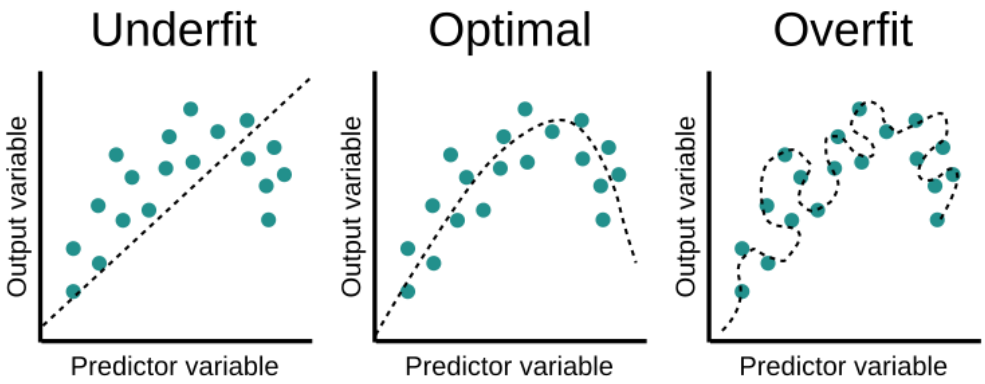
Generalno rad s umjetnim neuronskih mrežama dijelimo u dvije faze: fazu učenja i fazu iskorištavanja. U fazi učenja neuronskoj mreži predočavamo uzorke iz skupa za učenje, gdje je jedna *iteracija* predočavanje jednog uzorka skupa, a jedna *epoha* predočavanje čitavog skupa uzoraka. Uslijed učenja dolazi do promjena u jakosti veza između neurona, time se mreža onda prilagođava viđenim podacima, zatim ju u fazi iskorištavanja koristimo.

2.4. Učenje

Cilj faze učenja neuronske mreže je postići najbolju raspodjelu težina između neurona. Kao što je i navedeno, postoji više vrsti učenja neuronske mreže, dok ćemo u implementaciji koristiti nadzirano učenje. Ulaz nadziranog učenja smo prethodno definirali kao par (x_i, y_i), gdje x_i predstavlja podatke o primjerku, a y_i predstavlja ciljnu vrijednost koju pridružuje tom primjerku.

2.4.1. Unakrsna provjera

Unakrsna provjera (engl. *cross-validation*) je način pristupa učenju modela strojnog učenja. Prilikom učenja modelu predajemo skup podataka koji nazivamo skupom za učenje (engl. *train set*), te pomoću tog skupa, sa funkcijom gubitka i optimizacijom neuronske mreže prilagođavamo težine između čvorova neuronske mreže. Osim skupa za učenje, imamo skup za provjeru (engl. *test set*) koji nam služi za provjeru koliko dobro naš model radi. Za veći skup podatak generalno uzimamo omjer 80% podataka za skup za učenje, a 20% podataka kao skup za provjeru. Za manji skup podataka bolja podjela bi bila 70/30 ili 60/40. Važnost unakrsne provjere je ta da izbjegnemo prekomjerno prilagođavanje modela (engl. *overfitting*). Ako dođe do prekomjernog prilagođavanja našeg modela onda on loše generalizira, to jest na još neviđenim podacima će reagirati loše.



Slika 2.4 Nedovoljno, optimalno i prekomjerno prilagođavanja regresijske funkcije [2]

2.4.2. Funkcija gubitka

Prilikom učenja neuronske mreže, u svakoj iteraciji izlaz uspoređujemo sa ciljnim izlazom, te kako bismo „ocijenili“ koliko dobro mreža procjenjuje ciljnu vrijednost *y*, koristimo funkciju gubitka (engl. *loss function*) označenu sa *L*. Za odabir funkcije gubitka imamo više različitih opcija poput:

- Regresijske funkcije gubitka
  - Srednja kvadratna pogreška (engl. *Mean squared error loss*)
  - Srednja kvadratna logaritamska pogreška (engl. *Mean squared logarithmic error loss*)
  - Srednja apsolutna pogreška (engl. *Mean absolute error loss*)
- Binarne klasifikacijske funkcije gubitka
  - Gubitak zglobnice (engl. *Hinge loss*)
  - Kvadrirani gubitak zglobnice (engl. *Squared hinge loss*)
  - Binarna unakrsna entropija (engl. *Binary cross-entropy*)
- Višeklasne klasifikacije funkcije gubitka

Pošto se u ovom radu implementirani model bavi regresijom i kontinuiranim vrijednostima cijena dionica, koristimo funkciju srednje kvadratne pogreške (engl. *mean squared error - MSE*).

$$L = \frac{1}{N}\sum_{1}^{N}(y_i - \hat{y}_i)^2 \tag{2.5}$$

Formula (2.5) prikazuje izračun standardne pogreške kvadrata, gdje su $\hat{y}_i$ vrijednosti koje nam daje neuronska mreža, a y_i ciljne vrijednosti.

Nakon izračuna pogreške u svakoj iteraciji, cilj nam je prilagoditi neuronsku mrežu tako što mijenjamo težine veza između umjetnih neurona. Ovaj postupak nazivamo algoritmom propagacije pogreške unatrag (engl. *backpropagation algorithm*), odnosno postupak optimizacije neuronske mreže tako da minimiziramo funkciju gubitka.

2.4.3. Optimizacija neuronske mreže

Prilikom optimizacije mreže, odnosno smanjenja funkcije gubitka, imamo dvije mogućnosti optimizacije:

- Računamo gradijent na temelju svih primjera i zatim korigiramo težine veza
- Za određeni broj primjera (engl. *batch size*) računamo gradijent i odmah korigiramo težine veza

U implementaciji našega modela koristimo drugu navedenu metodu, odnosno za određeni broj primjera računamo prilagođeni gradijent i odmah korigiramo.

Za optimizaciju mreže postoje različiti algoritmi kao što su:

- Standardni gradijentni spust (engl. *gradient descent*)
- Stohastički gradijentni spust (engl. *stohastic gradient descent*)
- Adaptivni gradijent (engl. *adaptive gradient – AdaGrad*)
- AdaDelta
- RMSprop
- Adam

Prema provedenim eksperimentima [3], *Adam* nadmašuje ostale algoritme. Iako postoje argumenti da standardni gradijentni spust bolje generalizira [4], generalno najbolji za našu implementaciju je adaptivni optimizacijski algoritam *Adam* zbog najbrže konvergencije, te je memorijski efektivan za veliki skup podataka, zbog toga ćemo i prilikom implementacije modela koristiti njega. Izgled formule za standardni gradijentni spust je sljedeći:

$$\theta = \theta - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{2.6}$$

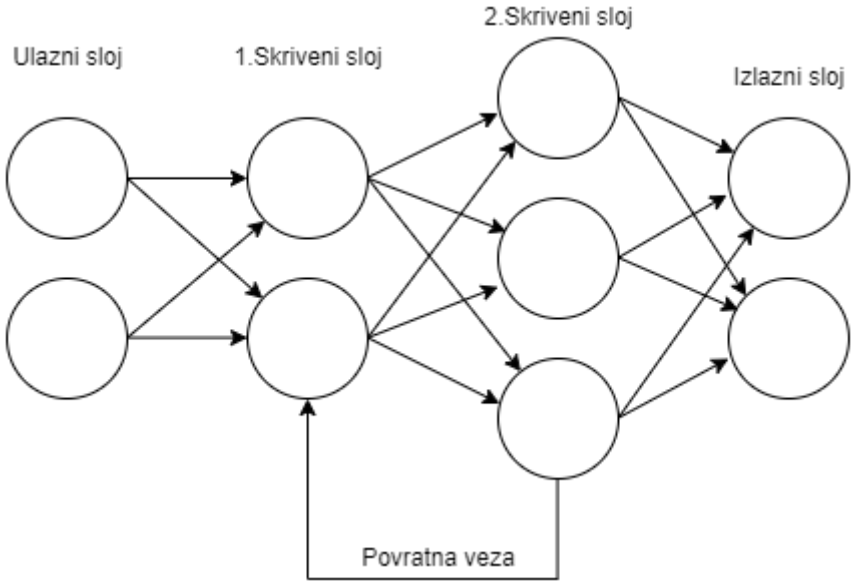
, gdje α predstavlja *stopu učenja*, a $\frac{\partial}{\partial \theta_j} J(\theta)$ predstavlja gradijent funkcije u θ_j.

Iako ga koristimo kasnije u samoj implementaciji, matematičku pozadinu *Adam* optimizacijskog algoritma nećemo navoditi u ovom radu zbog njene kompleksnosti. Za više o *Adam* optimizacijskom algoritmu i njegovoj matematičkoj pozadini pogledati [3].

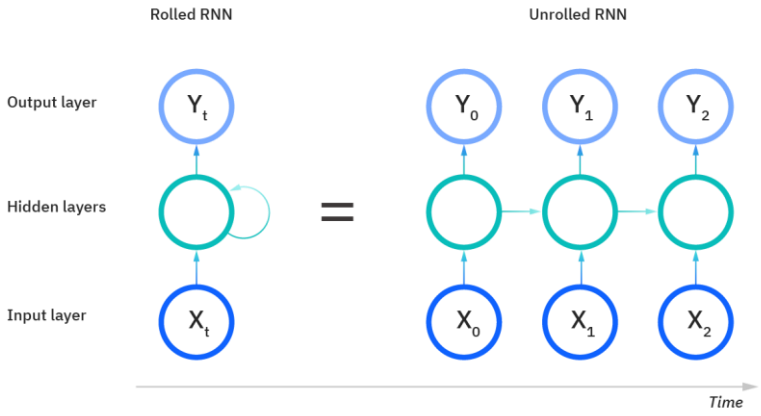
2.5. Povratne neuronske mreže

2.5.1. Definicija

Povratne neuronske mreže su vrsta umjetnih neuronskih mreža koje sadrže povratne veze između neurona, što im omogućava privremeno dinamičko ponašanje. Povratne neuronske mreže inspiraciju dobivaju iz dinamičkih sustava. Koristimo ih jer su idealne kod modeliranja slijednih podataka, kao što su to naprimjer tekst, zvuk, cijene dionica itd.



Slika 2.5 Povratna neuronska mreža



Slika 2.6 „Spakirana“ i „razmotana“ povratna neuronska mreža [5]

Osim slike 2.5, povratne neuronske mreže možemo prikazati kao i na slici 2.6, u njihovom „spakiranom“ stanju i „razmotanom“ stanju koji prikazuje neuronsku mrežu kroz vremenske korake *t*. Na slici 2.6 plavim čvorovima su označeni ulazi x_t, zelenim čvorovima skriveni slojevi h_t, te svjetloplave boje izlazi y_t.

Definirajmo h_t kao vektor skrivenog sloja u vremenskom koraku (engl. *timestep*) *t*, h_t vrijednosti onda računamo idućom formulom:

$$h_t = f(x_t, h_{t-1}) \tag{2.7}$$

Iz formule vidimo da stanje h_t ovisi o vrijednost stanja u prethodnom koraku h_{t-1} i o trenutačnoj vrijednosti vektora ulaza x_t. U jednostavnoj povratnoj neuronskoj mreži funkcija *f* je jednoslojna neuronska mreža.

$$net = W_h h_{t-1} + W_x x_t + b_h \tag{2.8}$$

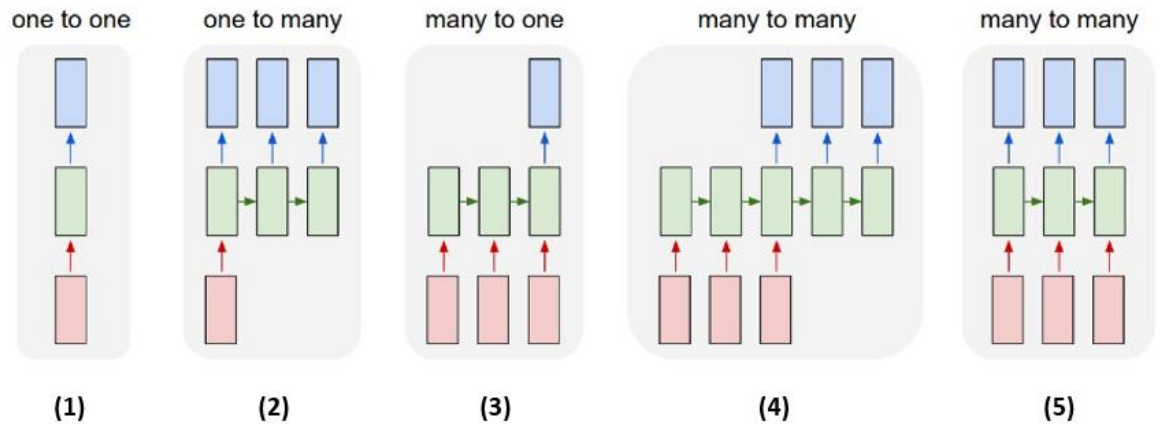
$$h_t = g(net) \tag{2.9}$$

Vrijednost *net* sloja h_t računamo prema formuli 2.8, gdje su W_h i W_x pripadajuće matrice težina između neurona, a b_h je vektor pristranosti, odnosno konstantni pomak. Za računanje projekcije u izlazni sloj koristimo sljedeću formulu:

$$y_t = g(W_y h_t + b_y) \tag{2.10}$$

Kod povratnih mreža za funkciju g uzimamo aktivacijske funkcije koje su prethodno definirane u poglavlju *2.2.1. Aktivacijske funkcije*, a to su *tanh* (2.3), σ (2.4) i ReLU koju jednostavno definiramo kao $ReLU(net) = \max(0, net)$.

2.5.2. Vrste povratnih neuronskih mreža



Slika 2.7 Vrste povratnih mreža [7]

Na slici 2.7 pravokutnici predstavljaju vektore, a strelice predstavljaju funkcije (npr. matrično množenje). Ulazni vektori su crvene boje, izlazni plave boje, a zeleni predstavljaju RNN stanja. Slika označena sa **(1)** nam predstavlja fiksiran ulaz i izlaz bez povratnih veza, kao primjer problema koji u to spada je klasifikacija slika. Nadalje slika **(2)** prikazuje slijedni izlaz, u što spada opis slike riječima. Brojem **(3)** označavamo slijedni ulaz i fiksiran izlaz, npr. rečenicu moramo klasificirat kao „pozitivna“ ili „negativna“. Na slici **(4)** imamo slijedni ulaz i izlaz, npr. prijevod rečenice iz jednog jezika u drugi. I na zadnjoj slici **(5)** imamo usklađeni slijedni ulaz i izlaz, npr. klasifikacija svake sličice u videu. U svakom od navedenih slučaja uviđamo da nema ograničenja na količinu zelenih pravokutnika koji predstavljaju povratnu transformaciju. Detaljnije u [7].

2.5.3. Problemi običnih povratnih neuronskih mreža

U nedostatke i probleme povratnih neuronskih mreža spada zahtjevno i sporo računanje, učenje može biti kompleksno, mreža ne uzima buduće ulaze u svrhu odlučivanja, itd. No glavni problem običnih povratnih mreža je da pati od problema kratke memorije, ako je slijedni ulaz dug, onda će mreža imati problem „prisjećanja“ ranijih ulaznih podataka. Konkretnije, za vrijeme propagacije pogreške unazad , dolazi do problema eksplodirajućeg i nestajućeg gradijent.

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 \ll 1 \tag{2.11}$$

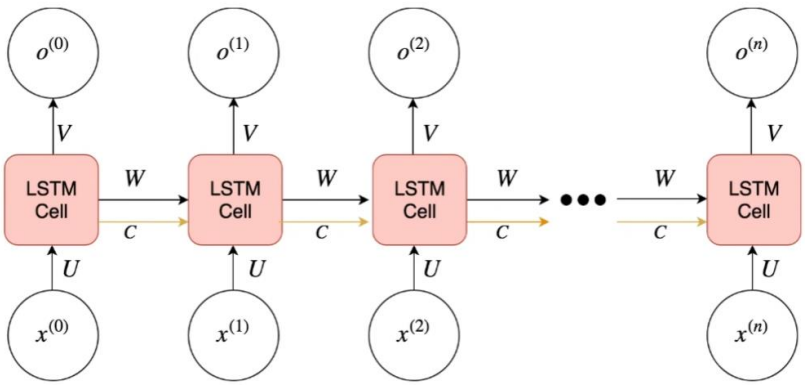
$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 \gg 1 \tag{2.12}$$

Problem nestajućeg gradijenta (2.11) je taj da kada želimo optimizirati mrežu nakon nekog ulaza i izračuna pogreške, gradijent između dva uzastopna skrivena stanje se eksponencijalno brzo smanji u vrijednosti i tako ne prilagodi ostatak mreže, što znači da kod tog dijela mreže neće doći do prilagođavanja težina. Kod problema eksplodirajućeg gradijenta (2.12) naš gradijent eksponencijalno postane ogroman. Više o problemima povratnih mreža i o matematičkoj pozadini problema nestajućeg i eksplodirajućeg gradijenta u [8].

Kao rješenje problema s kratkom memorijom i nestajućim/eksplodirajućim gradijentom koristimo podvrste povratnih mreža s propusnicama (engl. *gated recurrent neural networks*), kao što su *Long Short-Term Memory* (LSTM) i *Gated Recurrent Units* (GRU). Pošto u implementaciji koristimo LSTM model, on će biti detaljnije objašnjen u nastavku. Za detaljnije o povratnim mrežama s propusnicama pogledati [9].

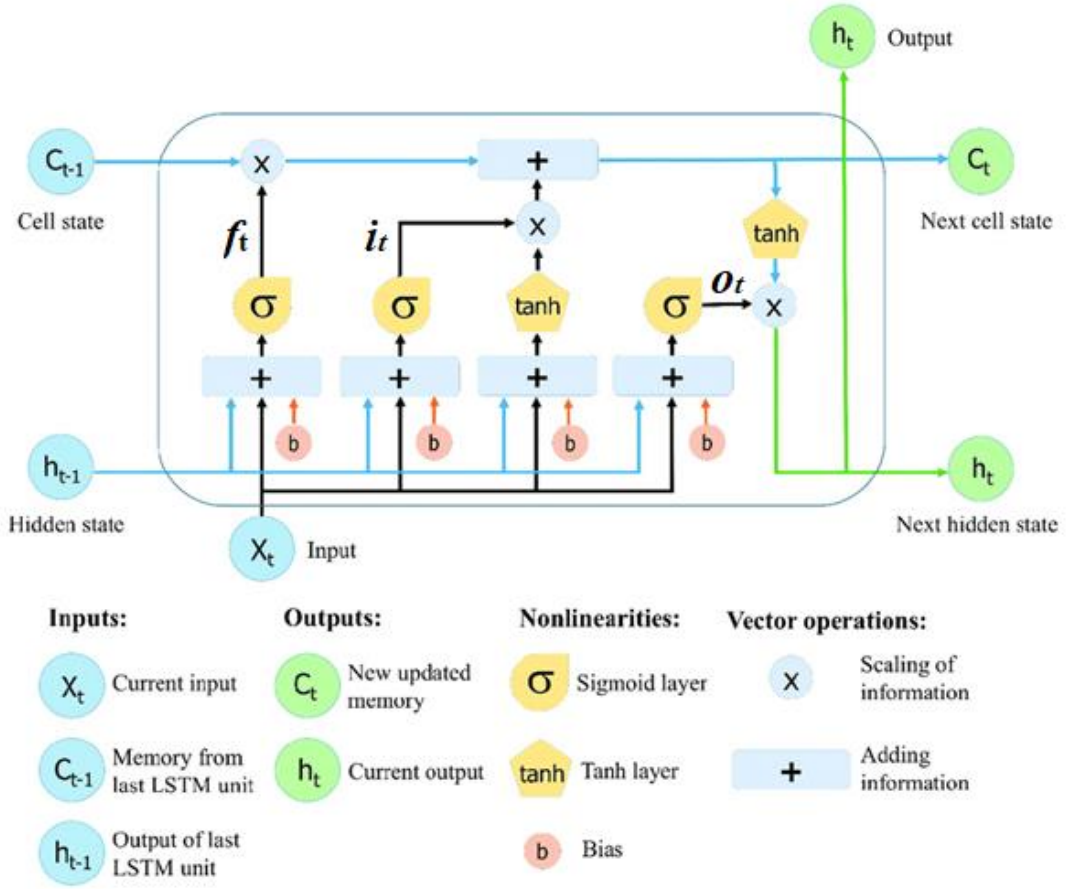
2.5.4. LSTM mreže

LSTM ili *Long Short-Term Memory* spada u podvrstu povratnih neuronskih mreža s propusnicama. Koristi se za prepoznavanje slijeda riječi, opis slike, video-tekst konverziju i ostale slične probleme. LSTM je napravljen u svrhu rješavanja problema kratkotrajne memorije koju obična povratna neuronska mreža ima. Da bismo bolje razumjeli kako se LSTM „bori“ protiv kratkotrajne memorije, to jest nestajućeg gradijenta, u nastavku objašnjavamo njegovu strukturu.



Slika 2.8 Prikaz povratne neuronske mreže sa LSTM ćelijama u skrivenom sloju

Na slici 2.8 vidimo izgled povratne neuronske mreže sa LSTM ćelijama, uz obične slojeve i već definirane parametre uočavamo i dodatnu povezanost između ćelija označenu sa *C* koja označava stanje prethodne ćelije.



Slika 2.9 Interna struktura LSTM ćelije (napomena: ulazi i izlazi W i C na slici 2.8 su zamijenjeni u odnosu na ovu sliku) [10]

LSTM ćelija sadrži tri propusnice. Ulaznu propusnicu (engl. *input gate*), označenu sa i_t na slici 2.9, koja provjerava je li ćelija ažurirana, računamo ju formulom:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.13}$$

Zatim propusnicu zaborava (engl. *forget gate*), označenu sa f_t na slici, koja provjerava je li memorija postavljena na nula, nju računamo:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.14}$$

Te još definiramo i izlaznu propusnicu (engl. *output gate*), označenu sa o_t na slici, koja kontrolira je li informacija trenutne ćelije vidljiva, njena formula je:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.15}$$

Uočavamo da sve funkcije za svoju aktivacijsku funkciju koriste *sigmoidnu* funkciju, nju koristimo iz razloga da naše funkcije sačinjavaju „glatke" krivulje u rangu od 0 do 1, time model ostaje diferencijabilan. Osim navedenih funkcija propusnica imamo i vektor $\bar{C}_t$ koji ažurira stanje ćelije, njega računamo kao:

$$\bar{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C) \tag{2.16}$$

Ovdje pak koristimo *tanh* aktivacijsku funkciju, zbog njenog ranga koji je centriran oko 0, odnosno ide od -1 do 1, ona distribuira gradijente poprilično dobro, što dozvoljava da informacija o stanju ćelije traje dulje bez nestajanja ili eksplodiranja u vrijednosti. Vrijednost stanja C_t onda modeliramo formulom:

$$C_t = f_t C_{t-1} + i_t \bar{C}_t \tag{2.17}$$

Direktno iz formule 2.17 vidimo da stanje C_t ovisi o *input gate* funkciji i_t koju primjenjujemo na $\bar{C}_t$, jer ona odlučuje hoćemo li ažurirati ćeliju. Osim toga, stanje C_t ovisi i o *forget gate* funkciji f_t koju primjenjujemo na prethodno stanje C_{t-1}, jer ona određuje koliko prošlog stanja zaboravljamo. Konačnu definiramo formulu s kojom računamo vrijednost skrivenog vektora h_t:

$$h_t = \tanh(C_t) \times o_t \tag{2.18}$$

Za dobivanje vrijednosti h_t, na vrijednost stanja ćelije C_t primjenjujemo *output gate* funkciju o_t.

Za još detaljnije shvaćanje svakog koraka i matematičkih definicija LSTM ćelije pogledati [7], [11] i [12].

3. Programsko ostvarenje modela za predviđanje

3.1. Opis problema predviđanja cijena dionica

Predviđanje kretanja tržišta dionica je pokušaj određivanja budućih cijena dionica. Jedan od razloga zašto predviđamo može biti ostvarivanje profita prilikom kupovanja i prodaje dionica. Usprkos volatilnosti, cijene dionica nisu nasumične vrijednosti, odnosno one najviše ovise o vanjskim utjecajima iz kojih možemo uočiti razlog promjene njihove vrijednosti. *Učinkovita hipoteza na tržištu* (engl. *efficient-market hypothesis*) je bitan ekonomski pojam za naglasiti, on tvrdi da trenutna vrijednosti imovine na tržištu odražava sve raspoložive podatke (povijesne, javne i privatne). Drugim riječima hipoteza govori da svaka buduća vijest ili događaj koji može utjecati na cijenu imovine će prouzročiti prilagodbu cijene tako brzo da je nemoguće dobiti ekonomsku korist od nje. [14]

3.2. Različite metode pokušaja predviđanja dionica

Osim korištenja neuronskih mreža kako bismo predvidjeli kretanje cijena dionica, u nastavku spominjemo i druge metode. Drugi načini na koje bi mogli predviđati cijene su da gledamo neke druge parametre i uvjete, te zbog njih očekujemo određeni obrazac ponašanja. Možemo navesti četiri primjera analitičkog ili vanjsko faktorskog predviđanja:

- Momentum
- Povrat srednje vrijednosti (engl. *Mean Inversion*)
- *Martingale*
- Potraga za vrijednošću

Momentum je kada očekujemo da će cijene pratiti trend trenutačnog gibanja bio to porast ili pad, pa sukladno tome reagiramo na tržište ulaganjem ili prodavanjem. Povrat srednje vrijednosti je teorija koja sugerira da kroz vrijeme vrijednost neke imovine dugoročno teži srednjoj vrijednosti skupa ukupnih vrijednosti. Martingale je investicijska strategija koja tvrdi da svaki put kada izgubimo „okladu“ udvostručujemo uloženi iznos, te time nastojimo vratiti izgubljeni kapital. Potraga za vrijednošću znači da pokušavamo što ranije pronaći dionice nečega u čemu vidimo kvalitetu dok su još jeftine, te ulažemo u njih jer znamo da će postati skuplje. Više o ovim metodama u [15].

3.3. Izgradnja modela neuronske mreže

3.3.1. Alati i tehnologije

Realizaciju našega modela za predviđanje ostvarujemo u programskom jeziku *Python* zbog raznih biblioteka i programske podrške koja nam olakšava pri implementaciji kompleksnih modela strojnog učenja.

Pythonove biblioteke koje se koriste u radu su:

- *numpy*
- *pandas*
- *sklearn*
- *tensorflow (keras)*

Biblioteku *numpy* koristimo u svrhu pretvaranja neobrađenih podataka u vektore ulaza i izlaza u naš model neuronske mreže. *Pandas* koristimo za pretvaranje podataka u *dataframeove* i za vađenje podataka o stanju cijena dionica, u našem slučaju ćemo ih izvlačiti preko *yahoo finance* API. *Sklearn* ili *scikit-learn* je besplatna pythonova biblioteka strojnog učenja, konkretnije mi ćemo koristiti njenu funkciju skaliranja (engl. *MinMaxScaler*) potrebnu za skaliranja ulaznih podataka prije unosa u neuronsku mrežu. Još koristimo i *tensorflow* biblioteku koja nam je najbitnija za izgradnju našega modela, konkretnije *keras* ima implementirane metode i objekte koje iskorištavamo da bismo složili našu željenu neuronsku mrežu.

3.3.2. Pregled podataka cijena dionica

Kao što je i prethodno navedeno, podatke o cijenama dionica uzimamo iz *yahoo finance* API-ja koristeći biblioteku *pandas*. Izgled podataka o dionica vidimo na tablici 3.1.

Date	High	Low	Open	Close	Volume	Adj Close
2012-05-18	45.000000	38.000000	42.049999	38.230000	573576400	38.230000
2012-05-21	36.660000	33.000000	36.529999	34.029999	168192700	34.029999
2012-05-22	33.590000	30.940001	32.610001	31.000000	101786600	31.000000
2012-05-23	32.500000	31.360001	31.370001	32.000000	73600000	32.000000
2012-05-24	33.209999	31.770000	32.950001	33.029999	50237200	33.029999
...	...	...	...	...	...	...
2019-12-24	206.789993	205.000000	206.300003	205.119995	6046300	205.119995
2019-12-26	207.820007	205.309998	205.570007	207.789993	9350700	207.789993
2019-12-27	208.929993	206.589996	208.669998	208.100006	10284200	208.100006
2019-12-30	207.899994	203.899994	207.860001	204.410004	10524300	204.410004
2019-12-31	205.559998	203.600006	204.000000	205.250000	8953500	205.250000

Tablica 3.1 Izgled podataka o dionicama

Date označava datum (subotom i nedjeljom tržište dionica je zatvoreno, pa tih datuma nemamo). *High* označava najveću vrijednost dionice u tom danu, *Low* označava najmanju vrijednost u danu. *Open* označava vrijednost dionice nakon otvaranja tržišta, *Close* označava zadnju vrijednost prije zatvaranja tržišta. *Volume* označava količinu razmijenjenih dionica u tom danu. *Adj Close* je *Close*, ali prilagođen tako da uzimamo dodatne faktore u račun kao što su dividendi, podjelu dionica (engl. *stock split*), itd. Iako nema velike razlike između vrijednosti *Close* i *Adj Close*, u implementaciji koristimo *Adj Close*.

3.3.3. Pregled bitnih dijelova programske realizacije

```
(1)data = pdr.DataReader(company, 'yahoo', start=start_date, end=end_date)
(2)scaler = MinMaxScaler(feature_range=(0,1))
(3)scaled_data = scaler.fit_transform(data['AdjClose'].values.reshape(-1, 1))
(4)for i in range(timestep, len(scaled_data)):
(5)           x_train.append(scaled_data[i-timestep:i, 0])
(6)           y_train.append(scaled_data[i, 0])
(7)x_train, y_train = np.array(x_train), np.array(y_train)
(8)x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

Kod 3.2 Dobavljanje i pripremanje podataka za treniranje

Redom objašnjavamo kod 3.1: (1) Učitavamo podatke s *yahoo finance* API, (2)(3) Skaliramo podatke u vrijednosti između 0 i 1 (glavni razlog je taj što nam mreža onda brže konvergira), (4)(5)(6) ovisno o *timestepu* punimo ulazne i odgovarajuće ciljne vrijednosti u varijable *x_train* i *y_train*, (7)(8) pretvaramo ulazne i ciljne vrijednosti u vektore.

```
(1)model = Sequential([
(2)LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)),
(3)Dropout(0.2),
(4)LSTM(units=50, return_sequences=True),
(5)Dropout(0.2),
(6)LSTM(units=50),
(7)Dropout(0.2),
(8)Dense(units=1)
   ])
(9)model.compile(optimizer='adam', loss='mean_squared_error')
(10)model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size)
```

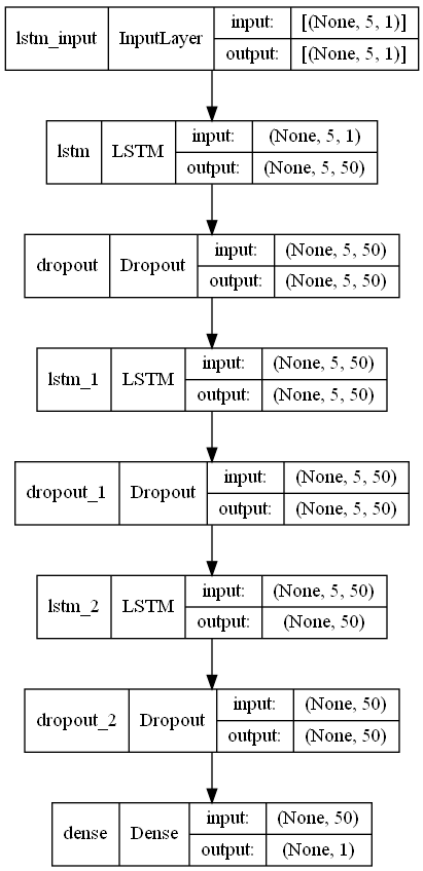
Kod 3.3 Izgradnja neuronske mreže i treniranje

Prilikom realizacije modela (1) koristimo *Sequential* klasu koja nam dozvoljava kreiranje neuronske mreže po slojevima. U liniji (2) definiramo LSTM sloj kojemu pridodajemo izgled ulaznog vektora preko parametra *input_shape*. *Units* parametar određuje dimenzije izlaza. Boolean parametar *return_sequence* određuje vraćamo li zadnji izlaz u izlaznom slijedu ili pak cijeli slijed. Kao zadnji izlazni sloj (8) koristimo obični NN sloj veličine 1 koji izbacuje vrijednost predviđanja. Između „gradivnih" slojeva naše mreže koristimo i *Dropout* slojeve (3)(5)(7) koji su još jedan način obrane od prekomjernog prilagođavanja naše mreže, tako što po određenoj stopi (u našem slučaju 20%) nasumično onemogućava neurone i njihove odgovarajuće veze nakon svakog skrivenog sloja mreže kao što je prikazano u izgledu mreže na slici 3.3. Na liniji (9) konfiguriramo koji optimizacijski algoritam (*optimizer*) i koju funkciju gubitka (*loss*) koristiti prilikom učenja mreže, detaljnije o optimizacijskom algoritmu u poglavlju *2.4.3 Optimizacija neuronske mreže*, a detaljnije o funkciji gubitka u poglavlju *2.4.2 Funkcija gubitka*. Metoda *fit* pokreće učenje našega modela mreže, u koji predajemo ulazne vrijednosti, ciljne vrijednosti, *epochs* i *batch_size*. *Epochs* parametar određuje koliko puta ćemo provesti naš skup ulaznih podataka ponovo kroz model za vrijeme učenja. *Batch_size* predstavlja broj primjeraka nakon kojih će gradijent ažurirati mrežu.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm (LSTM)                  (None, 5, 50)             10400
 dropout (Dropout)            (None, 5, 50)             0
 lstm_1 (LSTM)                (None, 5, 50)             20200
 dropout_1 (Dropout)          (None, 5, 50)             0
 lstm_2 (LSTM)                (None, 50)                20200
 dropout_2 (Dropout)          (None, 50)                0
 dense (Dense)                (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
```

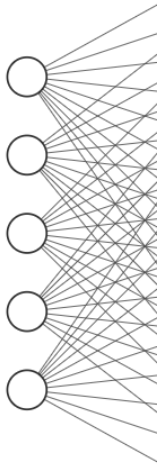
*Tablica 3.1 Sažetak našeg modela (*model.summary()*)*

Osim izlaza metode sažetka modela prikazanog na tablici 3.2, koristimo *keras* biblioteku *vis_utils* za dobivanje izgleda našega modela kao na slici 3.1.



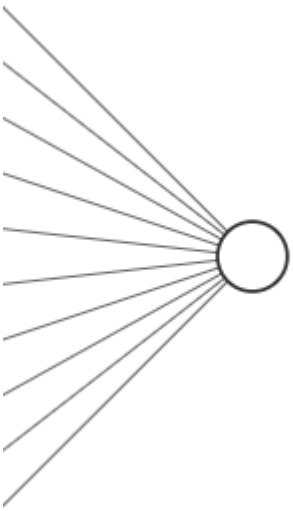
Slika 3.1 Izgled slojeva našega modela

Izgled ulaznog sloja ovisi o našem ulaznom parametru *timestep*, čija vrijednost definira broj ulaznih čvorova kao broj dana koji koristimo da predvidimo idući dan. Kao primjer ulaza navodimo sliku 3.2.



Slika 3.2 Ulazni sloj modela ako je vrijednost parametra timestep jednaka 5

Dok će izlazni sloj uvijek imati točno jedan izlazni čvor jer će to biti vrijednost predviđanja jednog dana.



Slika 3.3 Prikaz izlaznog sloja modela

Nakon učenja, spremni smo unositi naše vrijednosti skupa za provjeru. Skup za provjeru učitavamo na isti način kao i skup za test u kodu 3.1, no u ovom slučaju nam ne trebaju ciljne vrijednosti jer njih pokušavamo sami dobiti.

```
(1)predicted_prices = model.predict(x_test)
(2)predicted_prices = scaler.inverse_transform(predicted_prices)
```

Kod 3.3 Dobivanje predviđenih cijena iz modela

Predict metodi (1) predajemo podatke, a model nam vraća neskalirane vrijednosti. U liniji (2) nazad skaliramo vrijednosti u one prave i spremni smo za prikazivanje rezultata.

```
Input->[[205.1199951171875],[207.78999328613284],[208.1000061035156],
[204.41000366210935], [205.25000000000003]]

Prediction -> 205.48651123046875

Real value -> 209.77999877929688

Absolute Error -> 4.293487548828125


Input -> [[207.78999328613284], [208.1000061035156], [204.41000366210935],
[205.25000000000003], [209.7799987792969]]

Prediction -> 206.10516357421875

Real value -> 208.6699981689453

Absolute Error -> 2.5648345947265625


Input -> [[208.1000061035156], [204.41000366210935], [205.25000000000003],
[209.7799987792969],[208.6699981689453]]

Prediction -> 205.7769012451172

Real value -> 212.60000610351562

Absolute Error -> 6.8231048583984375
```

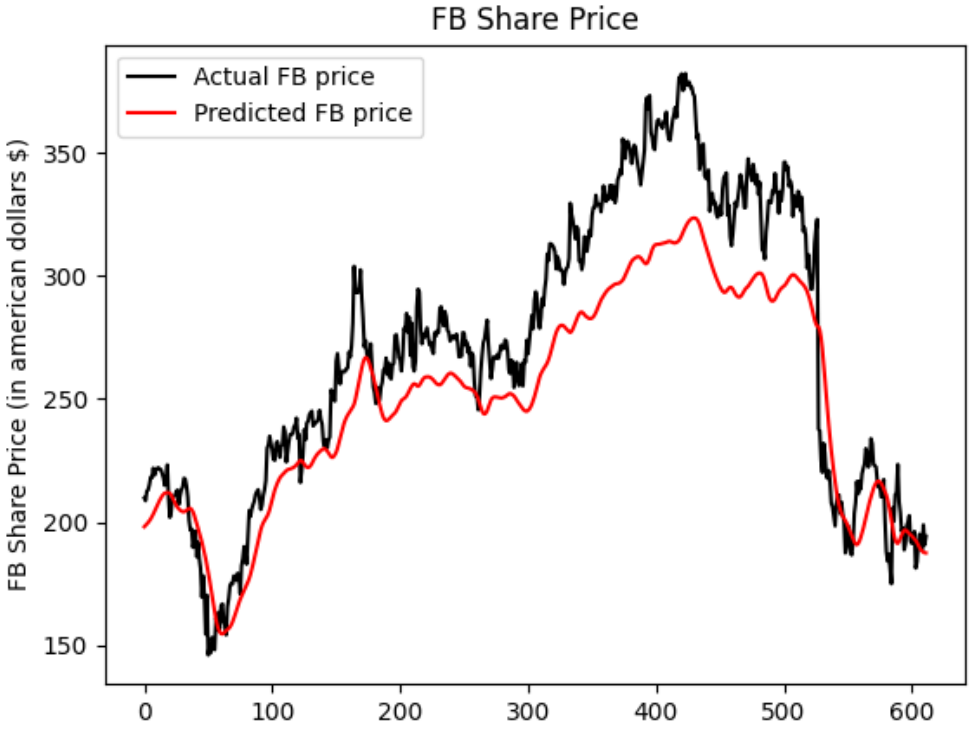
Tablica 3.3 Primjer ulaznih vrijednosti, predviđanja modela, prave cijene i pogreške (za timestep vrijednost smo uzeli 5, zbog toga ulaz ima 5 vrijednosti)

4. Rezultati izgrađenog modela

U rezultatima, generalno za skup učenja koristimo vrijednosti dionica određene tvrtke u razdoblju od 1.1.2012 do 1.1.2020, dok za skup za provjeru koristimo vrijednosti od 1.1.2020 do trenutnog datuma pokretanja programa. Ispisivati ćemo grafove predviđenih i pravih vrijednosti dionica za svaki dan u tom razdoblju.

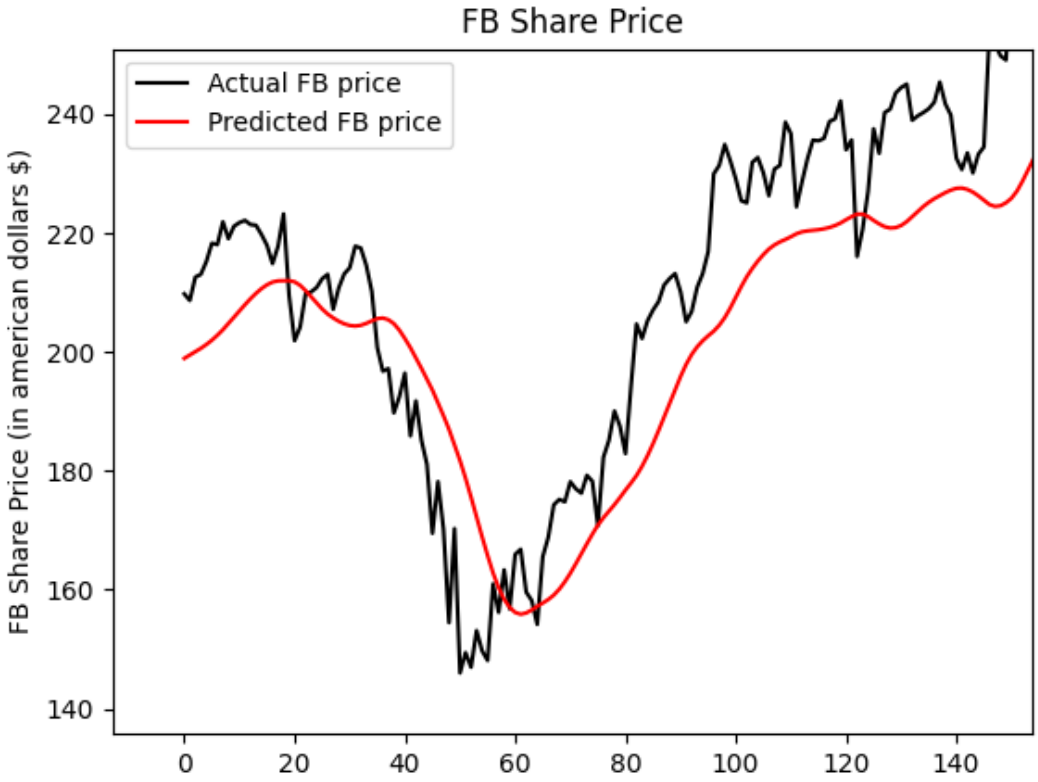
Sljedeći rezultati su temeljeni na tom da svaki dan posebno predviđamo preko definiranog broja prethodnih dana, gdje broj prethodnih dana u programskoj realizaciji označavamo *timestep* parametrom. Iako osim parametara možemo varirati i pojedine slojeve neuronske mreže da bismo dobili drugačije rezultate, u nastavku ćemo se bavit samo variranjem parametara izgrađenog modela čija je implementacija objašnjena u prethodnom poglavlju.

Parametre koje variramo u sljedećim rezultatima su *epoch* (određuje koliko puta ćemo provesti naš skup ulaznih podataka ponovo kroz model za vrijeme učenja), *timestep* (koliko dana unazad gledamo da bismo predvidjeli idući dan) i *batch_size* (predstavlja broj primjeraka nakon kojih će gradijent ažurirati mrežu).



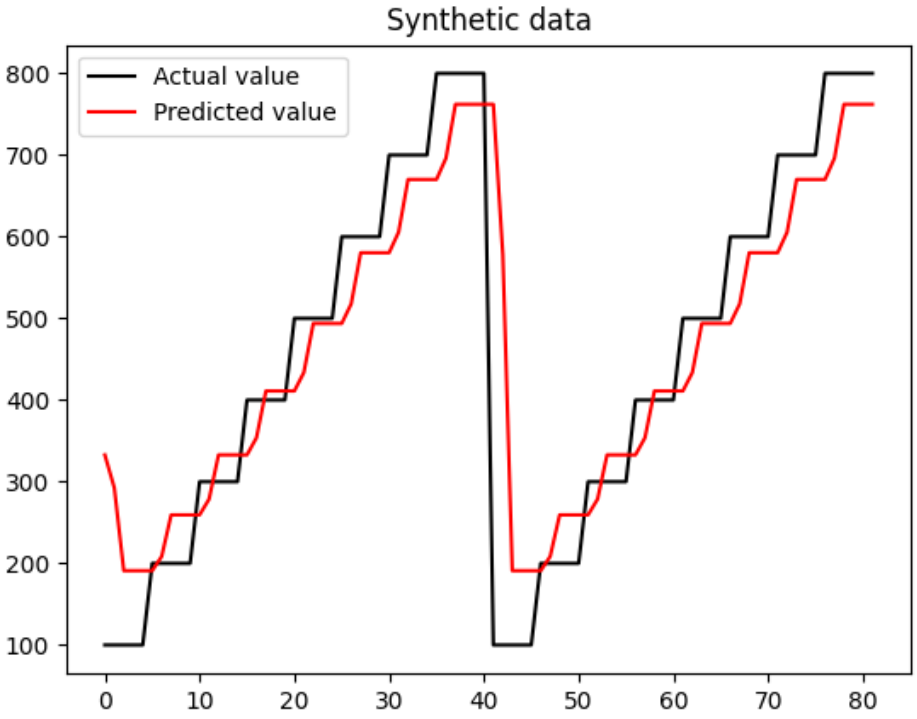
Slika 4.1 Rezultati predviđanja Meta(Facebook) dionica od 1.1.2020 do 1.6.2022 (epochs=25, timestep=60, batch_size=32)

Na slici 4.1 vidimo rezultate predviđanja za navedene parametre. Uviđamo da model ugrubo i prati krivulju stvarnih cijena.

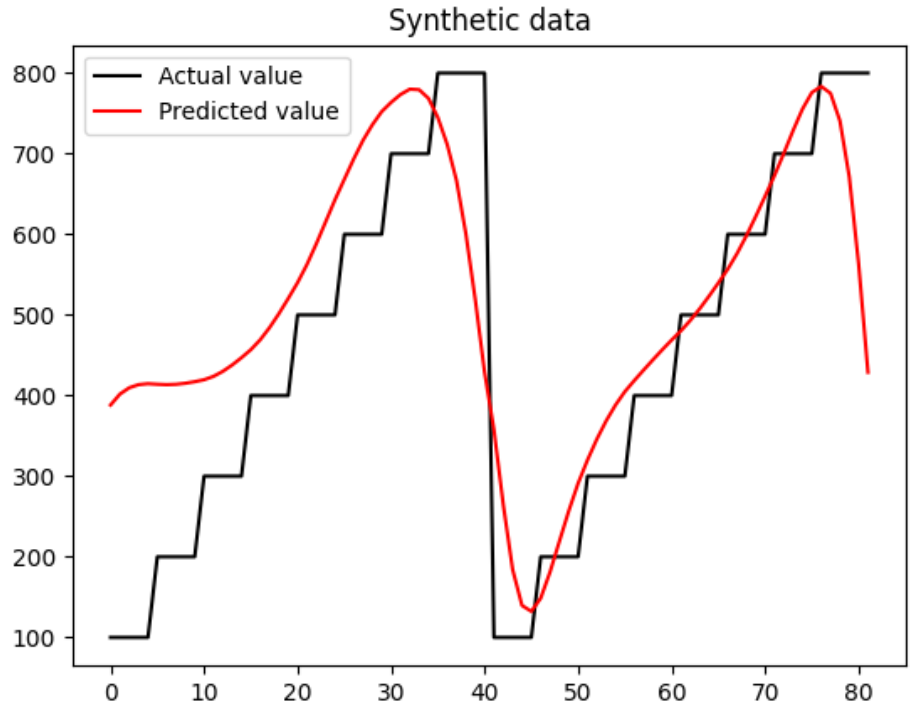


Slika 4.2 Približen početak 4.1 slike

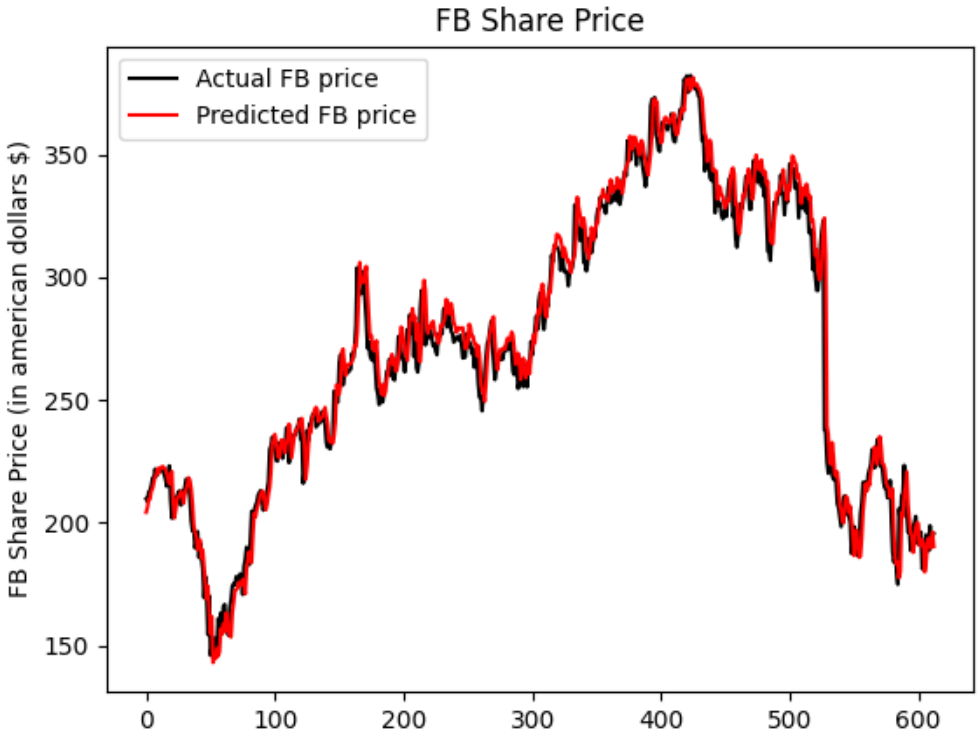
No približimo li početak grafa shvaćamo da model baš i nije optimalan, odnosno vidimo da kasni za pravim cijenama i više je „glađi“ od pravih cijena. Kašnjenje (engl. *lag*) grafa nastaje jer model iz prethodnih vrijednosti saznaje da su cijene narasle/pale pa zbog toga prilagođava trenutne cijene. Kašnjenje je očitije što manje dana gledamo unazad za predviđanje idućeg, a manje se izražava što više dana gledamo unazad. Dok su nam grafovi predviđanja „glađi“ što više dana gledamo unazad, a egzaktniji što manje dana gledamo unazad. Primjer očitog kašnjenja na sintetičkim podacima je na slici 4.3, a primjer „uglađenijeg“ grafa s manjim kašnjenjem je na slici 4.4.



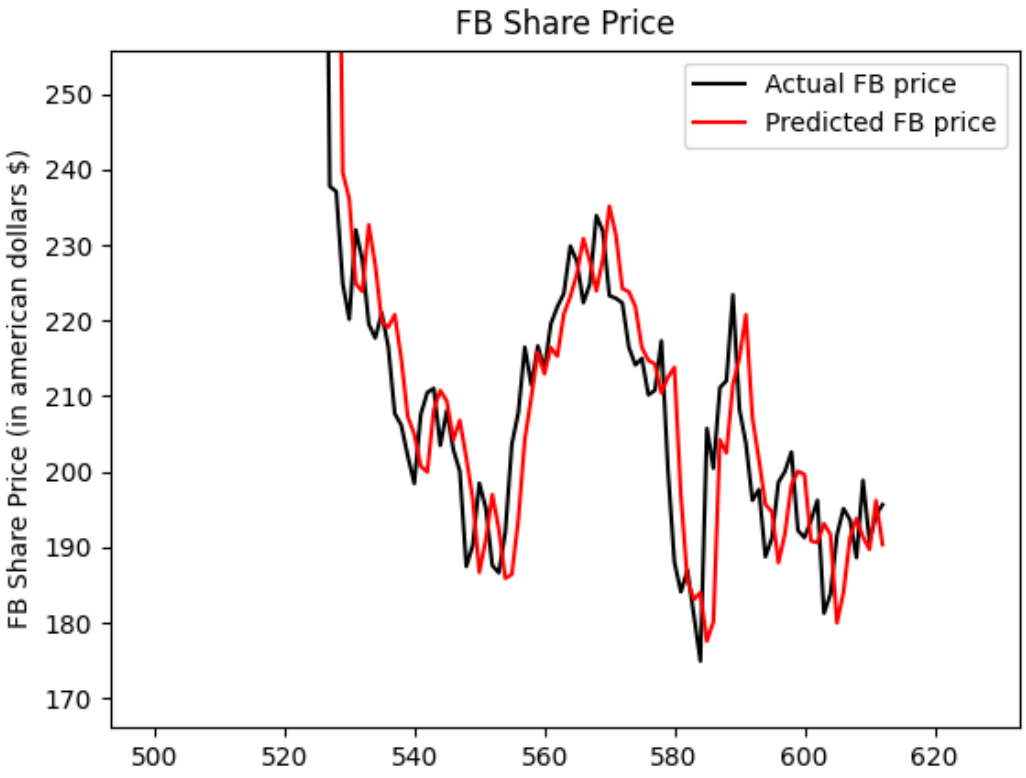
Slika 4.3 Sintetički podaci (epochs=25, timestep=2, batch_size=32)



Slika 4.4 Sintetički podaci (epochs=25, timestep=40, batch_size=32)



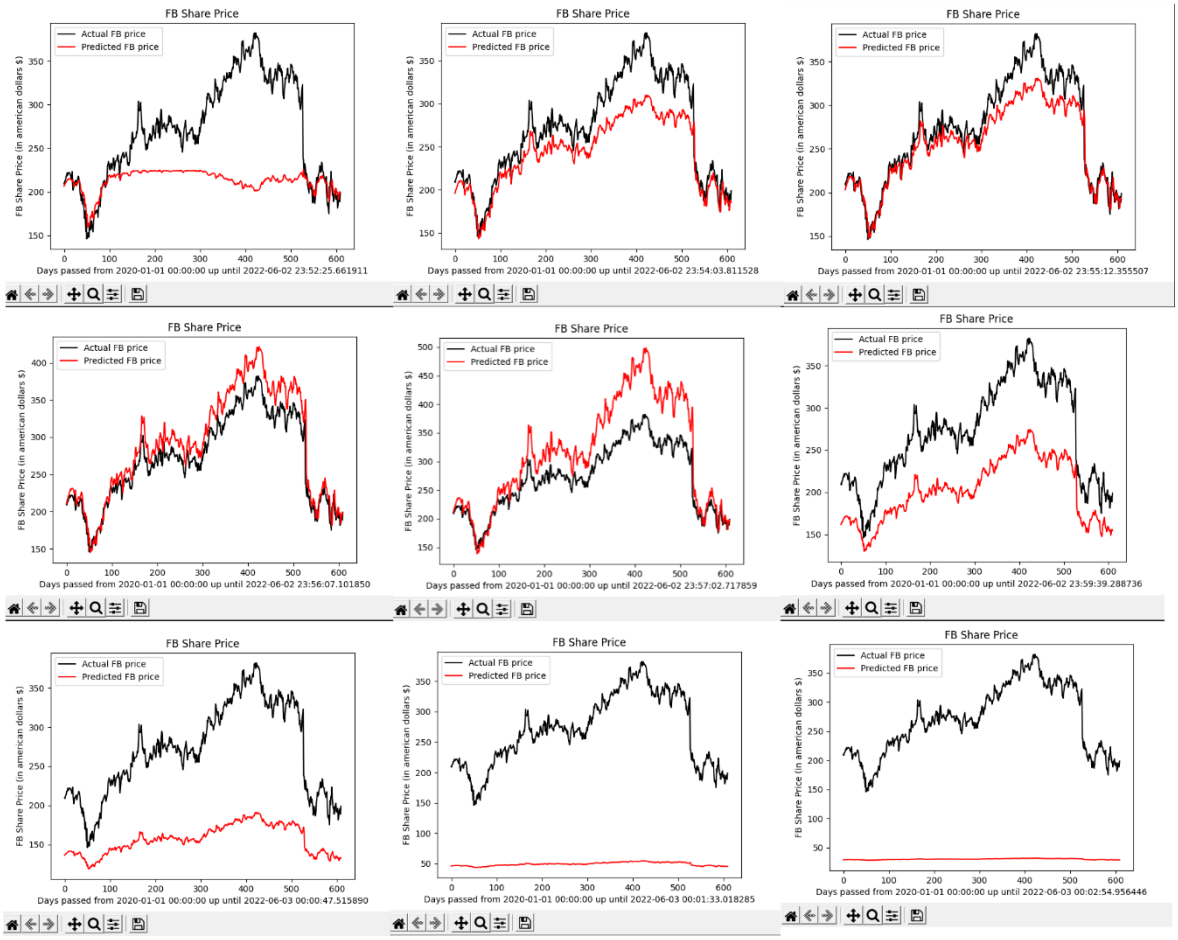
Slika 4.5 (epochs=5, timestep=2, batch_size=32)



Slika 4.6 Uvećani kraj 4.5 grafa

Slika 4.5 nam pokazuje predviđanje modela koje nas može zavarati, izgled grafa predviđanja „naoko“ prati prave cijene. No tu postoji problem, na približenom dijelu grafa na slici 4.6 je očitije da graf predviđanja kasni za grafom stvarnih cijena, što se događa jer koristimo „samo“ dva dana da predvidimo idući. To objašnjava zašto graf predviđanja toliko dobro prati graf stvarnih cijena, jer samo bazirano na prethodna dva dana odlučuje cijenu idućega dana, pa cijena predviđanja jednostavno neće varirati puno dalje od toga.

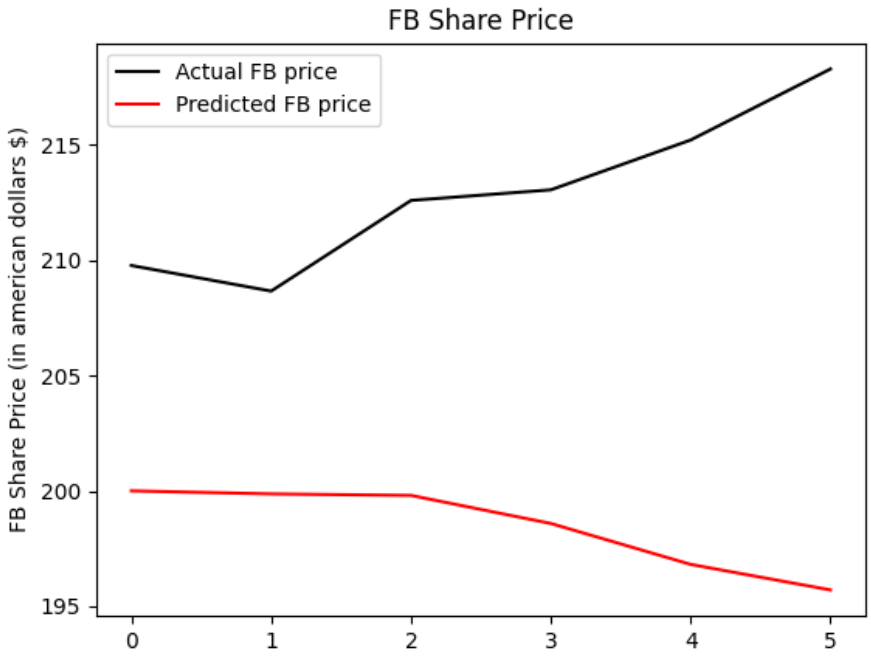
U nastavku fiksirajmo *epochs* i *timestep* parametre, a varirajmo parametar *batch_size*.



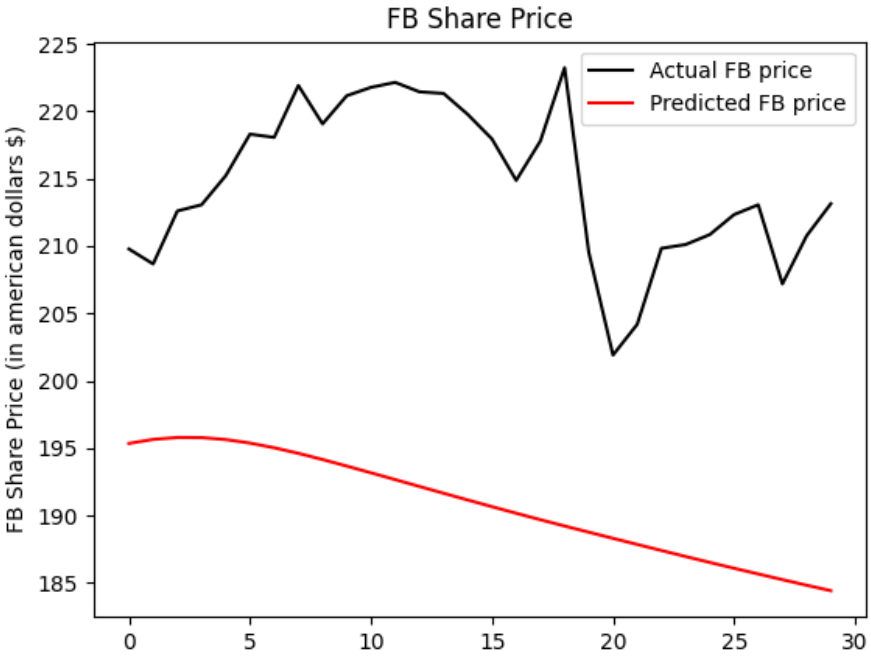
Slika 4.7 Grafovi s varirajućim batch_size parametrom

Slijeva nadesno vrijednosti parametra *batch_size* su iduće : 1, 16, 32, 64, 128, 256, 512, 1024, 2048. Vidimo da nam se model najbolje ponaša sa vrijednosti 32 pa nju općenito i uzimamo za vrijednost parametra *batch_size*.

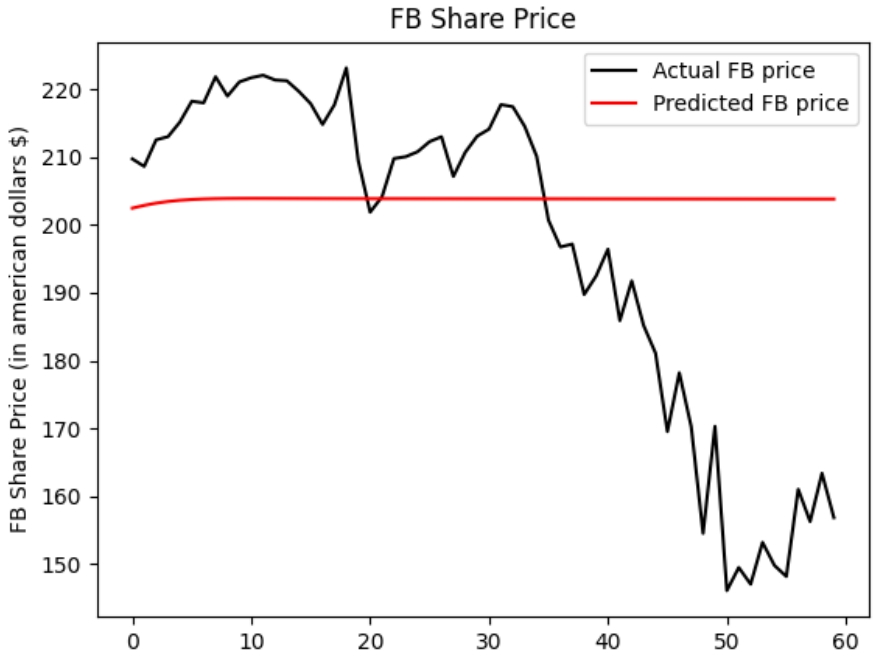
Dosadašnji rezultati su bili bazirani tako da uzmemo prozor dana preko kojih predviđamo svaki idući dan posebno. U nastavku su prikazani rezultati modela tako da model bazirano na svojim vrijednostima predviđanja predviđa nadolazeće cijene.



Slika 4.8 Rezultati modela kada model koristi svoje vrijednosti predviđanja da bi predvidio nadolazeće cijene (5 dana unaprijed)



Slika 4.9 Rezultati modela kada model koristi svoje vrijednosti predviđanja da bi predvidio nadolazeće cijene (30 dana unaprijed)



Slika 4.10 Rezultati modela kada model koristi svoje vrijednosti predviđanja da bi predvidio nadolazeće cijene (60 dana unaprijed)

Možemo uočiti da model koji koristi svoje vrijednosti u svrhu predviđanja nadolazećih, se ponaša poprilično loše, odnosno iz njega ne možemo ništa korisnoga iščitati. Graf predviđanja se nastavi kretati u onom smjeru u kojem je inicijalno i krenuo, bilo to prema povećanju, stagniranju ili smanjenu cijena.

Zaključak

Cilj ovoga rada je pokušaj predviđanja vrijednosti dionica koristeći određeni model neuronskih mreža. LSTM model smatramo kao izvrsnim u svrhu predviđanja vremenskog niza i traženja određenog uzorka iz skupa podataka. Inspirirano tom karakteristikom modela u radu je ispitano imaju li možda cijene dionica uzorak koji prate.

Iz rezultata implementiranog modela zaključujemo da dionice ne prate uzorak koji možemo iščitati iz čistih podataka o cijenama. Cijene dionica su uglavnom pod utjecajem vanjskih događaja koje neuronska mreža ne ubraja u svoja predviđanja. Neuronske mreže, konkretnije LSTM model se pokazao boljim jedino ako predviđamo kraće vremenske periode, zbog toga što je manji vremenski period to je manja šansa da dođe do vanjskog utjecaja.

Ako detaljno analiziramo cijene dionica uočit ćemo da oni generalno izgledaju kao nasumične vrijednosti, što je vrlo vjerojatno istina ako gledamo samo vrijednosti bez ostalih faktora i vanjskih utjecaja.

Literatura

[1] Čupić Marko, *Uvod u strojno učenje*, Zagreb, 2020

[2] Poveznica: https://www.educative.io/edpresso/overfitting-and-underfitting; pristupano 15.5.2022

[3] Kingma, D. P., Ba, J., *Adam: A method for stochastic optimization*., 2014

[4] Hardt, M., Recht, B., Singer, Y., *Train faster, generalize better: Stability of stochastic gradient descent*, *International Conference on Machine Learning*, 2016

[5] Poveznica: https://www.ibm.com/cloud/learn/recurrent-neural-networks; pristupano 20.5.2022

[6] Čupić Marko, *Umjetne neuronske mreže*, Zagreb, 2016

[7] Andrej Karptahy, poveznica: http://karpathy.github.io/2015/05/21/rnn-effectiveness/; pristupano 22.5.2022

[8] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio, *On the difficulty of training Recurrent Neural Networks*, 2012

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*, 2014

[10] Le, Xuan Hien & Ho, Hung & Lee, Giha & Jung, Sungho, *Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting*, 2019

[11] Poveznica: http://colah.github.io/posts/2015-08-Understanding-LSTMs/; pristupano 24.5.2022

[12] Poveznica: https://jaketae.github.io/study/dissecting-lstm/; pristupano 24.5.2022

[13] Poveznica: https://towardsdatascience.com/ai-machine-learning-deep-learning-explained-simply-7b553da5b960; pristupano 28.5.2022

[14] Poveznica: https://www.investopedia.com/terms/e/efficientmarkethypothesis.asp; pristupano 28.5.2022

[15] Poveznica: https://www.investopedia.com/articles/07/mean_reversion_martingale.asp; pristupano 28.5.2022

[16] Poveznica: https://towardsdatascience.com/is-it-possible-to-predict-stock-prices-with-a-neural-network-d750af3de50b; pristupano 9.6.2022

Sažetak

Predviđanje kretanja tržišta dionica neuronskim mrežama

U radu je detaljnije objašnjen način funkcioniranja neuronskih mreža i njenih bitnih dijelova. Predstavljene su i povratne neuronske mreže, posebice vrsta povratne neuronske mreže – LSTM. Prikazana je i implementacija LSTM modela u svrhu predviđanja cijena dionica, zbog njegove karakteristike dobrog predviđanja vremenskog slijeda. Rezultati modela predviđanja su uglavnom nezadovoljavajući zbog vanjskih utjecaja i nasumičnosti čistih podataka o cijenama.

Ključne riječi: strojno učenje, neuronske mreže, povratne neuronske mreže, LSTM, predviđanje cijena dionica

Summary

Predicting stock prices using neural networks

This thesis contains detailed explanation of neural networks and its important components. Recurrent neural networks are also elaborated in detail with emphasis on LSTM model. LSTM model is implemented for stock price prediction purposes because of its great time-series forecasting abilities. The results of the implemented model are mostly dissatisfying, because of external events and randomness of raw stock price data.

Key words: machine learning, neural networks, recurrent neural networks, LSTM, stock price prediction