Prediktering av kryptovalutor med återkommande neuronnät

Christian Karvonen

Examensarbete
Informationsteknik
2020

Förnamn Efternamn

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	7573
Författare:	Christian Karvonen
Arbetets namn:	Prediktering av kryptovalutor med återkommande neuronnät
Handledare (Arcada):	Jonny Karlsson
Uppdragsgivare:	

Sammandrag:

Inom de senaste åren har det utvecklats en massa tillämpningar, ramverk och verktyg för att lösa olika problem inom djupinlärning. Nya tekniker i maskininlärning och speciellt algoritmer inom djupinlärning introducerar kontinuerligt nya tillvägagångssätt för att nå goda resultat. Oförväntade förändringar i trender kan plötsligt orsaka stora variationer i tidsserierna och volatiliteten på kryptovalutor gör dem svåra att förutspå. Därför är syftet med examensarbetet att utnyttja effektiviteten av återkommande neuronnät för att förstå hur noggrannt en LSTM modell kan predicera en tidsserie med kryptovaluta som data. Som data används en datamängd (Quandl 2020) på Bitcoin, således förklaras det också hur en kryptovaluta hanteras som data i prognostisering. Teorigenomgången är uppbyggd av tidigare empirisk forskning kring prediktering av kryptovalutor med återkommande neuronnät, med fokus på långt korttidsminne (LSTM). För att uppfylla syftet utgår studien ifrån kvantitativa experiment för att välja den LSTM modell som tränar bäst och skapar de noggrannaste prediktionerna. Experimenten hjälper att välja parametrar, antalet LSTM lager och antalet epoker för den utvalda modellen. För att välja den noggrannaste modellen används en förlustfunktion, Mean Squared Error (MSE). MSE är en funktion som utvärderar graden av fel mellan träningsdelens beräknade output och förväntade output. Tills sist används visualiseringar för att slutligen analysera noggranheten av resultaten. Baserat på resultaten uppnår den utvalda LSTM modellen ett MSE värde på 0,001557, mycket nära 0,0 som är det bästa möjliga MSE värdet. Det betyder att resultaten och visualiseringarna visar att den utvalda modellen kan med en hög noggranhet skapa prediktioner på kryptovalutans tidsserie. En ökning av valideringsdelen kan hjälpa att ytterligare optimera modellen till en "perfect fit" modell. I framtiden skulle det därför vara intressant att vidareutveckla modellen för att bestämma den mest optimala storleken på valideringsdelen.

Nyckelord:	återkommande neuronnät, långt korttidsminne, prediktiv analytik, maskininlärning, djupinlärning
Sidantal:	29
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Information technology
Identification number:	7573
Author:	Christian Karvonen
Title:	Predicting cryptocurrencies with recurrent neural networks
Supervisor (Arcada):	Jonny Karlsson
Commissioned by:	

Abstract:

In the recent years a lot of applications, frameworks and tools have been developed to solve different problems in deep learning. New techniques in machine learning and especially algorithms in deep learning continuously introduce new approaches to achieve good results. Unexpected shifts in trends can lead to surprising fluctuations in time series and the volatility of cryptocurrencies make them hard to predict. Thus, the purpose of this thesis to use the effectiveness of recurrent neural networks to understand how accurately a LSTM model is able to predict on a time series with a cryptocurrency as a data. The data that is used is a dataset (Quandl 2020) on Bitcoin, thus it is also explained how a cryptocurrency is used as data in forecasting. The theory walkthrough is built around past empirical research on predicting cryptocurrencies with recurrent neural networks, with a focus on long short-term memory (LSTM). To fulfill the purpose, the study is based on quantitative experiments to choose the one LSTM model that trains the best and creates the most accurate predictions. The experiments also help to decide the different parameters, the amount of LSTM layers and the amount of epochs for the final model. To decide the most accurate model, a loss functions is used, Mean Squared Error (MSE). MSE is a function that evaluates the degree of error between the calculated outputs and the desired outputs. Finally, visualizations are used to analyze the accuracy of the results. Based on the results, the final model achieves an MSE value of 0.001557, very close to 0.0 which is the best possible MSE value. This means that the results and the visualizations show that the final model can create predictions with a high accuracy on the cryptocurrency time series. Increasing the size of the validation data could further help optimizing the model to a "perfect fit" model. Therefore, it could be interesting to further develop the model in the future applications to be able to choose the most optimal size of the validation data.

Keywords:	recurrent neural networks, long short-term memory, predictive analytics, machine learning, deep learning
Number of pages:	29
Language:	Swedish
Date of acceptance:	

INNEHÅLL

1 Inledning 6

1.1 Syfte 6
1.2 Metoder 7
1.3 Avgränsingar 7
1.4 Struktur 7

2 Teoretisk bakgrund 8

2.1 Djupinlärning 8
2.2 Kryptovaluta som data 8
2.3 Neuronnät 9
2.3.1 Kort om neuronnät 9
2.3.2 Aktiveringsfunktioner 10
2.3.3 Framåtmatande neuronnät 11
2.3.4 Återkommande neuronnät (RNN) 11
2.3.5 Långt korttidsminne (LSTM) 13
2.4 Träning av en LSTM modell 14
2.4.1 Förbehandling av data 14
2.4.2 Konfiguration av inlärningsprocessen 15
2.4.3 Definiera modellen 15
2.4.4 Träning av modellen 16

3 Data, verktyg och experiment 16

3.1 Datamängd 16
3.1.1 Fördelning av datamängden 17
3.2 Verktyg 18
3.3 Experiment 18
3.4 Experimentresultat 18

4 Implementering 19

4.1 Modellens arkitektur 19
4.2 Modellens träning 20
4.2.1 Den förbehandlade datamängden 20
4.2.2 Träning av modellen 21

5 Resultatredovisning 22

5.1 Resultat 22
5.2 Sammanfattning 25

6 Slutsatser 25

6.1 Diskussion 25
6.2 Framtida arbeten 26

Källor 27

FIGURER

Figur 1 OHLC + Volym tidsserie på BTC 9
Figur 2 Visualisering på ett simpelt neuronnät (Glosser.ca 2013.) 10
Figur 3 Illustration på när en aktiveringsfunktion appliceras på input. 10
Figur 4 Illustration på skillnaden i hur information flöder mellan ett återkommande neuronnät och ett framåtmatande neuronnät 11
Figur 5 Visualisering på RNN modulen. 12
Figur 6 Ett upprullat återkommande neuronnät. 12
Figur 7 Visualisering på LSTM modulen och dess olika portar. 13
Figur 8 Tabell på den slutliga datamängden, 3 första och 3 sista raderna. (15.3.2020) . 17
Figur 9 Tidsserie på datamängden fördelad i en träningsdel och en testdel.. 17
Figur 10 Visualisering av hur modellen ser ut som ett neuronnät. 20
Figur 11 Sammanfattning över hela träningen och modellens arkitektur 21
Figur 12 Visualisering av hur MSE förlustfunktionens värde förändrar sig under träningen. Den gråa linjen är för träningsdelen och den ljusblåa linjen är för valideringsdelen. 22
Figur 13 Tidsserie av det aktuella priset jämfört med det predikterade priset, ... 23
Figur 14 Tidsserie av det aktuella priset jämfört med det predikterade priset under de första månaderna i året 2020. 23
Figur 15 Visualisering på hela tidsserien. 24
Figur 16 Inzoomad tidsserie för de första månaderna under året 2020. 24

1 INLEDNING

Inom de senaste åren har det utvecklats en massa tillämpningar som använder sig av djupinlärning för att analysera data. Djupa neuronnät, såsom återkommande neuronnät (RNN) och långt korttidsminne (LSTM), har visat sig vara effektiva inom prediktiv analytik. I synnerhet har LSTM applicerats i många applikationer, såsom handskrivningsigenkänning (Graves & Schmidhuber 2009), taligenkänning (Graves et al. 2013a), naturlig språkbehandling (Tarwani & Edem 2017), prediktering av tidsserier (Gamboa 2017) och prediktering av aktiepriser (Roondiwala et al. 2017). Dessutom har en massa ramverk och verktyg utvecklats för att enklare kunna skapa LSTM modeller för att lösa en mängd olika uppgifter. Olika tekniker i maskininlärning och speciellt algoritmer inom djupinlärning har introducerat nya tillvägagångssätt för att nå prediktioner med goda resultat genom att använda djupa neuronnät med flera lager.

Prediktering av ekonomiska och finansiella tidsserier är en utmanande uppgift. Huvudsakligen på grund av oförväntade förändringar i ekonomiska trender som kan orsaka stora variationer i tidsserierna. Till exempel Covid-19 orsakade en plötslig nedåtgående trend på aktier och kryptovalutor. Volatiliteten på kryptovalutor gör dem nämligen svåra att förutspå.

I den här studien kommer fokuset inte att behandla hur oförutsägbara kryptovalutor kan vara, utan mera på att utnyttja effektiviteten av återkommande neuronnät för att nå ett prediktering resultat. Huvudsyftet med arbetet är således att med hjälp av LSTM kunna predicera kryptovalutans pris med en hög noggrannhet.

1.1 Syfte

Examensarbetets avsikt är att förklara vad neuronnät är, hur de fungerar och vilka som lämpar sig för prediktering. Specifikt hurdana djupa neuronnät som fungerar bäst för att predicera tidsserier, samt förklara hur man tränar neuronnät och diskutera hur kryptovalutor fungerar som en datamängd. Syftet med denna studie är att förstå hur effektivt och noggrannt en LSTM modell kan predicera en tidsserie med kryptovaluta som en datamängd.

Målet är att med hjälp av olika experiment kunna skapa en LSTM modell som kan skapa noggranna prediktioner på en kryptovalutas pris, samt analysera noggranheten av

studiens resultat med hjälp av visualiseringar för att ytterligare analysera hur modellen kan förbättras i framtida forskning.

1.2 Metoder

Hurdana neuronnät som idag fungerar bäst för att göra prognostiseringar på tidsserier baserar sig på litteratur, forskningar och artiklar om ämnet. Hur effektivt en LSTM modell prognostiserar en kryptovaluta, bestäms genom att jämföra det aktuella priset med det predikterade priset med hjälp av visualiseringar. Till sist, för att välja och komma fram till en LSTM modell med den bästa noggrannheten för implementeringen, körs ett antal experiment på flera liknande modeller. Hur bra noggranhet en modell har, mäts genom att evaluera modellen med en bedömingsmetrik; Mean Squared Error (MSE).

1.3 Avgränsingar

Ingen applikation eller bot implementeras för att automatisera implementeringen, utan allt körs manuellt i en Python notebook. Hur man prognostiserar på en tidsserie i framtiden, till exempel en dag eller en vecka framåt behandlas inte i detta arbete. Likaså hur man bestämmer när i tidsserien man borde köpa eller sälja sin kryptovaluta behandlas inte heller i detta arbete. Dessutom beskriver inte arbetet i detalj hur olika matematiska algoritmer fungerar i neuronnät eller hur verktygens inbyggda funktioner fungerar. Utöver detta går arbetet inte in på hur man kunde skapa egenskaper för att förbättra resultatet med extra kolumner, samt lämnas det bort extra kolumner från datamängden som kunde försämra resultatet.

1.4 Struktur

Resten av examensarbetet är strukturerat enligt följande; Det andra kapitlet ger en grundlig teoretisk bakgrund på vad djupinlärning är och hur olika neuronnät fungerar. Dessutom tas det upp hur kryptovalutor hanteras som data i prediktering samt hur man tränar en modell. Det tredje kapitlet presenterar datamängden och verktygen som används i examensarbetets implementering i mer detalj och beskriver experimenten och dess resultat. Det fjärde kapitlet ger en genomgång av implementeringen, presenterar den utvalda modellens arkitektur och hur träningen av modellen sker. Det femte kapitlet beskriver resultaten av implementeringen, speciellt med fokus på modellens noggrannhet

och på prediktionerna som modellen skapat. I det sjätte kapitlet ges en sammanfattande diskussion och slutsatser baserade på resultat och analys, samt förslag till fortsatt forskning.

2 TEORETISK BAKGRUND

Det här kapitlet introducerar grunderna i hur man hanterar kryptovaluta som data, hur olika neuronnät fungerar, främst återkommande neuronnät (RNN) och vad långt korttidsminne (LSTM) betyder, samt grunderna i hur man tränar en modell i återkommande neuronnät. Dessutom förklarar kapitlet hur LSTM skiljer sig från ett enkelt återkommande neuronnät och varför det fungerar bättre i prediktering av en kryptovaluta.

2.1 Djupinlärning

Inom datavetenskap är djupinlärning en typ av maskininlärning och dessa är undergrupper inom artificiell intelligens. Djupinlärning är en typ av artificiell intelligens som imiterar den mänskliga hjärnans funktionsätt. Djupinlärning och maskininlärning är båda lika på det viset att de använder sig av algoritmer och funktioner. Den främsta skillnaden är att djupinlärning använder sig av ett nät på flera lager av dessa algoritmer och funktioner. Var och en av dessa lager kan ge en annan sorts tolkning på data som matas i nätet. Ett sådant nät av algoritmer är ett artificiellt neuronnät som också kan kallas ett djupt neuronnät. (Rouse 2019a)

2.2 Kryptovaluta som data

För varje kryptovaluta, börjar beräkningen av datamängden från den dag kryptovalutan blev lanserad. Indexet på datamängden består av datum och kolumnerna består oftast av olika priskolumner samt en volymkolumn. Datamängden är alltså ordnad i tid och uppdaterad i intervaller på en dag. En sådan sekvens av data kallas också tiddsserie (se fig. 1) (Valkov 2019). Återkommande neuronnät, speciellt LSTM, är designade att identifiera mönster i sekventiell data, såsom tidsserier (Nicholson 2019).

Figur 1 OHLC + Volym tidsserie på BTC

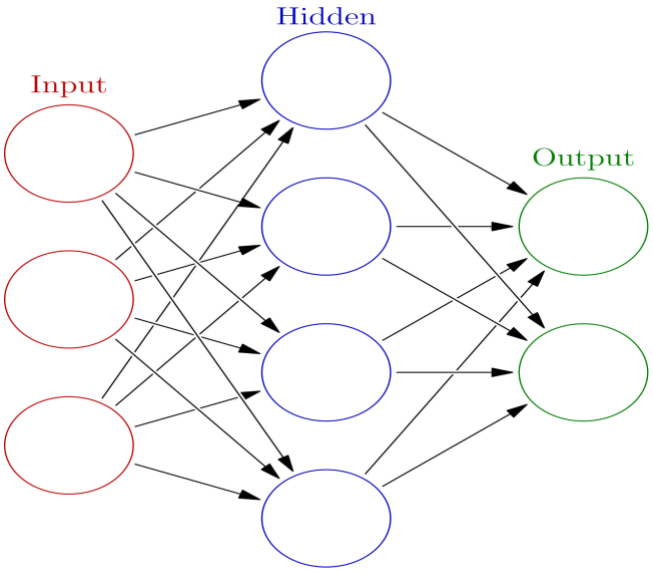
Enligt Sagar (2019) kan prediktering av kryptovalutor ändå vara väldigt restriktivt för att priset är beroende av många olika faktorer. I jämförelse med data på de flesta börserna, har de flesta kryptovalutor lite data. Den höga volatiliteten på kryptovalutor samt storleken på datamängden gör kryptovalutor relativt oförutsägbara (Sagar 2019).

2.3 Neuronnät

2.3.1 Kort om neuronnät

Inom djupinlärning strävar neuronnät till att fungera med samma princip som den mänskliga hjärnans neuronnät. Med andra ord är neuronnät en kedja av matematiska algoritmer, som försöker skapa observationer av en datamängd med hjälp av någon sorts maskin-inlärningsmetod. (Rouse 2019b). Ett simpelt neuronnät är uppbyggt av följande lager, ett input lager, ett dolt lager och ett output lager (se fig. 2). Denna uppsättning av lager består av neuroner. Neuronerna utför en serie av linjära och icke-linjära transformationer på inputen inom det dolda lagret för att generera en output. Därefter returnerar outputen en serie av prediktioner. Prediktionerna är alltså det som neuronerna generar med hjälp av de algoritmer och funktioner som finns i neuronerna. Till exempel vid träning av en kryptovaluta, lär sig neuronnätet på en input av en historisk tidsseries datamängd och skapar en output av prediktioner. I djupinlärning kan neuronnät bestå av flera dolda lager, varenda en med ett eget antal neuroner. (Torres 2018 s.54-55). Hårdvara och mjukvara möjliggör att neuronnät kan tränas genom att utnyttja datorers beräknings-förmåga. Beroende på problemet man vill lösa, använder man sig av olika sorters

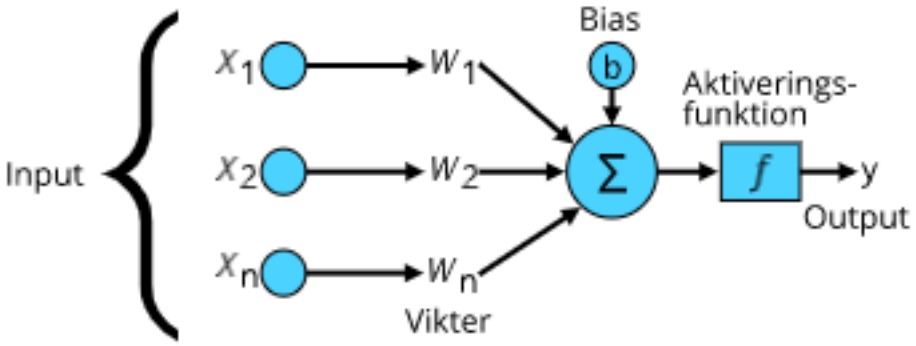
neuronnät. Vissa neuronnät är specifikt konstruerade för att användas för ett visst problem. (Rouse 2019b)



Figur 2 Visualisering på ett simpelt neuronnät (Glosser.ca 2013.)

2.3.2 Aktiveringsfunktioner

I ett neuronnät är en aktiveringsfunktion en del av en neuron som bestämmer neuronens output. Aktiveringsfunktioner hjälper neuronnät att bestämma om informationen som matas in är relevant eller irrelevant. Detta är viktigt för inlärningsprocessen av ett neuronnät, eftersom all information som matas in i ett neuronnät inte är lika användbart. Till exempel då information har matats in i en neuron, utför neuronen en linjär transformation på informationen med vikter och bias (se fig. 3).

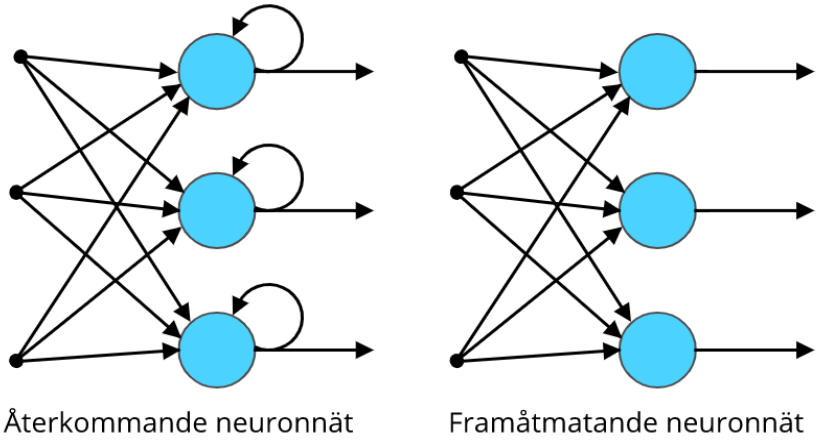


Figur 3 Illustration på när en aktiveringsfunktion appliceras på input.

Det finns flera olika aktiveringsfunktioner, det här arbetet fokuserar på tre icke-linjära aktiveringsfunktioner; tanh, sigmoid och ReLU. Tanh funktionen ger en vikt till input värdena och reglerar deras viktighet genom att transformera alla värden mellan -1 och 1. Sigmoid funktionen fungerar lika som tanh funktionen men transformerar istället alla värden mellan 0 och 1. Till skillnad från andra aktiveringsfunktioner aktiverar ReLU inte alla neuroner samtidigt. En typisk ReLU funktion aktiveras bara av de positiva input värdena och om input värdena är negativa förvandlas de till ett 0 värde. Inom djupinlärning har ReLU funktionen blivit väldigt populär. (Gupta 2020)

2.3.3 Framåtmatande neuronnät

För att ordentligt förstå hur ett återkommande neuronnät fungerar behövs det först kunskap om hur ett framåtmatande neuronnät fungerar. Återkommande neuronnät och framåtmatande neuronnät har fått sina namn från hur de processerar informationen som matas in i dem (Donges 2019). I ett framåtmatande neuronnät förs informationen i en riktning, från input lagret, genom det dolda lagret, till output lagret (se fig. 4). Framåtmatande neuronnät har inget minne av vilken sorts information som matats in och är därför dåliga på prediktering av data. (Donges 2019). Med andra ord har ett framåtmatande neuronnät ingen uppfattning av tid och ordning av data.



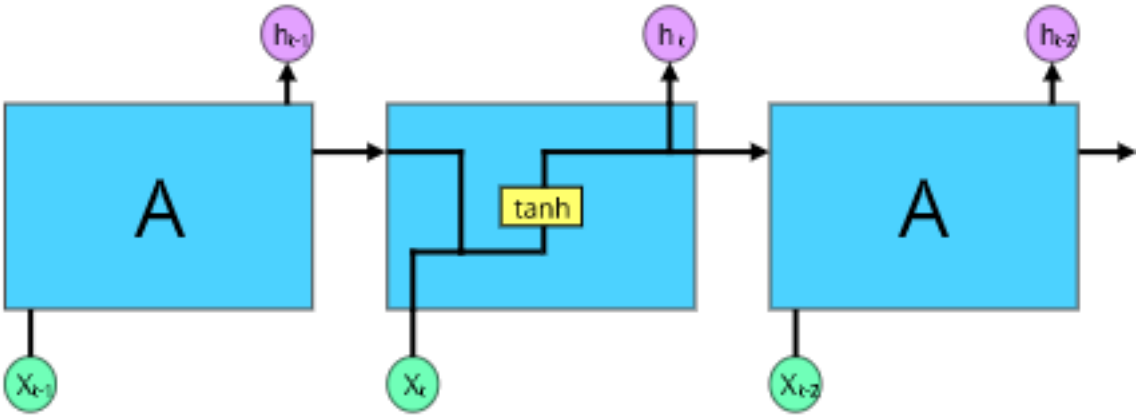
Figur 4 Illustration på skillnaden i hur information flöder mellan ett återkommande neuronnät och ett framåtmatande neuronnät

.

2.3.4 Återkommande neuronnät (RNN)

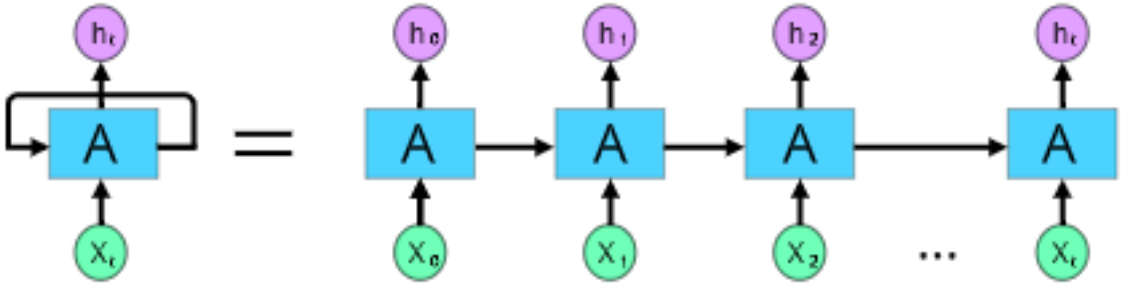
Ett återkommande neuronnät är en generalisering av ett traditionellt framåtmatande neuronnät med ett inre minne som kan processera sekvenser av inmatad data (Mittal

2019). På grund av dess interna minne kan återkommande neuronnät komma ihåg viktiga saker om tidigare data, vilket gör dem mycket exakta att förutspå vad som kommer att komma och att skapa prediktioner. Det är därför de är den föredragna algoritmen för sekvensiell data till exempel för tidsserier, tal, text, ekonomiska data, ljud, video, väder och mycket mer. Återkommande neuronnät kan bilda en mycket djupare förståelse av en sekvens och dess sammanhang jämfört med andra algoritmer. (Donges 2019). I ett återkommande neuronnät cirklar informationen via en loop som tillåter informationen att fortsätta till nästa neuron (se fig.5 och fig. 6). Vanligtvis består en typisk RNN modul av en tanh funktion. (se fig. 5). (Olah 2015)



Figur 5 Visualisering på RNN modulen.

Återkommande neuronnät utför samma funktion för varje input medan outputen av den nuvarande inputen beror på den före detta beräkningen. Efter att outputen har utarbetats, kopieras den och skickas tillbaka till det återkommande neuronnätet. Då det återkommande neuronnätet gör beslut, beaktar det den nuvarande inputen samt den outputen som den nyligen lärt sig av den förra outputen. (Mittal 2019). Arkitekturen av ett upprullat återkommande neuronnät (se fig. 6) avslöjer varför de är ett naturligt val för att processera sekvensiell data (Olah 2015).

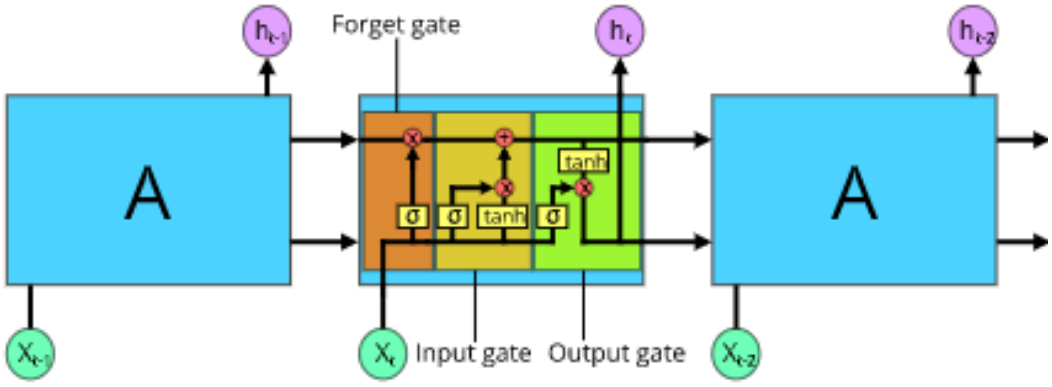


Figur 6 Ett upprullat återkommande neuronnät.

Problemet med ett vanligt återkommande neuronnät är att det endast har ett korttidsminne. Om en sekvens är tillräckligt lång, är det svårt att föra fram information från tidigare steg till senare steg. (Nguyen 2018). Detta innebär att återkommande neuronnät kan glömma och lämna bort viktig information från tidigare steg. Hochreiter (1991) och Bengio, et al. (1994) utforskade problemet mera djupgående, de kom fram med flera fundamentala anledningar på varför det kan vara svårt (Olah 2015). Gradientmetoderna, exploding gradient och vanishing gradient, var de främsta anledningarna som orsakade problem för återkommande neuronnät att komma ihåg långa sekvenser av information. Speciellt "vanishing gradient" förorsakar att återkommande neuronnät slutar att är lära sig information på långa sekvenser. Därför är långt korttidsminne (LSTM) lösningen på problemet. (Donges 2019)

2.3.5 Långt korttidsminne (LSTM)

Långt korttidsminne (LSTM), ursprungligen introducerat av Hochreiter och Schmidhuber (1997), är ett modifierat återkommande neuronnät (Mittal 2019). En typisk LSTM modul består av tre portar (gates), "input gate", "forget gate" och "output gate" (se fig. 7). I dessa portar finns det aktiveringsfunktioner, sigmoid och tanh funktioner, på flera olika ställen. De olika funktionerna bestämmer om informationen lagras, tas bort eller förs framåt till nästa steg (Olah 2015). LSTM är specifikt designade för att lösa problemet med korttidsminnet som uppstår med ett vanligt återkommande neuronnät, det vill säga att komma ihåg information för en längre tid är vad LSTM är bra på och därför uppstår inga problem med inlärningen (Olah 2015). Detta betyder att LSTM är väl lämpade på att klassificiera, processera samt predicera tidsserier (Mittal 2019).



Figur 7 Visualisering på LSTM modulen och dess olika portar.

Jämfört med en RNN modul är processen för en LSTM modul mer komplex och innefattar fler steg. Det första steget i en LSTM modul är att sortera all information beroende på vad som är relevant och vad som ska tas bort. Detta beslut fattas av en sigmoid funktion inom "forget gate" och sparas i en så kallad "cell state", "x" i visualiseringen (se fig. 7). Andra steget är att besluta vilken information som ska sparas. En sigmoid funktion besluter vad som ska uppdateras, parallellt skapar tanh funktionen möjliga värden som kunde adderas till "cell state". Detta händer inom "input gate" som bestämmer vad som ska uppdateras genom att kombinera sigmoid och tanh funktionen i en annan "cell state". I det tredje steget multipliceras båda föregående "cell states" med varandra för att skapa en ny gemensam "cell state", exklusive värdena som valdes bort i "forget gate". Det sista steget bestämmer genom en sigmoid funktion, vilka delar av "cell state" som slutligen ska matas ut. Sedan förs "cell state" ännu via en tanh funktion och multipliceras sedan med outputen från sigmoid funktionen för att nå en gemensam output. Den slutliga outputen av en LSTM neuron blir en så kallad "hidden state", "h" i visualiseringen (se fig. 7). (Olah 2015)

2.4 Träning av en LSTM modell

Det ingår flera steg för träning av en LSTM modell. I det här avsnittet presenteras en metod som kan användas för att konfigurera inlärningsprocessen av en LSTM modell.

2.4.1 Förbehandling av data

För att konfigurera samt utvärdera en modell inom djupinlärning, delas den tillgängliga datamängden upp i tre delar; en träningsdel, en valideringsdel och en testdel (Torres 2018 s.87). Alltså delas datamängden upp i procentuella mängder av den totala datamängden till exempel på följande sätt; träning 80%, validering 10% och test 10%. Träningsdelen används för att anpassa en modell (Shah 2017). Det vill säga tränar modellen med parametrar, såsom vikter och biaser på en stor del av datamängden (Torres 2018 s.53). Modellen ser och lär sig av den här datamängden (Shah 2017). Valideringsdelen används för att ge en utvärdering på den tränade modellen samtidigt som hyperparametrar finjusteras (Shah 2017). Slutligen används testdelen för testning och utvärdering av den tränade modellen (Shah 2017).

Inom förbehandling av data används det också en teknik som kallas normalisering. Vilket enligt Torres (2018 s.90) innebär att man skalar alla input värden inom en viss variationsvidd. För en LSTM modell normaliseras värden oftast mellan 0 och 1 (Srivastava 2017), vilket också kallas min-max. Det är vanligt att normalisera alla värden innan man tränar en LSTM modell. Ett LSTM förväntar också att all input data förbehandlas till ett format på 3 dimensioner, [sats, tidssteg, egenskaper]. Sats är den totala mängden datapunkter, tidssteg är antalet steg för en datapunkt och egenskaper hänvisar till antalet variabler som korresponderar det riktiga värdet. (Srivastava 2017)

2.4.2 Konfiguration av inlärningsprocessen

Inom djupinlärning kan man konfigurera inlärningsprocessen av en modell. Förlustfunktioner, optimerare och metriker är parametrar som kan användas för att kompilera en modell. En förlustfunktion används för att utvärdera graden av fel mellan träningsdelens beräkande output och förväntade output (Torres 2018 s.98). För tidsserier, alltså en regressionsmodell, är mean square error (MSE) och mean absloute error (MAE) de vanligaste förlustfunktionerna (Parmar 2018). En optimerare specifierar optimeringen för algoritmen som beräknar vikten på parametrarna av träningsdelens data (Brownlee 2017). Inom djupinlärning kallas en sådan algoritm för gradient descent. En klassisk gradient descent, stochastic gradient (SGD) är väldigt långsam. Adam (Kingma & Ba 2014) är en optimerings algoritm som är mycket snabbare på att beräkna än SGD och kan användas istället för SGD (Brownlee 2017). Adam har därför blivit väldigt populär för optimering inom djupinlärning för att den uppnår snabbt bra resultat (Brownlee 2017). Som en metrik parameter används det oftast av en noggrannhets metrik för att utvärdera modellens noggrannhet på träningsdelen (Torres 2018 s.98).

2.4.3 Definiera modellen

Att definiera en LSTM modell beror mycket på problemet, då LSTM kan bestå av flera dolda lager. Inom djupinlärning används en sekventiell modell som kan bestå av en linjär hög av lager. Den sekventiella modellen tar in lager av LSTM, dropout och dense samt aktiveringsfunktioner. I en modell av flera LSTM lager appliceras det oftast ett dropout lager efter varje LSTM lager för att förhindra överanpassning av modellen. Dropout förhindrar överanpassning genom att slumpmässigt lämna bort neuroner. Dense igen är ett lager som ser till att alla neuroner i det föregående lagret ansluter sig till alla neuroner

i det nästa lagret. I andra ord ser dense lagret till att hela neuronnätet ansluts ihop och spottar ut ett output med prediktioner. Olika aktiveringsfunktioner kan kopplas till LSTM och dense lagren för att förändra deras standard aktiveringsfunktion. (Srivastava 2017)

2.4.4 Träning av modellen

Efter att man förbehandlat datamängden, konfiguerat inlärningsprocessen och definierat en modell, samt valt alla parametrar kan man anpassa och träna sin modell. Vid anpassning av en modell i LSTM, kan man ännu bestämma storleken av hyperparametrar, satsstorlek och epoker. Satsstorleken fördelar träningsdelens data till mindre satser enligt argumentets storlek. Under en anpassning av en modell, uppdateras modellens parametrar enligt satsstorleken. Den optimala satsstorleken beror på många faktorer, inklusive minneskapaciteten och datorn som används för beräkningarna. Epoker gör det möjligt att slunga hela träningsdelen igenom en modell flera gånger. Hur många gånger som en träningsdel ska slungas igenom en modell bestäms av antalet epoker. (Torres 2018 s.100)

3 DATA, VERKTYG OCH EXPERIMENT

3.1 Datamängd

I det här arbetet används en datamängd (Quandl 2020) på Bitcoin från 2014 till dagens datum. All data hämtas direkt från "Quandl.com" och därför uppdateras också datamängden dagligen. I denna implementering är sista datumet för hämtning av datamängd den 15.3.2020. Indexet i datamängden är "Date" som innehåller alla datum i "år-månad-dag" format. Datamängden består av totalt 7 kolumner; "Open", "High", "Low", "Close", Volume (BTC), "Volume (Currency)" och "Weighted Price". I den slutliga implementeringen används bara de fyra första kolumnerna (se fig. 8). "Open" är det första priset för BTC för dagen, "High" är det högsta priset för BTC för dagen, "Low" är det lägsta priset för BTC för dagen och "Close" är det sista priset för dagen (Sagar 2019). De tre sista kolumnerna är inte nödvändiga för predikteringen av priset på

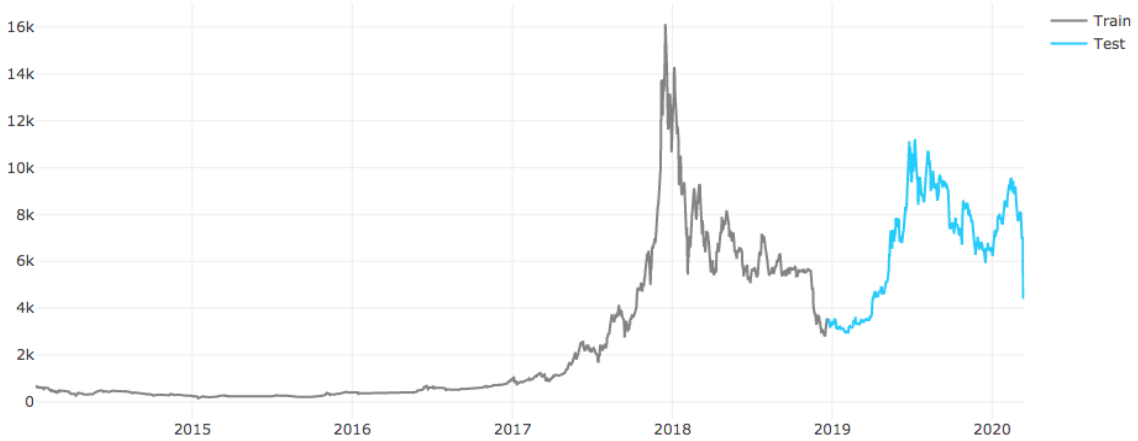
kryptovalutan och kan till och med försämra resultatet. Detta bevisas senare i experimenten.

Date	Open	High	Low	Close
2014-01-08	624.01000	640.0000	600.0	613.47231
2014-01-09	613.47231	626.9999	590.0	614.22456
2014-01-10	615.00000	630.0000	605.0	625.73866
2020-03-13	4392.70000	5431.9000	3550.0	5113.40000
2020-03-14	5106.50000	5136.1000	4575.1	4683.90000
2020-03-15	4679.20000	5389.7000	4617.5	4817.70000

Figur 8 Tabell på den slutliga datamängden, 3 första och 3 sista raderna. (15.3.2020)

3.1.1 Fördelning av datamängden

Inom maskininlärning kallas label det som man försöker predicera i en modell (Torres 2018 s.51). Målet i implementeringen är att predicera priset på "Close" kolumnen, alltså är denna kolumn label eller "Y" (Torres 2018 s.52). Resten av alla kolumner inklusive "Close" är egenskaper eller "X" (Torres 2018 s.52). Dessa egenskaper är input för träning av en modell och hjälper modellen att hitta samband mellan "X" och "Y". Datamängden fördelas således i två delar, en träningsdel och en testdel (se fig. 9). Träningsdelen och testdelen förbehandlas också till följande grupper; X_train, y_train, X_test och y_test. Där X_train och y_train används för att träna upp modellen och där X_test och y_test används för att skapa prediktioner, samt för att evaluera den upptränade modellen. (Torres



Figur 9 Tidsserie på datamängden fördelad i en träningsdel och en testdel..

3.2 Verktyg

Arbetet i det här examensarbetet är implementerat[1] i Python. Det huvudsakliga Python biblioteket som används i arbetet är Keras (Chollet 2015), vilket är ett ramverk som tillåter användare att skapa och träna effektiva modeller inom djupinlärning, till exempel LSTM modeller. De andra huvudsakliga biblioteken är Tensorflow (Abadi et al. 2015), Pandas (McKinney 2010), NumPy (Oliphant 2006).

Python valdes för att Python redan har en massa olika bibliotek som fungerar bra för maskininlärnings syften. Till exempel används Pandas för att läsa in och hantera datamängder och NumPy för att hantera arrayer och beräkningar. Keras ramverket valdes för att det är byggt i Python. Keras är också ett populärt alternativ för att enkelt skapa djupa modeller med få kod rader, jämfört med till exempel SciPy ramverket. Dessutom har Keras en inbyggd funktion för att skapa LSTM lager, vilket gör det enkelt att skapa LSTM modeller med flera LSTM lager. Dock kräver Keras också ett backend ramverk för att kunna träna modeller. Här kommer Tensorflow in då det är ett backend ramverk och har till och med blivit ett default backend ramverk för Keras.

[1] Koden är tillgänglig på https://github.com/KarvonenC/LSTM

3.3 Experiment

För att nå den slutliga implementeringen, inklusive modellens slutliga arkitektur, antal epoker, antal neuroner, satsstorleken, vilka kolumner och hyperparametrar som används, kördes ett antal experiment. Dessa experiment involverade främst dock att nå ett beslut på hur många lager av LSTM som modellen behöver för att nå den bästa möjliga prediktionen. För att förenkla beslutet av antal LSTM lager skapades en pipeline av modeller. Denna pipeline hjälper till att träna och evaluera modellerna på en och samma körning. Den utvalda modellen för implementeringen beskrivs mer i detalj i Kapitel 4 och resultaten av experimenten redovisas i Avsnitt 3.4.

3.4 Experimentresultat

Experimenten hjälpte att bestämma den slutliga modellen i sin helhet. Antalet epoker för den utvalda modellen bestämdes genom att observera när MSE värdet slutade bli lägre under en körning. Med andra ord då modellen slutade att prestera bättre på träningsdelen.

För denna datamängd var det alltid runt 50 epoker tills modellen inte längre presterade bättre. Därför blev det 50 epoker i den slutliga modellen.

Antalet neuroner för varje LSTM lager för den slutliga modellen bestämdes genom trial and error metoden. Den optimala mängden neuroner blev 32 neuroner i varje LSTM lager. En lägre mängd neuroner försämrade MSE värdet. En högre mängd neuroner visade sig att bara ta en längre tid men förbättrade inte resultatet på något vis. Satsstorleken bestämdes också genom trial and error metoden. Sattstorleken blev 100 för att försnabba träningen av modellen. Det visade sig inte vara någon större skillnad på träningen om satsstorleken var mindre än 100, men om den var större blev resultatet negativt.

I den utvalda modellen används en aktiveringsfunktion ReLU i det första LSTM lagret. Där visade det sig att ReLU fungerar bäst och de facto bara försämrade resultatet om det användes i flera LSTM lager.

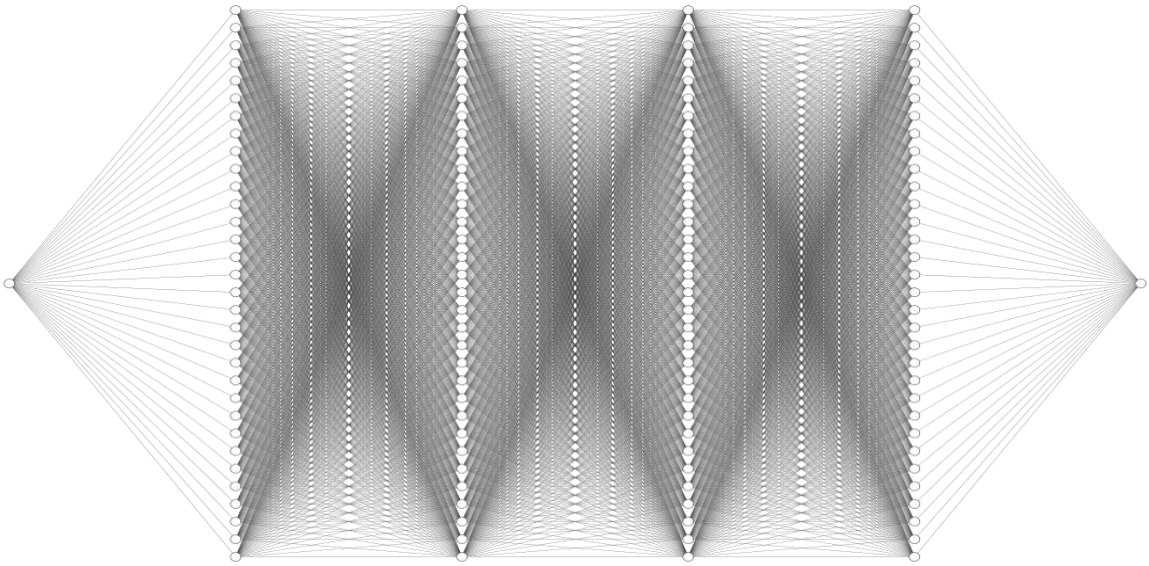
Antalet LSTM lager bestämdes genom trial and error, ett antal körningar kördes på en pipeline av fem modeller. Dessa modeller var likadana men bara antalet LSTM lager skilde dem från varandra, 1-5 lager respektive modell. Vilken modell som presterade bäst bestämdes genom att evaluera modellerna med hjälp av error värdet av MSE förlustfunktionen under ett antal körningar. Att ha flera lager spelade inte en så stor roll på resultatet, skillnaden på MSE värdena var väldigt minimala. Modellen med 4 LSTM lager fick bästa MSE resultat i 4 av 10 körningar. Således resulterade det till att den slutliga och utvalda modellen blev en modell med 4 LSTM lager. Körningarna av experimenten visade sig också att de tre sista kolumnerna; "Volume (BTC)", "Volume (Currency)" och "Weighted Price" i datamängden bara försämrade resultatet. Genom att lämna bort dessa kolumner blev MSE värdet lägre och modellernas inlärning blev en aning nogrannare.

4 IMPLEMENTERING

4.1 Modellens arkitektur

Den utvalda modellens arkitektur är inspirerad av (Graves 2013b), främst hur man kan stapla flera LSTM lager på varann. Dock är den slutliga arkitekturen uppnådd via experimenten.

Den utvalda modellen är en sekventiell modell som består av nio block. Åtta av dessa block består av fyra LSTM lager och fyra Dropout lager, varje LSTM block är följt med ett Dropout block. Alla fyra LSTM lager består av 32 neuroner och alla fyra dropout lagrer har en dropout hastighet på 0.2. Det första LSTM blocket tar också emot inputet och en ReLU aktiveringsfunktion är också tillsatt på det första LSTM lagret. Det sista blocket består av ett Dense lager som ansluter ihop neuronnätet och spottar ut ett output. Sammafattningsvis är modellens arkitektur ett djupt neuronnät (se fig. 10 och fig. 11) från vänster till höger; input, fyra dolda lager med 32 neuroner och output.



Figur 10 Visualisering av hur modellen ser ut som ett neuronnät.

4.2 Modellens träning

4.2.1 Den förbehandlade datamängden

Datamängden är uppdelad i en träningsdel och en testdel, respektive 80 % och 20 % av datamängden. Dessa delar är normaliserade och förbehandlade till ett 3D input format som LSTM modellen kan förstå. Dessutom för att urskilja X och y i både träning och testing är följande tränings- och testgrupper skapade; X_train, y_train, X_test och y_test. Vid anpassningen av modellen används en funktion "validation_split" med ett värde på

0,1. Funktionen väljer en valideringsdel på 10 % från slutet av träningsdelen, X_train. Vilket betyder att modellen tränar på 70 % och validerar på 10 % av datamängden.

4.2.2 Träning av modellen

Modellens arkitektur är kompilerad med en funktion "model.compile()" och en förlustfunktion, MSE, och en optimerings funktion, Adam. MSE räknar medelvärdet av standardavvikelserna mellan X_train och y_train. Under träningen uppdateras MSE värdet vid varje epok, ju lägre desto bättre. Adam funktionen försnabbar inlärningen genom att uppdatera vikterna på träningsdelens parametrar för att snabbare nå ett mindre MSE värde. Modellen anpassar sig på träningsdelen, X_train och y_train, med funktionen "model.fit()" och dess hyperparametrar. Modellen tränar i 50 epoker, det vill säga, att hela träningsdelen slingas 50 gånger igenom modellens arkitektur. Satsstorleken i modellen är 100, alltså fördelas träningsdelen i satser på 100 datapunkter. Antalet datapunkter eller längden på träningsdelen är 1622, vilket betyder att modellen går igenom 16 mindre satser vid varje epok. Utöver detta används en funktion "shuffle=True" som endast blandar träningsdelen vid varje epok. Det förhindrar modellen att lära sig ordningen på träningsdelen, vilket i sin tur förhindrar att det uppstår bias i träningen. Det hjälper också modellen att konvergera snabbare för att nå ett bättre resultat med en mindre mängd epoker.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 5, 32)	4736
dropout_1 (Dropout)	(None, 5, 32)	0
lstm_2 (LSTM)	(None, 5, 32)	8320
dropout_2 (Dropout)	(None, 5, 32)	0
lstm_3 (LSTM)	(None, 5, 32)	8320
dropout_3 (Dropout)	(None, 5, 32)	0
lstm_4 (LSTM)	(None, 32)	8320
dropout_4 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 1)	33

Total params: 29,729
Trainable params: 29,729
Non-trainable params: 0

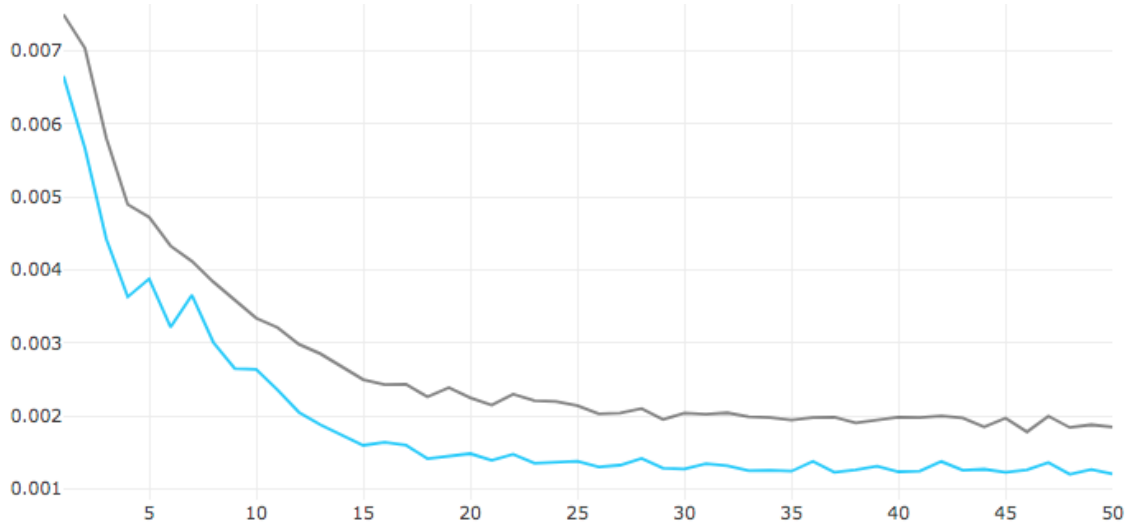
Figur 11 Sammanfattning över hela träningen och modellens arkitektur

5 RESULTATREDOVISNING

Efter att modellen blivit tränad med träningsdelen, X_train och y_train, kan modellen slutligen evalueras och prediktioner kan skapas. Modellen evalueras genom att anpassa "model.evaluate()" funktionen med den oberörda testdelen, X_test och y_test. Funktionen returnerar ett resultat på MSE värdet, om värdet är likt MSE värdet i träningen betyder det att modellen presterar bra. Modellen skapar prediktioner genom att anpassa "model.predict()" funktionen med en oberörd testdel, X_test. Funktionen genererar ett output av prediktioner på basen av de riktiga värderna i inputet X_test.

5.1 Resultat

Resultaten är uppdelade i två huvudsakliga delar, evaluering av modellen och de skapade prediktionerna. Evalueringen delas också upp i två delar, evaluering av förlustfunktionen på hela modellen och evaluering av förlustfunktionen på träningsdelen jämfört med valideringsdelen under träningen. Hur förlustfunktionen MSE förändrade sig under träningens 50 epoker på de båda delarna är visualiserat i ett linjediagram (se fig. 12), träningsdelen är den gråa linjen och valideringsdelen är den ljusblåa linjen.



Figur 12 Visualisering av hur MSE förlustfunktionens värde förändrar sig under träningen. Den gråa linjen är för träningsdelen och den ljusblåa linjen är för valideringsdelen.

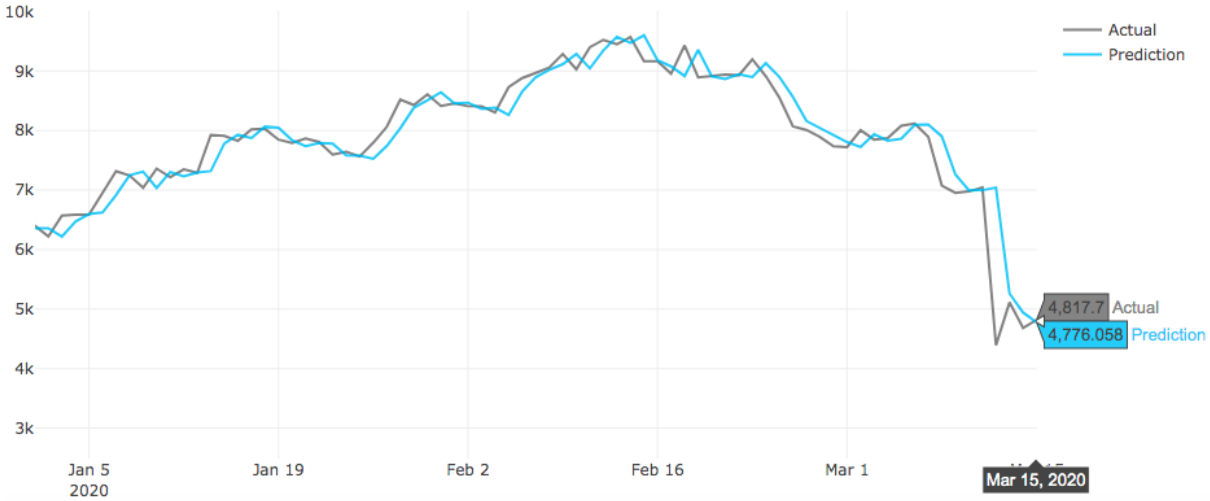
Linjediagrammet visar tydligt att båda delarna börjar med ett högre MSE värde men sjunker nästan till modellens mest optimala MSE värde redan efter 20 epoker. MSE värdet förändrar sig hemskt lite mellan epokerna 20 och 50 men minskar sig ändå kontinuerligt. Funktionen "model.evaluate()" spottade ut ett resultat på MSE värdet för hela modellen,

vilket under experimenten rörde sig mellan 0,00155 och 0,00165. Den sista körningen på den utvalda modellen fick med 6 decimalers noggrannhet, ett resultat på 0,001557.

Efter att den utvalda modellen var färdigt tränad skapades prediktionerna på kryptovalutans "Close" pris med funktionen "model.predict()" på testdelen X_test. För att lättast visa resultaten på prediktionerna, skapades några visualiseringar där det aktuella priset jämförs med det predikterade priset (se fig. 13-16). I visualiseringen på hela prediktionsdelen (se fig. 13) finns knappt någon skillnad mellan det aktuella priset och det predikterade priset.

Figur 13 Tidsserie av det aktuella priset jämfört med det predikterade priset,

Den mer inzoomade visualiseringen (se fig. 14) visar bättre hur nära kryptovalutans predikterade pris är jämfört med kryptovalutans aktuella pris under de första månaderna i året 2020. Det sista värdet för båda priserna visas också i samma visualisering (se fig. 14), vilket bevisar hur lite prediktionerna skiljer sig från det aktuella priset.



Figur 14 Tidsserie av det aktuella priset jämfört med det predikterade priset under de första månaderna i året 2020.

I nästa visualisering (se fig. 15) på hela tidsserien, är det tydligt att prediktionerna på testdelen är nära vid sidan om i hela tidsserien. Prediktionerna följer väldigt nära till kryptovalutans allmänna trend, på det viset att det predikterade priset går upp och ner där det aktuella priset också gör det.

Figur 15 Visualisering på hela tidsserien.

I den inzoomade visualiseringen (se fig. 16) av hela tidsserien för de första månaderna under året 2020 kan det konstateras hur lite alla priser skiljer sig från varandra. Dock går linjen på prediktionerna upp och ner vid olika ställen jämfört med det aktuella priset. Då kryptovalutans pristrend stiger mycket tenderar prediktionerna att resultera i en lägre trend än det aktuella priset. Däremot när kryptovalutans pristrend sjunker mycket tenderar prediktionerna att visa en högre trend än det aktuella priset. Då kryptovalutans pristrend varken stiger eller sjunker mycket verkar det som att prediktionerna i viss mån följer det aktuella prisets trend ganska exakt.

Figur 16 Inzoomad tidsserie för de första månaderna under året 2020.

5.2 Sammanfattning

Resultaten och visualiseringarna visar att den utvalda modellen kan prediktera väldigt noggrannt på kryptovalutans tidsserie. Det vill säga helt tydligt har modellen tränat bra på träningsdelen och lyckats skapa prediktioner med en hög noggrannhet på den oberörda testdelen. Valideringsdelens förlustvärde är lite lägre än träningsdelens förlustvärde under hela träningen, vilket enligt Brownlee (2019) betyder att modellen möjligtvis predicerar enklare på valideringsdelen jämfört med träningsdelen. Dock betyder det inte att modellen predicerar dåligt, utan enligt Brownlee (2019) betyder det att valideringsdelen kanske är för liten jämfört med träningsdelen. I varje fall överanpassar inte modellen, för enligt Brownlee (2019) överanpassar modellen först då träningsdelens förlustvärde blir lägre än en valideringsdelens förlustvärde. Däremot kunde valideringsdelen varit lite större för att vidare optimera modellen till en "perfect fit" modell. Dessutom då ett MSE värde på 0,0 är det bästa möjliga MSE värdet en modell kan ha, blev den utvalda modellens MSE värde väldigt bra med ett MSE värde på 0,001557.

6 SLUTSATSER

Den här studien har visat att återkommande neuronnät, specifikt en LSTM modell med flera lager kan effektivt och noggrannt skapa prediktioner av en tidsserie på en kryptovaluta. Dock kunde det göras mer arbete i framtiden för att ytterligt optimera LSTM modellen.

6.1 Diskussion

Att bestämma vilka verktyg, parametrar och funktioner att utnyttja för att skapa en effektiv och noggrann LSTM modell kan vara svårt utan en ordentlig förståelse av dessa. Därför gav studien en inblick på hur ett återkommande neuronnät, LSTM, fungerar och hur det kan tränas och användas, inklusive parametrar och funktioner, för att skapa prediktioner. Dessutom förklarades det hur en kryptovaluta fungerar som en datamängd och hur det måste förbehandlas före träningen. Ett antal visualiseringar användes också för att klargöra den teoretiska bakgrunden, datamängden, arkitekturen och resultaten. Fastän det fanns alternativ till andra verktyg var valet av verktygen självklart, speciellt

Python, Tensorflow och Keras, på grund av att de fungerar bra tillsammans och har visat sig vara populära att använda för liknande implementeringar.

Experimenten förenklade valet av en LSTM modell som fungerade bäst för denna tidsserie av en kryptovaluta. Speciellt hjälpte experimenten att bestämma antalet LSTM lager, epoker och satsstorleken, för att avgöra en slutlig LSTM modell och arkitektur för implementeringen. Den slutliga implementeringen visade sig uppnå en god inlärning och resultaten på de prediktioner som skapades visade också hur nära det predikterade priset var jämfört med det aktuella priset.

6.2 Framtida arbeten

Det här examensarbetet fokuserade på hur en LSTM modell kan användas för att skapa noggranna prediktioner. I framtiden skulle det vara intressant att vidareutveckla modellen och göra fler tillämpningar med den utvalda modellen. Ett exempel är att optimera den till en "perfect fit" modell, genom att köra fler experiment för att bestämma den mest optimala storleken på valideringsdelen.

Ett kompletterande element till studien kunde vara att göra flera liknande experiment parallellt för att studera hur Covid-19 möjligtvis kommer att påverka kryptovalutan de kommande åren. Ett aktuellt forskningsområde som kan förändra framtidsutsikterna nämnvärt.

Slutligen skulle det vara intressant att utföra tillämpningar som är utanför den här studiens tillämpningsområde. Några möjligheter är att skapa nya egenskaper och anlaysera när på tiddsserien det lönar sig att köpa eller sälja för att göra vinst.

KÄLLOR

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Schuster, M., Monga, R., Moore, S., Murray, D., Olah, C., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015, *Tensorflow : Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Tillgänglig på https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf Hämtad: 19.2.2020.

Bengio, Y., Simard, P., Frasconi, P., 1994, Learning long-term dependencies with gradient descent is difficult. I: *IEEE Transactions on Neural Networks*, Vol.5, no.2 s. 157-166.

Brownlee, J., 2017, *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Tillgänglig på https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ Hämtad 10.2.2020.

Brownlee, J., 2019, *How to use Learning Curves to Diagnose Machine Learning Model Performance*. Tillgänglig på https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/ Hämtad 24.3.2020.

Chollet, F., 2015, *Keras*. Tillgänglig på https://github.com/fchollet/keras Hämtad: 19.2.2020.

Donges, N., 2019, *Recurrent Neural Networks 101: Understanding the Basics of RNNS and LSTM*. Tillgänglig på https://builtin.com/data-science/recurrent-neural-networks-and-lstm Hämtad 25.1.2020.

Gamboa, J., 2017, *Deep Learning for Time-Series Analysis*. Tillgänglig på https://arxiv.org/pdf/1701.01887.pdf Hämtad: 14.3.2020.

Glosser.ca., 2013, *Colored neural network,* Wikipedia Commons. Tillgänglig på https://upload.wikimedia.org/wikipedia/commons/4/46/Colored_neural_network.svg Hämtad 26.1.2020.

Graves, A. & Schmidhuber, J., 2009, Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. I: Bengio, Y, Schuurmans, D, Lafferty, J.D, Williams, C.K.I och Culotta, A, *Advances in Neural Information Processing Systems 22*, MIT Press, Vancouver, s. 545-552.

Graves, A., Mohamed, A.R. & Hinton, G., 2013a, Speech Recognition with Deep Recurrent Neural Networks. I: 2013 *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Vancouver, BC, s. 6645-6649.

Graves, A., 2013b, *Generating Sequences With Recurrent Neural Networks.* Tillgänglig på https://arxiv.org/pdf/1308.0850.pdf Hämtad: 24.3.2020.

Gupta, D., 2020, *Fundamentals of Deep Learning - Activation Functions and When to Use Them?*. Tillgänglig på https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/ Hämtad: 13.2.2020.

Hochreiter, S., 1991, *Untersuchungen zu dynamischen neuronalen Netzen*.

Hochreiter, S. & Schmidhuber, J., 1997, *Neural Computation : Long Short-Term Memory*, Vol. 9, No. 8, s. 1735-1780.

Kingma, D., Ba, J., 2014, Adam: A Method for Stochastic Optimization. I: *International Conference on Learning Representations*.

McKinney, W., 2010, Data Structures for Statistical Computing in Python. I: *Proceedings of the 9th Python in Science Conference*, Vol. 445, s. 51-56.

Mittal, A., 2019, *Understanding RNN and LSTM*. Tillgänglig på https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e Hämtad 25.1.2020

Nicholson, C., 2019, *A Beginner's Guide to LSTMs and Recurrent Neural Networks*. Tillgänglig på https://pathmind.com/wiki/lstm#recurrent Hämtad 11.2.2020.

Nguyen, M., 2018, *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. Tillgänglig på https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 Hämtad 25.1.2020

Olah, C., 2015, *Understanding LSTM Networks*. Tillgänglig på http://colah.github.io/posts/2015-08-Unerstanding-LSTMs/ Hämtad 22.1.2020

Oliphant, T.E., 2006, A *guide to Numpy*, Vol. 1, Trelgol Publishing, USA.

Parmar, R., 2018, *Common Loss functions in machine learning*. Tillgänglig på https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23 Hämtad 10.2.2020.

Quandl, 2020, *Bitcoin Markets (krakenEUR)*. Datamägnden är tillgänglig på https://quandl.com/data/BCHARTS/KRAKENEUR Hämtad 15.3.2020.

Roondiwala, M., Patel, H. & Varma, S., 2017. Predicting Stock Prices Using LSTM. I: *International Journal of Science and Research (IJSR)*, Vol. 6, No. 4.

Rouse, M., Brush, K., Burns, E., 2019a, *Deep Learning.* Tillgänglig på https://searchenterpriseai.techtarget.com/definition/deep-learning-deep-neural-network Hämtad 1.1.2020.

Rouse, M., Burns, E., Burke, J., 2019b, *Artificial Neural Network*. Tillgänglig på https://searchenterpriseai.techtarget.com/definition/neural-network Hämtad 25.1.2020.

Sagar, A., 2019, *Cryptocurrency Price Prediction Using Deep Learning*. Tillgänglig på https://towardsdatascience.com/cryptocurrency-price-prediction-using-deep-learning-70cfca50dd3a Hämtad 24.1.2020.

Shah, T., 2017, *About Train, Validation and Test sets in Machine Learning*. Tillgänlig på https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7 Hämtad 19.2.2020.

Srivastava, P., 2017, *Essentials of Deep Learning : Introduction to Long Short Term Memory*. Tillgänglig på https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/ Hämtad 11.2.2020.

Tarwani, K.M. & Edem, S., 2017, Survey on Recurrent Neural Network in Natural Language Processing. I: *International Journal Of Engineering Trends and Technology (IJETT)*, Vol. 48, No. 6, s. 301-304.

Torres, J., 2018, *First contact with Deep Learning Practictal introduction with Keras*. Tillgänglig på https://torres.ai/first-contact-deep-learning-practical-introduction-keras/ Hämtad 25.1.2020.

Valkov, V., 2019, *Cryptocurrency prediction using LSTMs*. Tillgänglig på https://towardsdatascience.com/cryptocurrency-price-prediction-using-lstms-tensorflow-for-hackers-part-iii-264fcdbccd3f Hämtad 31.1.2020.