

Prediktera antalet fall av covid-19 med hjälp av artificiell intelligens

Wilmer Johansen & Joy Sabbagh

Handledare: Andreas Wiman, Jan-Erik Hägglöf



**KAROLINSKA
GYMNASIET**

Gymnasiearbete, 100 poäng

6 april 2022

Abstract

Title: Forecasting future number of covid-19 cases with the help of artificial intelligence

Keywords: covid-19, artificial intelligence, model, LSTM, prediction.

An epidemic of the infectious disease covid-19 struck the world unexpectedly in December 2019. This contagious disease proved to be a severe acute respiratory syndrome from the coronavirus family which consequently received the scientific name SARS-CoV-2. Its influenza-resembling symptoms plagued many nations and resulted in the death of millions, threatening the economy and politics of hundreds of civilizations. Therefore, a predictive model of future number of cases becomes crucial in such a critical situation. Thus, the aim of this study is to construct a model backed by artificial intelligence with the ability to forecast future number of covid-19 cases. To accomplish said ambition, the question to be investigated abides as such: Can a model for prediction of future cases of covid-19 produce adequate results with the help of artificial intelligence?

For the construction of the model, multiple preliminary steps were required. First and foremost, finding and gathering covid-19 statistics from public databases such as Our World in Data and country-specific authorities e.g., Folkhälsomyndigheten. The secondary step consisted of compiling all the found statistics into a singular multivariable dataset. The final performed step was to implement a supervised learning function from which the model could learn and adapt. When all preliminaries were taken care of, the training segment of the model took place in Python with the help of third-party libraries designed specifically for the creation of AI such as TensorFlow and Keras.

To examine the performance of the model, 3 different dates have been chosen in which the artificial intelligence will train on the previous data for each corresponding date with the aim to predict each day for two weeks for a total of 14 days into the future. The results, calculated with MSE (Mean Squared Error) and RMSE (Root Mean Squared Error), reveal that the model can predict future cases of covid-19 with an average MSE of 0,0087 and an average RMSE of 0,0778. To consider a MSE value optimal, the value must be within in an acceptable range. Therefore, in accordance with the results, we can draw the conclusion that it is possible to develop a model for prediction of future cases of covid-19 with the help of artificial intelligence with adequate results.

Innehållsförteckning

Abstract	i
1 Inledning	1
1.1 Syfte	1
1.2 Frågeställning	2
1.3 Avgränsningar	2
2 Bakgrund	3
2.1 Tidsserier	3
2.2 Python	3
2.3 Mjukvarubibliotek	4
2.3.1 TensorFlow	4
2.3.2 Keras	4
2.4 Covid-19	5
2.4.1 Oroväckande varianter	5
2.5 Artificiell intelligens	6
2.6 Maskininlärning	7
2.6.1 Väglett lärande	7
2.6.2 Förstärkningsinlärning	8
2.6.3 Överanpassning	8
2.6.4 Optimering	8
2.6.5 Hyperparametrar	9
2.7 Djupinlärning	9
2.7.1 Cybernetik	10
2.7.2 Artificiella neuronnät	11
2.7.3 Rekurrenta neuronnät	13
2.7.4 Långa korttidsminnesnät	15
2.8 Modellutveckling	19
2.9 Utvärderingsmått	21
3 Metod	22
3.1 Datainsamling	22

3.2	Genomgång av källkod	27
3.3	Justering av hyperparametrar	38
3.3.1	Val av datum	39
4	Resultat	40
4.1	Modell för 2021-04-09	40
4.2	Modell för 2021-05-04	44
4.3	Modell för 2021-09-20	48
4.4	Jämförelse av utvärderingsmått	52
5	Diskussion	53
6	Slutsats	55
7	Källförteckning	56
8	Bilagor	59
8.1	Bilaga 1	59

1 Inledning

I december 2019 påträffade smittskyddsmyndigheten i Kina en ny slags virusvariant av familjen coronavirus bland människor som besökt en lokal marknad i staden Wuhan. Viruset kom senare att döpas till SARS-CoV-2 och den luftvägsinfektion som orsakas till följd av det är mest känd som covid-19. Den 31 januari konstaterades det första fallet av det nya coronaviruset i Jönköping i Sverige. Vad ingen visste då var att den kommande smittspridningen skulle leda till en världsomfattande global pandemi. Symptomen för covid-19 liknar de för influensa men kan skilja mellan olika personer från att inte uppfattas över huvud taget till att kräva intensivvård som kan resultera i döden. Sjukdomen överförs i huvudsak mellan människor genom så kallad droppsmitta. Framtagningen av ett vaccin mot covid-19 påbörjades redan inom en månad av det första fallet men det skulle dröja fram tills december 2020 innan massvaccineringen inleddes i flertalet länder, däribland Sverige. Sedan begynnelsen har flertalet olika virusvarianter utvecklats, alla med olika egenskaper rörande smittspridning, dödlighet och vaccinresistens. Regeringar världen över har vidtagit omfattande smittskyddsåtgärder och infört aldrig tidigare i sådan utsträckning beprövade restriktioner i hopp om att lyckas ”plana ut kurvan”. Pandemin till följd av covid-19 har hittills resulterat i otaliga mängder dödsfall och enorma förluster i form av tillgångar, men har samtidigt medfört insiktsfull information om hur framtida pandemier bör hanteras.

Statistik för antalet fall per dag av covid-19 utgör ett bra exempel på data i form av en tidsserie: en sekvens av data med successivt lika stora tidsavstånd. Med hjälp av artificiell intelligens går det att modellera komplexa algoritmer för prediktion av framtida data för tidsserien. Denna form av extrapolering kan uppnås genom att studera historiska data eller andra tidsserier som tros ha en inverkan.

1.1 Syfte

Syftet med arbetet är att ta fram en modell för prediktion av antalet framtida fall av covid-19 med hjälp av artificiell intelligens. Detta ska uppnås genom en utförlig undersökning för metoder avseende området maskininlärning, och med underlag i teorin självständigt programmera ovan preciserade modell i programspråket python.

1.2 Frågeställning

För att uppnå syftet med arbetet ska således följande frågeställning besvaras i slutsatsen med avseende på de resultat som presenterats:

- Kan en modell för prediktion av antalet framtida fall av covid-19 framställas med tillfredsställande resultat till hjälp av artificiell intelligens?

1.3 Avgränsningar

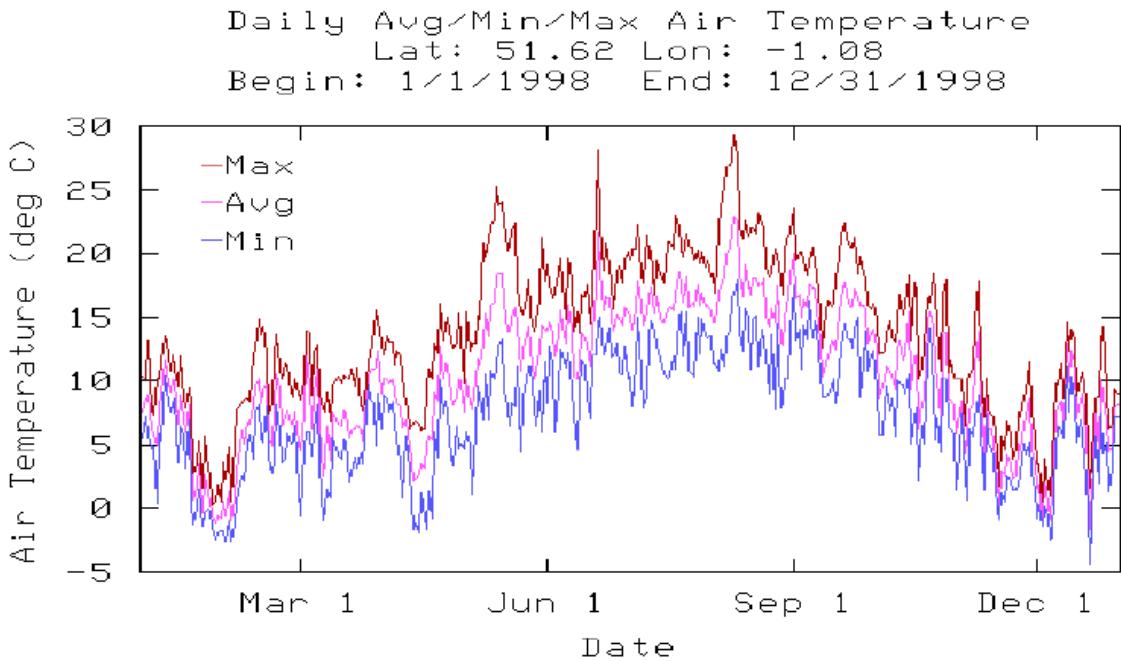
Till följd av arbetets omfattning och begränsade tidsram har vissa avgränsningar gällande metodiken gjorts. I enlighet med arbetets slutdatum har endast tidsseriedata från och med startdatumet 2020-02-02 fram till 2022-02-04 (motsvarande 735 dagar) använts. Av samma bakgrund har även datasetet avgränsats till att enbart inkludera data för Sverige. Den begränsade tidsramen har även medfört att optimeringen och justeringen av hyperparametrar för modellen gjorts i grova drag. Med hänsyn till teorin i bakgrundsdelen så har endast två olika sorters modeller av långa korttidsminnesnät använts för att framställa resultatet, även om många andra algoritmer kan användas för att prediktera data av tidsserier. Gällande resultatet så har längden för samtliga prediktioner förutbestämts till 14 dagar för att begränsa tidsramen.

Sedermera klassas covid-19 dessutom inte längre som en samhällsfarlig sjukdom efter 1 april, och där de flesta regioner avstår från testning, varför framtida data ej går att använda av naturliga skäl [1].

2 Bakgrund

2.1 Tidsserier

En tidsserie (engelska: time series) är en serie datapunkter med observationer över en given tidsperiod. Den vanligast förekommande typen av tidsserier utgörs av en enhetlig sekvens bestående av successivt utmätt data med samma tidsintervall mellan varje datapunkt. Detta kan sedan kombineras med flera olika oberoende mätningar för att framställa ett tidsseriedataset (engelska: time series dataset). Ett tidsseriedataset används i synnerhet för att identifiera och modellera systematiska variationer samt göra framtidsprognosar [2].



Figur 1: Exempel på en tidsserie för den dagliga lufttemperaturen för en viss geografisk plats över ett års tid visualisering med hjälp av en graf. [3]

2.2 Python

Python är ett programmeringsspråk på hög nivå som lanserades 1991 av Guido van Rossum. Språket tillkom som en efterträdare till programspråket ABC. Skrivsättet utvecklades med en klar vision om kodläsbarhet i åtanke, varför språkets syntax präglas av enkelhet utan ”överflödiga” symboler och beteckningar. Ett tydligt exempel på detta är att Python saknar det annars vanligt förekommande kravet att varje rad ska avslutas med ett semikolon. Språkets lättillgänglighet har lett till att det idag är ett av de populäraste programspråken och tillämpningar finns inom åtskilliga domäner. Python

har en öppen källkod, vilket innebär att varje användare eller intresserad utomstående har fri åtkomst till att observera hur språket är uppbyggt från grunden [4]. Detta särdrag är en anledning till de åtskilliga olika utvecklingsmiljöer, standardbibliotek och ett stort antal tredjehandsbibliotek som utvecklats av utomstående parter. Python används ofta inom områden såsom datorgrafik, textbehandling, spelprogrammering, maskininlärning, datavisualisering eller vetenskapliga beräkningar [5].

2.3 Mjukvarubibliotek

Flera olika färdiga mjukvarubibliotek kan installeras och sedan importeras till ett projekt för att snabbt och enkelt utöka programmets möjliga användningsområden. Mjukvarubibliotek medför att skaparen av programmet själv inte behöver utforma rutiner och funktioner från grunden varje gång de behöver tillämpas, utan kan istället direkt använda sig utav de färdigt framställda funktioner som importerats. Gränssnittet mellan biblioteket och programmeringsspråket kallas applikationsprogrammeringsgränssnitt (ofta förkortat API) och utgör den del som möjliggör användningen av de färdiga funktionerna. Sådana bibliotek kan till exempel innehålla funktioner som framhäver grafik, grafer, matematiska uttryck, datahantering eller möjliggör utveckling av artificiell intelligens [6].

2.3.1 TensorFlow

Tensorflow är ett mjukvarubibliotek som avser numerisk beräkning genom flödesdiagram och bistår en utförlig API för att utföra sådana räkneoperationer. TensorFlow utvecklades 2015 av forskare och ingenjörer på Google Brain, Googles egna dedikerade maskininlärnings-forskningsgrupp. Programbiblioteket har en bred tillämpning inom artificiell intelligens men fokuserar primärt på djupinlärning [7].

2.3.2 Keras

Mjukvarubiblioteket Keras definieras såsom att det ”ansvarar för att förse Python med ett gränssnitt för artificiella neuronnät”. Keras agerar som en förbindelse mellan TensorFlow och tredje parters specifika implementationer av transferfunktioner, glömgrindar och optimeringsfunktioner. Detta utan en förminskning av programmets prestanda eller den kognitiva belastningen [8].

2.4 Covid-19

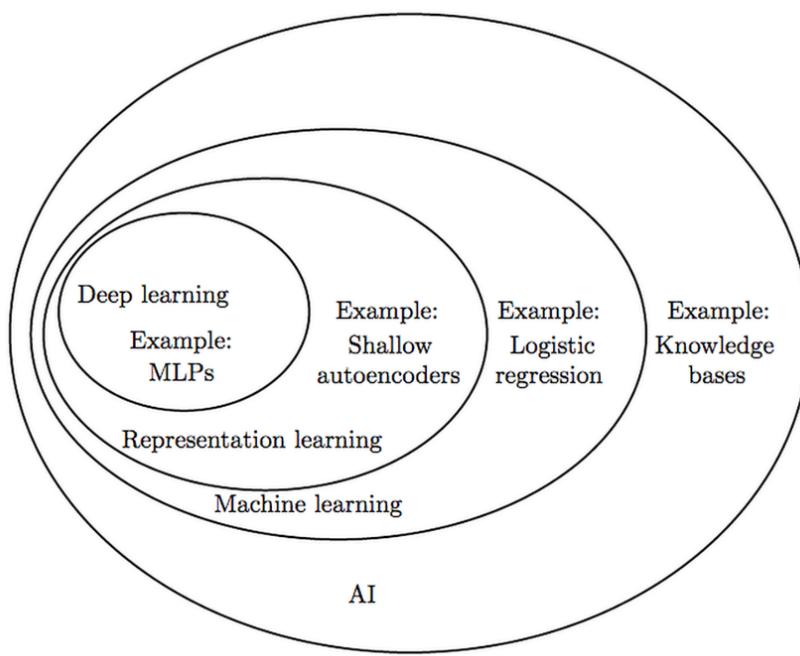
Alla virus, inklusive Sars-CoV-2 som orsakar luftvägsinfektionen covid-19, förändras över tid. De allra flesta mutationers inverkan ger endast en obetydlig grad av förändringar av virusets egenskaper. I vissa fall uppstår dock varianter vars mutationer förorsakar en ökad morbiditet, virulens, mortalitet och eventuellt en ökad resistens gentemot vacciner, terapeutiska läkemedel eller diagnostiska verktyg [9]. Att virus som trots allt saknar både intelligens och egen vilja kan mutera beror på evolutionära processer. Medan det naturliga urvalet styr en populations avkomma har medvetslösa virus en dito process: egenskaper som ökar sannolikheten för fortplantning förs vidare till nästkommande generationer [10].

2.4.1 Oroväckande varianter

Världshälsoorganisationen (WHO) har sedan början av 2020 i samarbete med flertalet nationella myndigheter, expertnätverk, institutioner och forskare övervakat utvecklingen av covid-19 och samlat in data för utvärdering av olika virusvarianters utveckling. Den första samhällsfarliga varianten, betitlad alfavarianten (B.1.1.7), upptäcktes i mitten av november år 2020. Alfavarianten utmärktes av en betydligt högre mortalitet än vad en infektion av ordinär Sars-CoV-19 innebar och bedömdes därmed som en oroväckande variant (engelska: Variant of Concern, förkortat VOC) [11]. Nästa variant av särskild betydelse, betavarianten (B.1.351), spreds betydligt snabbare än sin föregångare och hade dessutom förmågan att ackumulera flera mutationer [12]. Deltavarianten (B.1.617.2) var ytterligare en bekymmersam variant som åtminstone hade en transmissions- och dödlighetsgrad ekvivalent till, och i vissa fall till och med högre än alfavarianten [13]. Den fjärde oroväckande varianten, omikronvarianten (B.1.1.529), avviker från sina företrädare genom sina ovanligt många mutationer som uppgår i över 30 stycken. Omikronvariantens höga spridningshastighet kombinerat med dess förmåga att undvika både dubbelvaccination och kroppens eget immunförsvar har placerat varianten som en VOC [14].

2.5 Artificiell intelligens

Artificiell intelligens, ofta förkortat AI, är ett samlingsbegrepp för förmågan hos datorprogram eller robotar att efterlikna andra entiteters¹ naturliga intelligens. Forskare har länge strävat efter att åstadkomma en allmänt accepterad definition av AI med hänsyn till dess många olika användningsområden, varför någon regelrätt sådan inte finns. Ett försök till detta kan däremot lyda såsom följande: ”Artificiell intelligens som område är konstruktionen av intelligenta agenter som upptar information om sin miljö och vidtar åtgärder som påverkar denna miljö”. AI omfattar ett stort antal olika tillämpningsområden och täcker allt från generella syften som inlärning, problemlösning och självmedvetande till mer specifika situationer som att spela schack, skriva poesi eller självkörande bilar. Historiskt sett har forskare strävat efter att uppnå flera olika sorters artificiell intelligens. De två vanligaste av dessa brukar vara att antingen försöka efterlikna, alternativt om möjligt överträffa mänsklig prestanda, eller med en mer abstrakt definition av intelligens som rationell i åtanke – kort sagt att ”göra det rätta” utifrån den rådande situationen [15, s. 19–52].



Figur 2: Venndiagram visande en klassisk uppdelning av korrelationen mellan artificiell intelligens och dess olika delmängder samt exempel på dessa.

¹Ett objekt av något slag, till exempel en biologisk organism.

2.6 Maskininlärning

Maskininlärning (engelska: machine learning) är det delområde inom artificiell intelligens som avser metoder för inlärning på egen hand. Vanligtvis lär sig en entitet om den förbättrar sin prestationsförmåga utifrån de observationer den gjort av sin omgivning. Om denna entitet är en dator kallas inlärningen för maskininlärning: en dator observerar sin miljö och tar fram en modell för att lösa en uppgift med avseende på miljön baserat på sina observationer, helt utan någon förutfattad mening eller information om uppgiften sedan tidigare [15, s. 669–671].

2.6.1 Väglett lärande

Väglett lärande (engelska: supervised learning) är en viss typ av maskininlärning. Precis som namnet indikerar innefattar inlärningen någon form av vägledning, i detta fall indata parat med en respektive utdata. Det är kopplingen mellan dessa två som datorn lär sig att identifiera och skapar en modell för att kunna replikera framtida inmatade uppgifter. Ett exempel på väglett lärande är ett dataset med olika kamerabilder på bussar och bilar där varje bild har en tillhörande beskrivning, en så kallad *etikett*, i detta fall antingen ”buss” eller ”bil”. Datorn skapar sedan en modell för datasetet och försöker sedan när det blir utsatt för en ny bild av något av objekten para ihop denna med en passande etikett. I kontrast till detta finns icke-väglett lärande, där indata inte har någon tillhörande etikett och datorn således tvingas klustra (gruppera) ihop datan på egen hand. Formellt gäller för väglett lärande följande:

Givet ett dataset med n antal indata-utdata par

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

där varje par genererats av en okänd funktion $y = f(x)$ är målet att upptäcka och modellera en funktion h som kan approximera den sanna funktionen f .

Funktionen h kallas för en *hypotes*² och är alltså modellerad utifrån det givna datasetet. Den sanna utdata y_i kallas för *sanningsvärde* (engelska: ground truth) och är den data eller etikett som modellen försöker förutspå och sedan jämförs med [15, s. 671–672].

²Ej att förväxlas med vad som inom vetenskapliga sammanhang oftast är ett antagande om verkligheten och används vid hypotesprövning.

2.6.2 Förstärkningsinlärning

Förstärkningsinlärning (engelska: reinforcement learning) är ett delområde inom maskininlärning där en agent³ lär sig efter en serie av handlingar, där varje handling genererar antingen en belöning eller ett straff beroende på handlingens värde. Agenten får sedan utvärdera vilka aktioner som frambringade belöningen eller straffet, och använda den nya vetskapsen för att försöka maximera sin belöning vid framtida handlingar i serien. På så sätt lär sig agenten av sina egna fel (eller rätt) och datan kräver inga etiketter [15, s. 671].

2.6.3 Överanpassning

Överanpassning (engelska: overfitting) är det fenomen som uppstår när en funktion fokuserar allt för mycket på den data som användes under inlärningsprocessen, och till följd av detta presterar avsevärt sämre när den utsätts för sedan tidigare okänd data. Oftast finns det en kompromiss mellan en mer komplex hypotes som passar träningsdatan bättre eller en mer ”enkel” hypotes som är bättre på att generalisera och således även presenteras inför ny data. Oftast är det lämpligast att välja en hypotes som är enkel men samtidigt inte försummar viktig information. Enkelhet är emellertid inte alltid lätt att definiera då modeller som följer maskininlärning ofta är tämligen bra att generalisera trots att de är mycket komplexa. Istället borde alltså hypotesen väljas utifrån lämpligheten till situationen sett och inte dess enkelhet. I kontrast till överanpassning så är en hypotes underanpassad om den misslyckas med att hitta mönster i datan [15, s. 673]. Generellt sett så ökar risken för överanpassning i takt med antalet parametrar som påverkar datan, och minskar desto fler exempel på indata som träningsdatan innehåller [15, s. 681].

2.6.4 Optimering

Målet med maskininlärning är att framställa en hypotes som är optimalt anpassad för framtida scenarion. För att uppnå detta krävs en mer precis definition av vad ”optimalt anpassad” respektive ”framtida scenarion” innebär. En viktig förutsättning är att framtida scenarion kommer likna det förflutna, vilket kallas för ett *stationaritetsantagande*. Detta innebär att varje exempel på ett scenario har samma sannolikhetsfördelning samt är oberoende föregående scenarion. Gällande ”optimalt anpassad” så är det när hypotesen

³Enskilt ombud, oftast en del av en större grupp agenter, som arbetar för ett datorprogram.

h åstadkommer minimal *felmarginal*: andelen gånger $h(x) \neq y$ för ett scenario i indata-utdata par (x, y) . Felmarginalen för en hypotes kan estimeras genom ett särskilt test. Detta åstadkoms genom att mäta prestandan för hypotesen på ett separat dataset enbart avsett för testningen. Det vore naturligtvis orättvist för en hypotes (eller student) att tjuvkika på provsvaren innan ett test genomförs. För att undvika att något sådant inte inträffar delas datasetet in i två delar: ett *träningsset* för att modellera hypotesen, och ett *testset* för att utvärdera hypotesens prestanda och felmarginal. Den hypotes som sammantaget presterade bäst med lägst felmarginal under testfasen väljs sedan ut som den slutgiltiga modellen för datasetet. [15, s. 683–684].

För att lyckas få ett maskininlärt program att modellera en så sanningsenlig hypotes som möjligt behövs oftast en viss förståelse för datan sedan tidigare. Detta kan enklast uppnås genom att genomföra olika sorters dataanalyser. Statistiska tester, visualisering av datan med hjälp av exempelvis histogram, färgdiagram eller lådagram kan alla hjälpa till att skapa en uppfattning om datan [15, s. 671].

2.6.5 Hyperparametrar

En hyperparameter är en parameter för reglering av algoritmen (och inte modellen i sig). Således påverkas den inte av algoritmen och måste fastställas innan träningsfasen samt förbli konstant vid det ursprungliga värdet genom hela körningen [16, s. 30]. Genom att finjustera hyperparametrarna för en viss algoritm så kan prestandan öka avsevärt. Detta uppnås enklast genom att ställa in värdena för hand: gissa ett värde med avseende på tidigare vetskapp om prestandan, träna modellen, jämföra prestandan med tidigare värden och med den nya ingivelsen återgå till steg ett igen. Denna process genomförs sedan kontinuerligt tills prestandan nått tillfredsställande resultat. Aktiviteten kan automatiseras genom att skapa en modell för att testa alla möjliga kombinationer och sedan välja ut den uppsättning med mest gynnsamt resultat [15, s. 689–690].

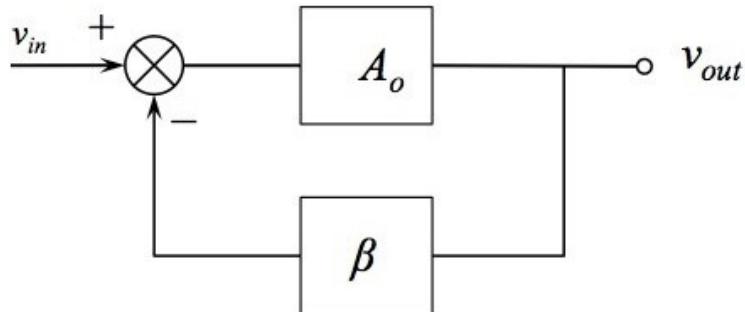
2.7 Djupinlärning

Djupinlärning (engelska: deep learning) är ett brett delområde inom maskininlärning som avser inlärning genom algoritmer med komplexa strukturer av algebraiska kretsar med justerbara kopplingar. Termen ”djup” hänvisar till att kretsarna oftast är arrangerade i

flera lager, och vägen från indata till utdata följer många olika räkneoperationer. Djupinlärning användes till en början inom forskning som försökte replikera hjärnans neuroner (nervceller), varför modeller som använder sig av djupinlärning ofta kallas för neuronnät även om de inte delar någon märkvärd likhet med biologiska neuroner. Djupinlärning används idag frekvent inom områden såsom bild- och taligenkänning, översättning samt har en framstående roll inom förstärkningsinlärning [15, s. 801].

2.7.1 Cybernetik

Begreppet cybernetik (engelska: cybernetics) myntades kort efter andra världskrigets slut som en allmän beteckning för studier avseende styrning av komplexa kommunikationssystem. Cybernetiska styrsystem är idag inom datorvetenskapliga sammanhang nära besläktat termer som reglerteknik och styrteknik. Ofta innefattar systemen någon form av *återkoppling* (engelska: feedback) där processade observationer används för att vidare reglera en signal i form av förstärkning eller försvagning [17]. Ett styrsystem med återkoppling består som enklast av en process och en regulator. Processenheten mottar indata i form av en insignal och avger en processad utdata till utsignalen. Även regulatorn mottar utsignalen och jämför denna med ett inre *börvärde* bias. Börvärdet är oftast förutbestämt för systemet och pekar mot utdatans önskade värde⁴. Regulatorn viktat utdata mot börvärdet, vars värde sedan kopplas samman med nästkommande indata för systemet. De inre funktionerna för processen och regulatorn utgörs som oftast av matematiska funktioner men kan till exempel även bestå av kriterier för temperaturregleringen av ett rum [18].



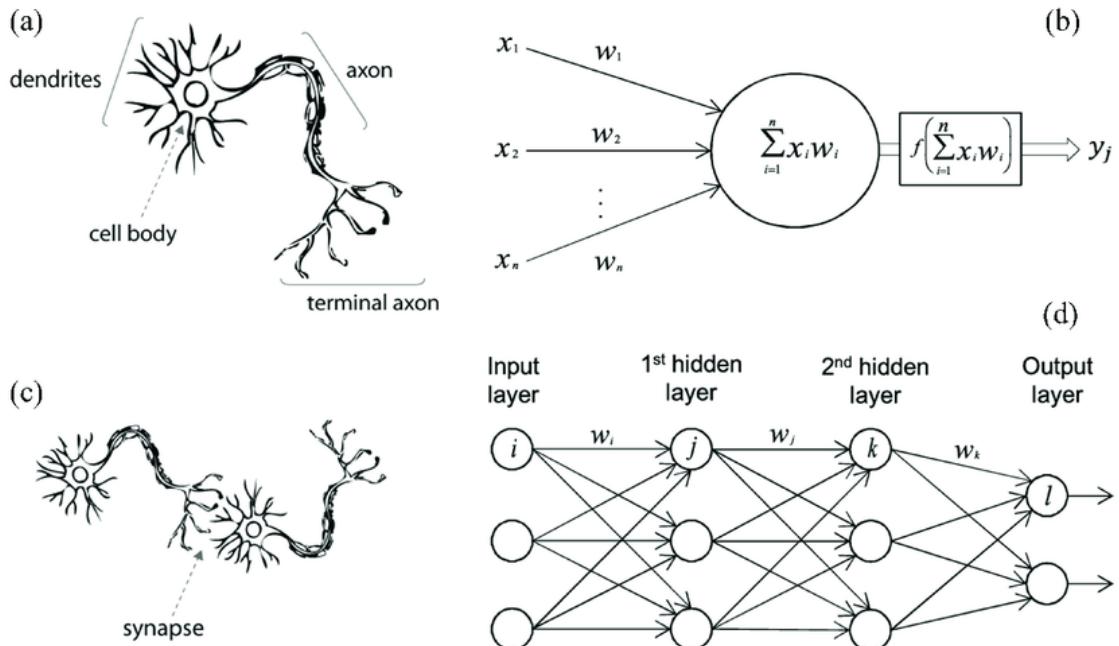
Figur 3: Enkelt styrsystem där utdata från processenheten A_O återkopplas med indata V_{in} i form av försvagning efter passering av regulatorn β [19].

⁴Vilket oftast motsvarar sanningsvärdet för modellen.

Cybernetik och reglertechnik har idag en framstående roll inom flertalet tillämpningsområden såsom rymdfart, telekommunikation, industri, elektronik eller som autopilot i flygplan.

2.7.2 Artificiella neuronnät

En neuron är inom sammanhanget den minsta beståndsdelen av ett större neuronnät [20, s. 32]. Delarna av en artificiell neuron kan jämföras med de av en biologisk: dendriter mottar information från andra neuroner och kan liknas vid indata; soma (cellkroppen) processar informationen från dendriterna och kan liknas vid artificiella cellkroppens kärna; axonet skickar den processade informationen vidare till nästa neuron och kan liknas vid utdata. Överföringen av information från en neuron till en annan kallas synaps. Trots dessa till synes distinkta jämförelser överträffar den inre komplexiteten av en biologisk cell markant den av en artificiell [21, s. 47–49].



Figur 4: Visuell jämförelse mellan biologiska neuroner och artificiella neuroner: (a) mänsklig neuron; (b) artificiell neuron; (c) synaps inom ett biologiskt neuronnät; (d) sekvens av synapser inom ett artificiellt neuronnät.

Den artificiella neuronens kärna är som enklast bestående av två funktioner som samverkar: en linjärt utformad *summafunktion*, samt en *transferfunktion* som kan utgöras av flertalet olika utseenden. Summafunktionen är i praktiken en linjär regression där varje indata $v_j^{(i)}$ är multiplicerad med en vikt $w_j^{(i)}$ för att producera utdata, den viktade

summan z_j . De viktade värdena är koefficienter som det neurala nätverket lär sig genom den givna datan. Även ytterligare en indata *bias*, b adderas för att justera datan och neuronens aktivitet [20, s. 33].

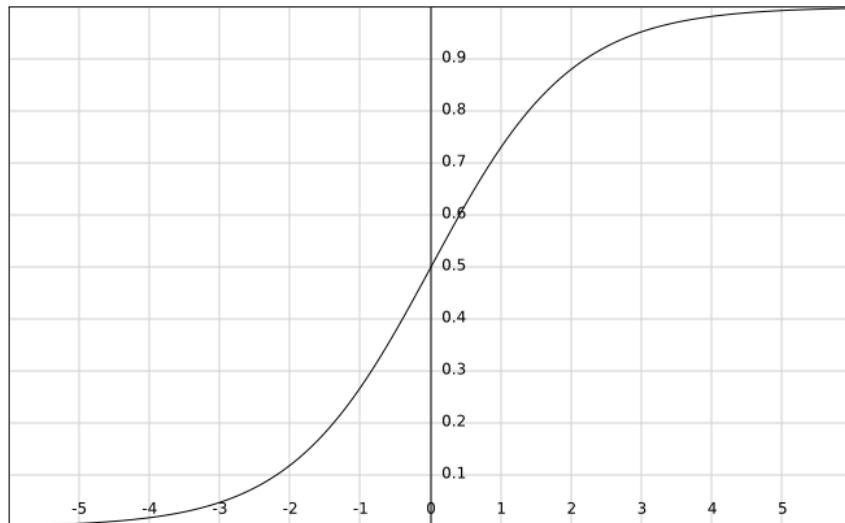
Summafunktionen kan följaktligen uttryckas som

$$z_j = \sum_{i=1}^n v_j^{(i)} w_j^{(i)} + b \quad (1)$$

där z_j är den viktade summan i ett steg mellan neuron i och j [15, s. 803].

Neuronens andra beståndsdel transferfunktionen tillämpas för att skapa en olinjärhet mellan neuroner. Dessa funktioner beror av summafunktionen z som indata. En av de vanligaste transferfunktionerna är *sigmoid*-funktionen. När denna appliceras så översätts utdata från neuronen till ett värde mellan 0 och 1, där små tal närmrar sig 0 och stora tal närmrar sig 1. Sigmoidfunktionen betecknas ofta med σ och kan se ut såsom följande:

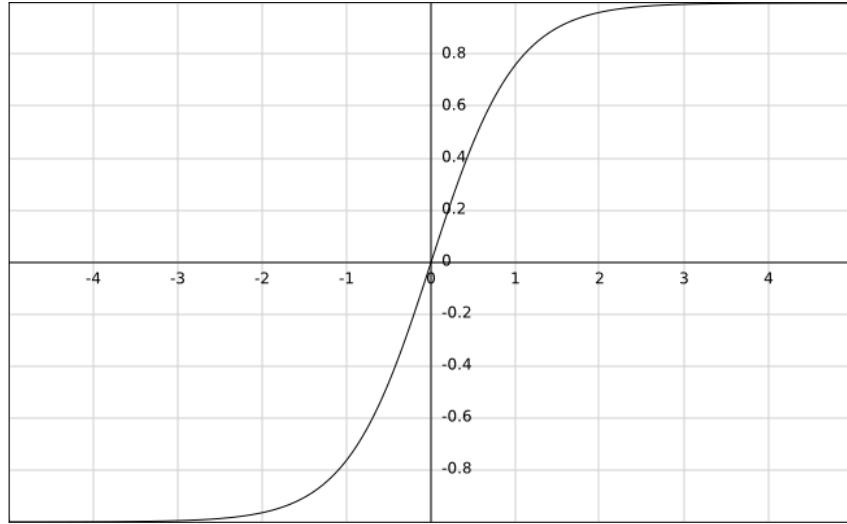
$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (2)$$



Figur 5: *Sigmoid* funktion

En annan vanlig transferfunktion är den hyperboliska funktionen *tanh*. Till skillnad från sigmoida funktionen så närmar sig nu små tal -1 och stora tal 1. Notera att *tanh* är en skiftad version av *sigmoida* funktionen då $\tanh(z) = 2\sigma(2z) - 1$. Tanh ser ut som följande:

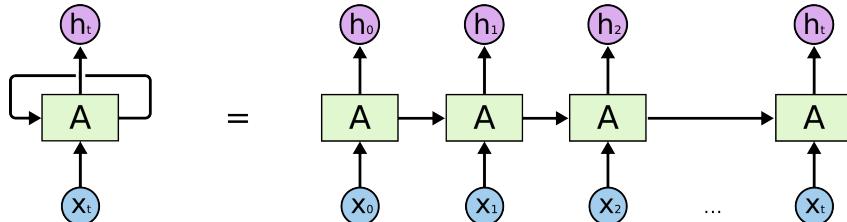
$$\tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \quad (3)$$



Figur 6: *Tanh*-funktion

2.7.3 Rekurrenta neuronnät

Rekurrenta neuronnät (engelska: recurrent neural networks, häданefter förkortat RNN) skiljer sig en aning från ordinära neuronnät i den mån att de möjliggör för en serie av uträkningar. Den mest distinktiva skillnaden är att denna serie tillåter neuronen att innehålla ett inre tillstånd i formen av ett minne. Indata som passerat för flera iterationer av tidsserien sedan kan alltså fortgå att påverka neuronens bearbetning av det nuvarande stegets indata [15, s. 824].



Figur 7: (Vänster): Flödesschema för ett simpelt RNN där utdata från det gömde lagret A återanvänds igen vid nästa iteration; (Höger): Samma flödesschema utvecklat t iterationer där utdata h_t både lämnar nätverket och kan återanvändas igen i kombination med indata x_{t+1} vid nästa iteration. [15]

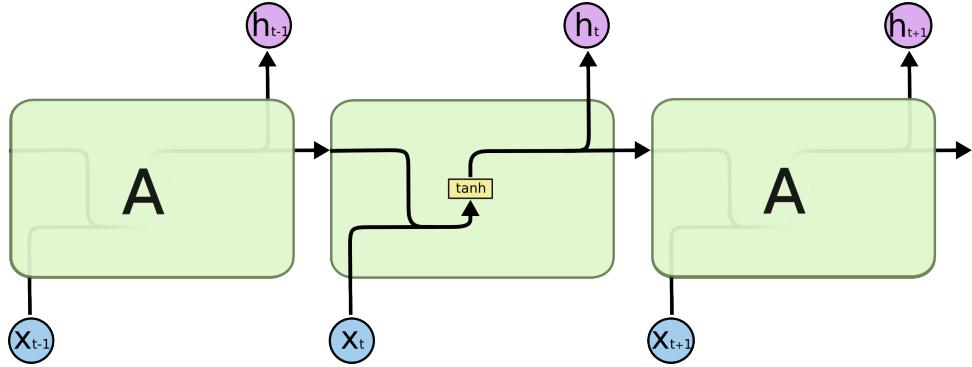
Vanliga RNN som visualiseras i Figur 5 använder sig av ett eget viktat värde för att blanda den återkommande indata från föregående iteration med den nuvarande indata och dess viktade värde. Summan av dessa passerar sedan en transferfunktion, oftast *tanh*, och viktas igen som utdata. Flödet kan beskrivas som följande för nuvarande tidssteg t :

$$z_t = Uh_{t-1} + Vx_t + b_1 \quad (4)$$

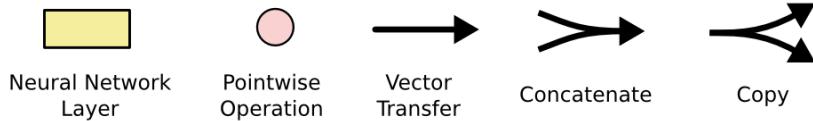
$$a_t = \tanh(z_t) \quad (5)$$

$$h_t = W a_t + b_2 \quad (6)$$

där z_t är den linjära viktade summan av återkommande dataan h_{t-1} och nuvarande indata x_t , båda viktade med U respektive V , samt bias b_1 . Det gömda lagret a_t är det icke-linjära värdet för den viktade summan när den passerat transferfunktionen \tanh . Till sist viktas det icke linjära värdet a_t med vikten W och justeras med bias b_2 till utdataan h_t . Utdataan lämnar sedan nätverket och passeras vidare till iterationen för tidssteg $t + 1$ [15].



Figur 8: Utvecklat flödesschema för ett RNN där den viktade summan av återkommande dataan h_{t-1} och indataan x_t passerar transferfunktionen \tanh i det gömda lagret a_t och sedan skickas vidare till nästa tidssteg $t + 1$. [22]



Figur 9: Förklaring av symboler för ovan stående figur samt följande figurer inom detta område. Gula rutor symbolisera lager i det neurala nätet och består oftast av olika transferfunktioner. Rosa cirklar symbolisera elementvisa operationer av vektorer, till exempel multiplikation eller addition. Pilar symbolisera överföring av vektorer med data som utdata från en nod till indata vid en annan nod. Pilar kan både konvergera, då data sammefogas, eller divergera då data kopieras och förs vidare mot två olika destinationer. [22]

En bristfällighet bland RNN beskrivna så som ovan är lättheten för viktig data som passerar tidigt i en sekvens att glömmas bort och helt försvinna ur minnet allt eftersom ny data passerar. Matematiskt sett så kan innehållet i minnet både försvinna och ta över helt. Denna bristfällighet innebär i praktiken att traditionella RNN möter stora besvär i att lära sig långsiktiga mönster i dataan [20, s. 254–256].

2.7.4 Långa korttidsminnesnät

Långa korttidsminnesnät (engelska: long short-term memory networks, häданefter förkortat LSTM) är en variant av RNN specialiserade på att bibehålla information över många tidssteg. Det inre långtidsminnet i ett LSTM utgör en minnesbuss, ofta betecknad C , och kopieras från ett tidssteg till det nästa. (Till skillnad från traditionella RNN som viktat minnet mellan varje iteration, som kan ses i Ekvation 6). Ny information tillgår minnet genom att adderas i form av kontinuerligt linjära uppdateringar, vilket försummar risken för information att exponentiellt ta över minnet. LSTM innehåller även flertalet *grindar* i formen av vektorer för att styra flödet av information. Informationen passerar respektive grind genom elementvis multiplikation i form av matriser. Grindarna i ett LSTM är följande [15]:

- Glömgrind (engelska: forget gate): betecknas f och avgör om passerande element i minnesbussen ska kommas ihåg (kopieras till nästa tidssteg) eller glömmas bort (återställas till 0).
- Ingångsgrind (engelska: input gate): betecknas i och avgör om passarande element från minnesbussen ska adderas med indata från det nuvarande tidssteget.
- Utgångsgrind (engelska: output gate): betecknas o och avgör om passarande element från minnesbussen ska överföras till korttidsminnet z (likt det gömda lagret a i traditionella RNN).

Grindar i ett LSTM skiljer sig något från de booleska grindar som traditionellt används vid flödeshantering i den mån att de är ”mjuka”—ett exempel på detta är att element kan passera glömgrinden och glömmas bort om värdet är tillräckligt litet men ändå inte 0. Värdet för en enhet som passerar en grind ligger alltid inom värdemängden $[0, 1]$ då all indata passerar en *sigmoid* funktion innan den processas.

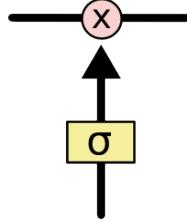
Respektive grind kan således tecknas såsom följande:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (7)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (8)$$

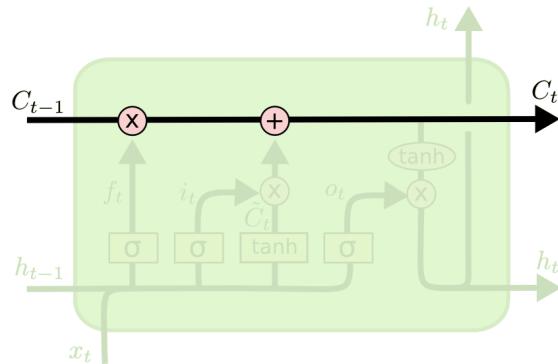
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (9)$$

där W är vikten och b bias för respektive funktion. Notera att alla grindar passerat genom den *sigmoida* transferfunktionen innan.



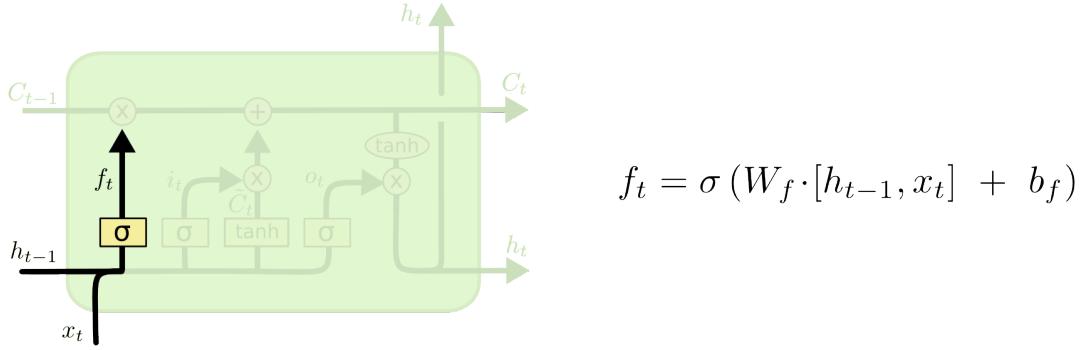
Figur 10: Visuell representation för ett exempel av en grind i ett flödesschema. En vektor passerar en *sigmoid* transferfunktion och utsätts sedan för elementvis multiplikation [22].

För en neuron av LSTM gäller för varje tidssteg t , där indata för tidssteget kallas x_t och utdata o_t , att en minnesbuss C överför minne från föregående tidssteg C_{t-1} till nästa tidssteg C_t . Minnesbussen passerar hela neuronen med endast ett fåtal mindre linjära interaktioner vilket gör det enkelt för information att bibehållas och överföras till senare tidssteg [22].



Figur 11: Visualisering av flödet för minnesbussen C_{t-1} till nästa tidssteg C_t . På vägen utsätts vektorn för två elementvisa operationer till följd av olika grindar [22].

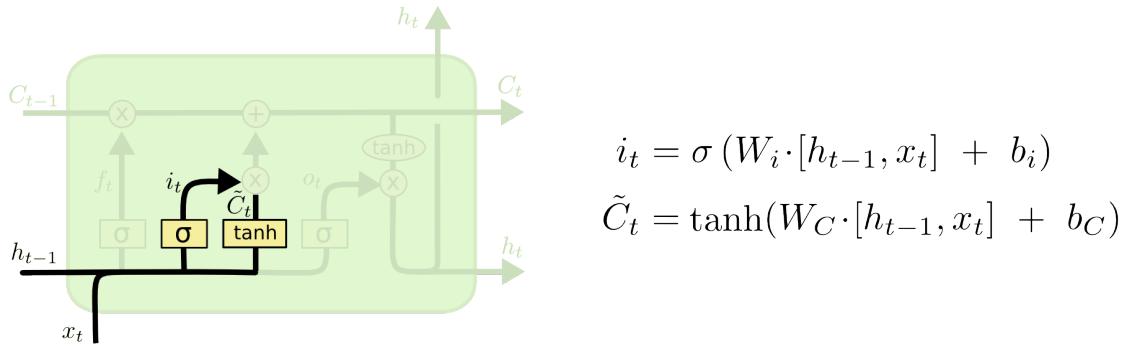
Det första steget i flödesschemat för ett tidssteg i ett LSTM är att bestämma om summan av indata x_t och föregående tidssteg h_{t-1} ska behållas och eventuellt överföras till minnesbussen. Den viktade summan för dessa vektorer ges ett värde mellan 0 och 1 i och med att den passerar *sigmoida* transferfunktionen i glömgrinden. Värden närmare 1 resulterar i att värdet behålls och förs vidare, medan värden närmare 0 resulterar i att värdet glöms bort och inte förs vidare i flödet. Detta första steg kan betecknas med hjälp av Ekvation 7 [22].



Figur 12: Visuell representation av första steget i ett flödesschema för ett LSTM. Indata passerar i detta första skede en glömgrind där det bestäms hur mycket av data som ska föras vidare till minnesbussen [22].

I det andra steget i flödesschemat bestäms vilken ny information som ska adderas till minnesbussen. Detta steg är uppdelat i två delar. Den första delen är ingångsgrinden där dataan passerar den *sigmoida* transfergrinden och bestäms vilken data som ska uppdateras. Andra delen följer transferfunktionen *tanh* där en vektor av kandiderande data \tilde{C}_t bestäms, alltså data som potentiellt ska adderas till minnesbussen. Andra delen kan tecknas som [22]:

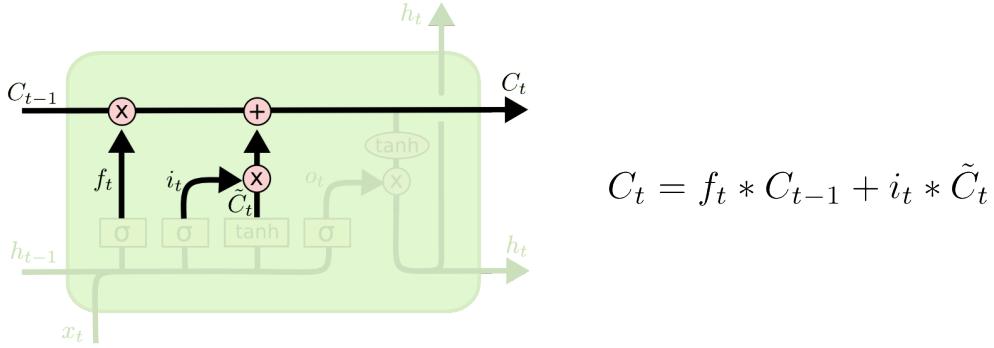
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (10)$$



Figur 13: Visuell representation av andra steget i flödesschemat. Här bestäms vilka värden av minnesbussen som ska uppdateras, samt vilka kandiderande värden som ska adderas till minnesbussen [22].

I det tredje steget uppdateras minnesbussen från steg C_{t-1} till det nuvarande tidssteget C_t . Detta sker genom att uppdatera vektorn med de värden som bestämdes i de två föregående flödesstegen. Först multipliceras C_{t-1} med de värden som ansågs värla att behålla i f_t . Sedan multipliceras ingångsgrinden i_t med de kandiderande värdena \tilde{C}_t . Denna vektor adderas sedan till minnesbussen. Detta steg kan tecknas som [22]:

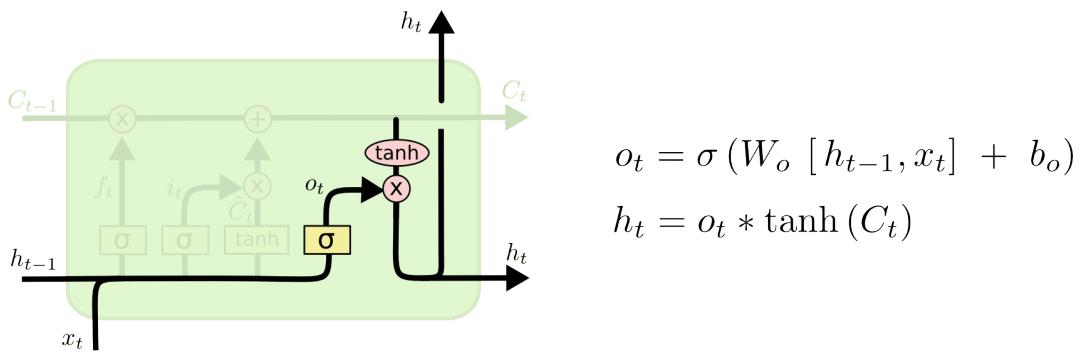
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (11)$$



Figur 14: Visuell representation av tredje steget i flödesschemat. Här uppdateras minnesbussen med de vektorer som framställts i de föregående stegen [22].

I det fjärde och sista steget beslutas om vilka värden som ska utmatas. Utdatan är inledningsvis baserat på den i minnesbussen C_t , som först passerar en transferfunktion i formen av \tanh . Denna vektor multipliceras sedan med indataen som passerat utgångsgrinden med en *sigmoid* funktion. Den slutgiltiga vektorn h_t överförs sedan till nästa neuron i nätverket samt matas ut som utdata, vilket kan tecknas som:[22].

$$h_t = o_t * \tanh(C_t) \quad (12)$$



Figur 15: Visuell representation av sista steget i flödesschemat. Här förbinds data från minnesbussen och ingångsgrinden för att sedan matas ut som utdata [22].

Samtliga föregående beskriva steg för flödesschemat i ett LSTM kan således beskrivas enligt följande sekvens av formler:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (13)$$

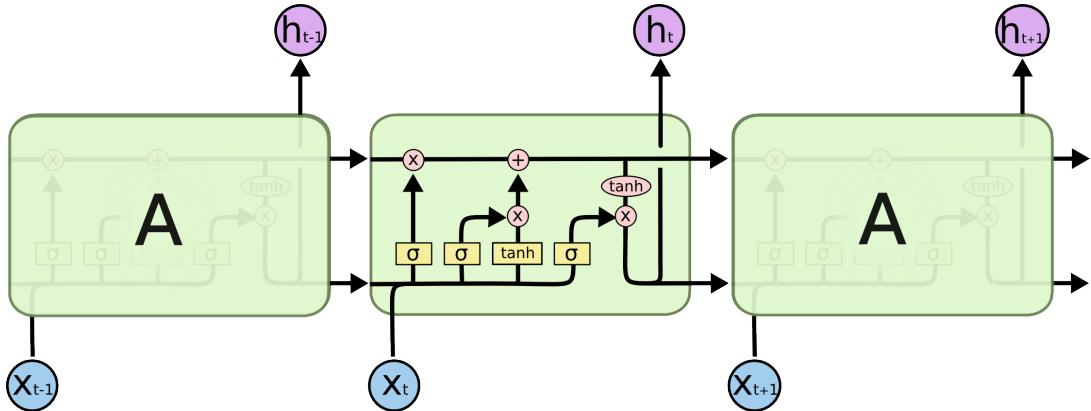
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (14)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (15)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (16)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (17)$$

$$h_t = o_t * \tanh(C_t) \quad (18)$$



Figur 16: Visuell representation av samtliga steg sammanlagda i flödesschemat för en neuron i ett LSTM [22].

2.8 Modellutveckling

För en modell av typen maskininlärning finns det vissa parametrar som inte går läras från själva träningsprocessen. Som tidigare nämnt är hyperparametrar variabler som bestäms innan träningsprocessen och påverkar exempelvis inlärningshastigheten eller längden av träningsprocessen. Dessa värden är och förblir konstanta och kan betraktas som oberoende. Till skillnad från hyperparametrar kallas de värden som varierar utefter hela träningsprocessen för inlärningsparametrar. Dessa består i huvudsak av olika vikter och bias och är de värden som modellen eftersträvar att anpassa så bra utifrån det givna datasetet som möjligt. Dessa värden utgörs från begynnelsen av till synes slumpvist valda värden och justeras sedan succesivt allt eftersom träningsprocessen pågår.

Inlärningsparametrarna justeras med hjälp av återkoppling utifrån vilken felmarginal varje given prediktion hade jämfört sanningsvärdena för prediktionen. Om en modell med en vikt W används för att prediktera värdet 10, men sanningsvärdet är 11 blir felmarginalen 10% och modellen kommer med hjälp av informationen justera vikten så att den nästa gång predikterar ett (förslagsvis) högre värde.

Begreppet epok menas en passerad genomgång (en iteration) av hela datasetet där modellen hela tiden strävar efter att förbättra sina inlärningsparametrar och således även resultatet. Antalet epoker är ett exempel på en hyperparameter och motsvarar oftast längden på träningen, antalet epoker kan variera mellan allt från 10 till 10 000 beroende på datasetets längd och komplexitet. Inlärningsparametrarna från slutet av varje epok överförs sedan till nästa epok där modellen får chans att förbättras igen [23].

För att begränsa beräkningskraften som krävs för träningen av en modell delas varje epok oftast in i mindre delar, så kallade satsstorlekar (engelska: batch size). Satsstorleken motsvarar antalet tidssteg för tidsserien som processas vid varje steg i en epok och utgörs av en hyperparameter. Om antalet epoker fastställs till 1000 och satsstorleken till 100 för ett tidsseriedataset med längden 500 tidssteg (exempelvis dagar), så kommer modellen tränas genom att processa 100 dagar åt gången i 5 steg ($500/100 = 5$) över 1000 iterationer (antalet epoker). Totalt sett kommer modellen träffa på en sats av data 5000 gånger. Modellen kommer alltså uppdatera sina vikter och bias (inlärningsparametrarna) 5000 gånger och försöka anpassa dessa så bra som möjligt utifrån det givna datasetet.

Det är emellertid inte alltid säkert att träningen för en modell genomgår samtliga angivna epoker. Detta sker oftast om modellen tränas för många epoker och drabbas av överanpassning alternativt om inga tydliga mönster i datan återfinns och modellen underpresterar. När inlärningsparametrarna slutar förbättras och inlärningsprocessen stagnerar eller till och med försämras avbryts oftast träningen för undvika förluster.

När modellen har tränats klart och anpassat en funktion för det givna datasetet kan den användas för att prediktera ny data (givet att datan liknar den i datasetet). Beroende på de olika parametrarna som ställts in och tränats kommer modellen prestera olika bra beroende på hur bra den är på att generalisera utifrån aldrig tidigare påträffad data [24].

2.9 Utvärderingsmått

När en modell har framtagits utefter ett dataset och används för att prediktera framtida data behövs metoder för att evaluera riktigheten hos det predikterade resultatet. Två sedvanliga metoder för denna evaluering är de två statistiska mättenheterna *medelkvadratfel* och *rotmedelkvadratfel* (engelska: mean squared error och root mean squared error).

Medelkvadratfel (häданefter förkortat MSE) mäter den genomsnittliga standardavvikelsen, alltså den genomsnittliga kvadratskillnaden mellan estimerade prediktionsvärdet och sanningsvärdet. MSE kan ses som en riskfunktion som korresponderar till de förväntade värdena av kvadratfelsförlusten. Resultatet för tillämpningar av MSE har alltid ett positivt värde och ju närmare noll värdet är desto noggrannare och således hög grad av riktighet har den predikterade datan. MSE följer det matematiska uttrycket

$$MSE = \frac{1}{n} \sum_{i=1}^n (x_i - y_i)^2 \quad (19)$$

där x_i är sanningsvärdet och y_i den predikterade datan för varje värde i .

Rotmedelkvadratfel (häданefter förkortat RMSE) är ett något strävare sätt att evaluera resultatet. RMSE följer samma matematiska funktion som MSE förutom att summan fås genom kvadratroten för uttrycket. På detta sätt blir värdet en aning större och enklare att åskådliggöra. RMSE följer det matematiska uttrycket enligt samma variabler [25]:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - y_i)^2} \quad (20)$$

3 Metod

Detta avsnitt innehåller genomgående förklaringar av den metodik som används för att framställa resultatet. Först kommenteras valet av dataset med tillhörande beskrivningar av de variabler som inkluderats. Andra delen fokuserar på valet av programspråk och de ramverk som utnyttjats. Sist utförs en omfattande genomgång av koden som står till grund för modellen.

3.1 Datainsamling

För att få ett så kvalitativt resultat som möjligt krävs en gedigen samling data som står till grund för datasettet. Den största mängden av datan som används har hämtats från en publik databas från *Our World in Data* (förkortat OWID). OWID är en vetenskaplig organisation som samlar och publicerar statistik inom en stor mängd olika ämnesområden. Datatan från OWID rörande statistik om covid-19 uppdateras kontinuerligt varje dag och finns att tillgå i filformatet *kommaseparerade värden* (.csv).

Datasettet, vid namn *owid-covid-data.csv* har storleken 173488 rader över 69 kolumner (vid datumet 2022-04-02) och innehåller statistik för 241 olika länder. Huvudsakliga källor för datan är baserade på bland andra *Center for Systems Science and Engineering, Förenta nationerna* samt *Folkhälsomyndigheten* för de data som specifikt rör Sverige. Tabell 1 beskriver vilken ursprunglig källa som används, uppdateringsfrekvensen och mängden omfang för varje kategori av data som används av OWID.

Eftersom datasettet från OWID saknar omfattande statistik kring olika virusvarianter och dess spridning har ytterligare en källa av data funnit sig nödvändig. För att tillgå statistik om virusvarianter har således data från *Europeiska smittskyddsmyndigheten* (engelska: *European Centre for Disease Prevention and Control* förkortat ECDC) används. Datasettet innehåller respektive virusvariants procentuella andel bekräftade fall med statistik för varje vecka och land. ECDC baserar sin statistik på forskningsstationen GISAID, vars upphovsrättslag hindrar detta arbetet från att publicera fullständig statistik då formellt tillstånd saknas. Ett urval för vecka 21 år 2021 i Sverige kan däremot se ut som representerat i Tabell 2.

Kategori	Källa	Uppdateringsfrekvens	Antal länder
Vaccinationer	Officiell data från Our World in Data	Dagligen	218
Tester och positivet	Officiell data från Our World in Data	Veckovis	174
Sjukhus och IVA-platser	Officiell data från Our World in Data	Dagligen	47
Antal bekräftade fall	JHU CSSE covid-19 data	Dagligen	216
Antal bekräftade dödsfall	JHU CSSE covid-19 data	Dagligen	216
Basal reproduktionskvot	Arroyo-Marioli F m.fl.	Dagligen	190
Restriktioner	Oxford covid-19 restriktionsmätare	Dagligen	187
Övriga variabler	Internationella organisationer (FN, Världsbanken, OECD mm.)	Konstanta	241

Tabell 1: Tabell som förklarar hur OWID sammantäcker sin kollektion av data.

År-vecka	Virusvariant	Procentuell andel
2021-21	B.1.1.7	91.9
2021-21	B.1.1.7+E484K	1.6
2021-21	B.1.351	1.9
2021-21	B.1.617.2	3.4
2021-21	P.1	0.3
2021-21	Other	1

Tabell 2: Procentuell andel bekräftade fall för för olika virusvarianter vecka 21 i Sverige (endast virusvarianter med en andel större än 0).

De två olika dataseten sammansätts sedan med hjälp av mjukvarubiblioteket *pandas* till ett nytt dataset. För att undvika onödig data tas de kolumner med konstanta värden bort, då de inte tillför någon förändring utan enbart upptar beräkningskraft i programmet. Det nya datasetet, som i koden namnges ”*df*”, har formen 741 rader över 65 kolumner och innehåller bara statistik för Sverige. Ett urval av datasetet visualiseras i Tabell 3 för de 5 första respektive 5 sista raderna för 6 olika kolumner. Samtliga kolumner i datasetet presenteras även i Bilaga 1.

	new_cases	total_cases	new_cases	...	Other	P.1	P.3
date			_smoothed	...			
2020-02-01	1.0	1.0	0.000	...	0.0	0.0	0.0
2020-02-02	0.0	1.0	0.000	...	0.0	0.0	0.0
2020-02-03	0.0	1.0	0.000	...	0.0	0.0	0.0
2020-02-04	0.0	1.0	0.000	...	0.0	0.0	0.0
2020-02-05	0.0	1.0	0.000	...	0.0	0.0	0.0
...
2022-02-05	0.0	2287785.0	31047.000	...	0.0	0.0	0.0
2022-02-06	0.0	2287785.0	31047.000	...	0.0	0.0	0.0
2022-02-07	0.0	2287785.0	31047.000	...	0.0	0.0	0.0
2022-02-08	66670.0	2354455.0	24363.571	...	0.0	0.0	0.0
2022-02-09	18182.0	2372637.0	21360.286	...	0.0	0.0	0.0

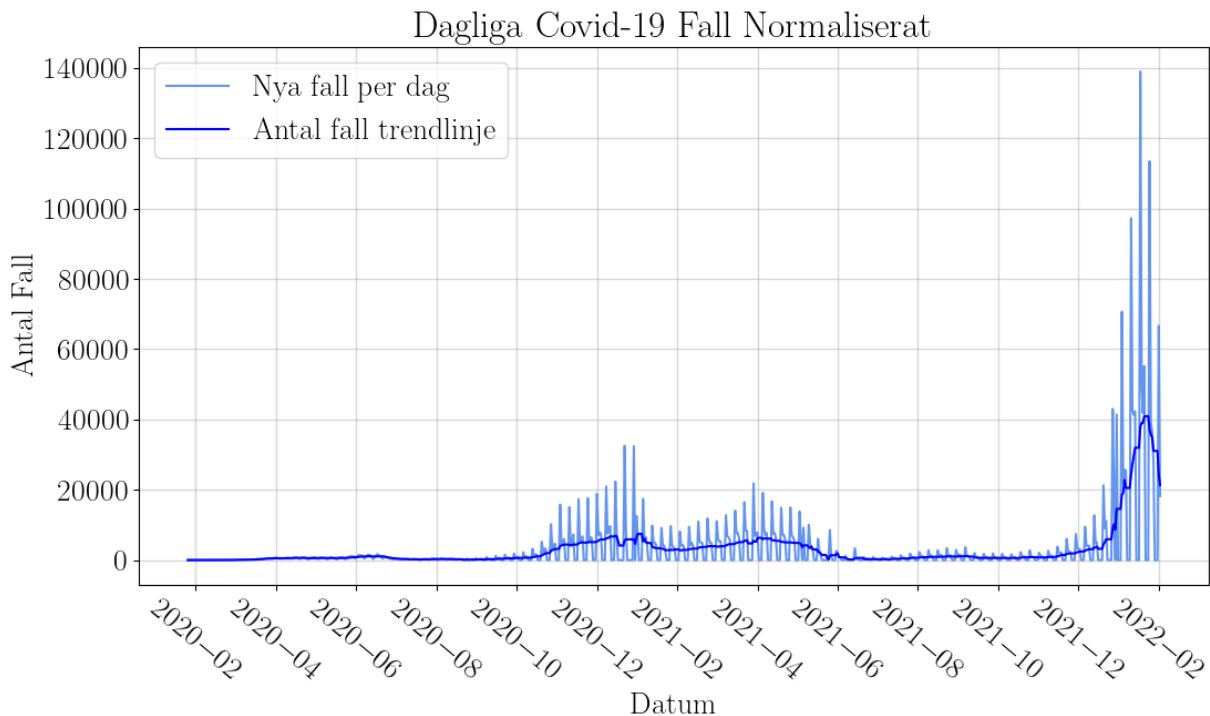
[741 rows x 65 columns]

Tabell 3: Representativt urval av datasetet.

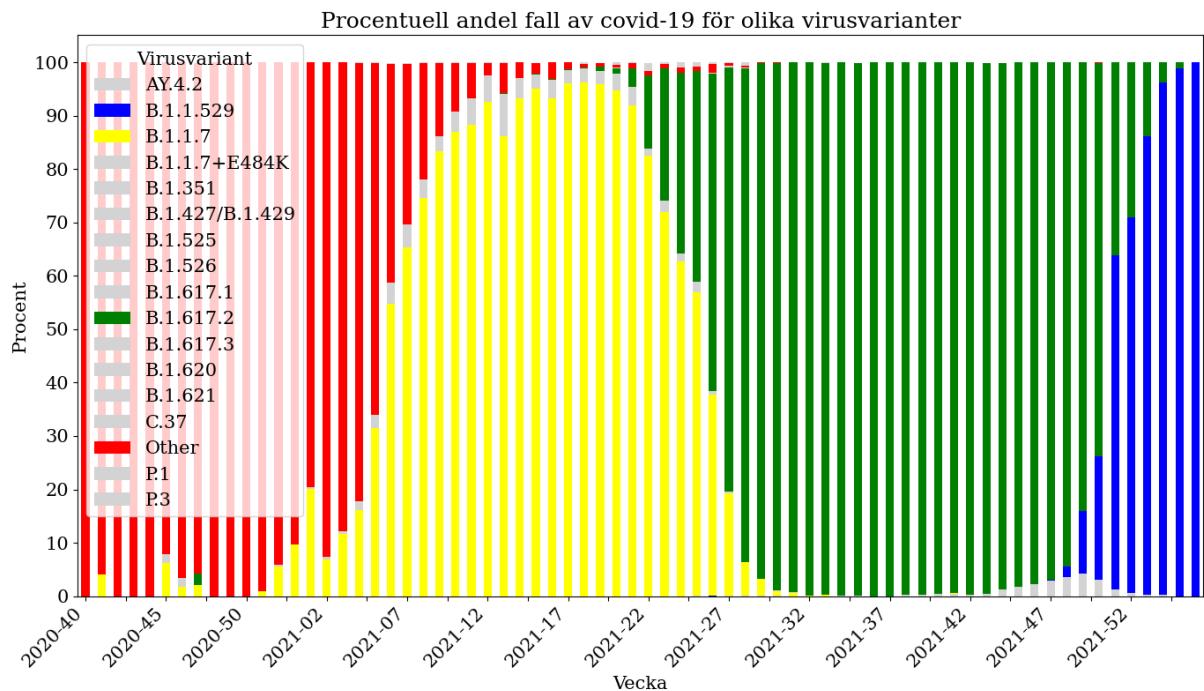
Den variabel som är målbilden för modellen är kolumnen för antalet dagliga bekräftade nya fall av covid-19, namngiven *new_cases* i datasetet. En graf för denna tidsserien kan visualiseras såsom i nedan stående figur tillsammans med en trendlinje, namngiven *new_cases_smoothed*.

För att ytterligare åskådliggöra vilka olika faktorer som har en inverkan på antalet dagliga fall kan en korrelationsundersökning genomföras, där de variabler med högst påverkan framtonas. En sådan undersökning återfinns i Tabell 4.

Även ytterligare variabler såsom de olika virusvarianternas procentuella fördelning kan visualiseras, denna gången med hjälp av ett kumulativt stolpdiagram. För tydlighetens skull har enbart de varianter med en märkbar påverkan betonats med olika färger även om de övriga varianterna är inkluderade i datasetet. Denna graf kan hjälpa till att förstå varför pandemins olika våguppgångar uppstår och återges i Figur 18.



Figur 17: Graf visande antalet nya fall per dag respektive en trendlinje för tidsserien.



Figur 18: Kumulativt stolpdiagram som illustrerar den procentuella fördelningen av antalet fall per dag för de olika virusvarianterna. Notera att enbart de virusvarianter med en substansiell andel har färglagts för tydlighetens skull.

Variabel	Korrelationsvärde
new_cases	1
positive_rate	0.573198
new_tests	0.465821
total_vaccinations	0.252596
hosp_patients	0.243249
people_fully_vaccinated	0.21891
total_boosters	0.207158
total_deaths	0.201871
new_deaths	0.188712
icu_patients	0.054632

Tabell 4: Korrelationsvärde mellan olika kolumner i datasetet utefter kategori och antalet fall per dag. Notera att samtliga variabler ej är representerade i tabellen utan ett selektivt urval har gjorts utifrån intresse. Däremot är de flesta olika kategorier av kolumner representerade i urvalet

Som kan ses i ovan tabell är det andelen positiva fall för tester som har störst korrelationsindex för antalet fall per dag, även antalet nya tester per dag har en hög betydelse för antalet nya fall av covid-19. Som kan ses korrelerar antalet nya fall endast 5 procent med antalet patienter inlagda på IVA (Intensivvårdsavdelning). Genom att studera ovan tabell kan parametrar med låg korrelation selektivt tas bort från datasetet då de inte tillför alternativt försvarar för modellen. Till exempel tas kolumner med konstant värde genom hela datasetet bort då de har en negativ korrelation med antalet fall.

3.2 Genomgång av källkod

Följande sektion kommer innehålla förklaringar och kommentarer till programkoden som har använts för att framställa resultaten för de olika undersökningarna. Koden har delats upp i stycken efter olika områden, men förklaras fortfarande i en linjärt konsekvent ordning med angivna radnummer. Varje sådant stycke anges som *Källkod* följt av det nummer som stycket antar i ordningen av förklaringar. Koden följer sedanliga principer för indrag och variabelnamn och är skrivet i programspråket Python (version 3.10.4) med tillhörande färgbehandling för programspecifik syntax.

```
1 import pandas as pd
2 import numpy as np
3 import keras
4 import tensorflow as tf
5 import matplotlib.pyplot as plt
6 from sklearn.preprocessing import StandardScaler
7 from numpy import split
8 from numpy import array
9 import seaborn as sns
10 import dataframe_image as dfi
11 import matplotlib.dates as mdates
```

Källkod 1: Importering av mjukvarubibliotek.

Ovanstående stycke importerar de essentiella programbiblioteken som krävs för att exekvera och kompilera programkoden. De bibliotek som specifikt hanterar processen för djupinlärda artificiell intelligens är *tensorflow* och *keras*.

```
12 df = pd.read_csv('owid-covid-data.csv')
13 df['date'] = pd.to_datetime(df['date'])
14 df.set_axis(df['date'], inplace=True)
15 df.fillna(method='ffill', inplace=True)
16
17 df = df[:462]
18 df_date = df['date']
19
20 print(df)
```

Källkod 2: Importering av datasetet.

Ovanstående stycke läser in datasetet i formen av en csv-fil och sparar det i variabeln *df* (engelska: dataframme). Följande rad ändrar formatet på datumet till ett tidformat som

går att arbeta med utan förlorad beräkningskraft. Rad 15 ersätter alla tomma värden i *df* med det nästkommande värdet för att undvika körfel såsom division med 0 vid exekvering. Av ändamålsenliga skäl i enlighet med arbetets avgränsningar förkortas datasetet till en viss dag, i detta exemplet dag nummer 462 i tidsserien. Sist skrivs ett urval av datasetet ut, vilket illustreras genom Tabell 3.

```

21 from sklearn.preprocessing import MinMaxScaler
22
23 df = df.drop(axis=1, columns=['date'])
24 Y = df['new_cases'].values
25 Y = Y.reshape((Y.shape[0], 1))
26
27 scaler = MinMaxScaler()
28
29 scaler.fit(Y)
30 Y = scaler.transform(Y)

```

Källkod 3: Applicering av skalär.

För en så effektiv exekvering som möjligt bör värden som passerar en modell vara förhållandvis små, detta uppnås enklast genom att anpassa och applicera en lokal skalär på datasetet. I stycket ovan importeras således just en sådan sorts skalär och sparas i variabeln *scaler*. Rad 25 till och med 30 anpassar skalären till *Y*, som i detta fallet är målbildsvariabeln för antalet fall per dag. Resultatet blir att *Y*-värdena normaliseras och anpassas till ett värde mellan 0 och 1.

```

31 def plot(date_format, intervals, x_axis, y_axis, title,
32         x_label, y_label, legend_1, formater=True, color=None,
33         line_2=False, x_axis_2=None, y_axis_2=None,
34         legend_2=None, color_2="orange"):
35
36     plt.rc('text', usetex=True)
37     plt.rc('font', family='serif')
38
39     fig = plt.figure (figsize=(12, 6))
40     plt.grid(color='gray', linewidth=1, alpha=0.3)
41
42     if formater:
43         formatter = mdates.DateFormatter(date_format)
44         plt.gca().xaxis.set_major_formatter(formatter)
45         plt.gca().xaxis.set_major_locator(mdates.DayLocator(
46                                         interval=intervals))
47         ...

```

```

48     ...
49     plt.xticks(rotation=-40)
50
51     plt.plot(x_axis, y_axis, color=color)
52     if line_2:
53         plt.plot(x_axis_2, y_axis_2, color=color_2)
54
55     plt.title(title)
56     plt.xlabel(x_label)
57     plt.ylabel(y_label)
58     plt.legend([legend_1, legend_2])
59
60     plt.show()

```

Källkod 4: Definiera allmän funktion för framställning av grafer som sedan kan anropas vid behov.

I ovanstående stycke definieras en allmän funktion som fölaktligen anropas för att framställa grafer av olika slag. För varje enskild graf åstadkoms det önskade resultatet genom att ange specifika värden för tidformat, tidintervall, namnen på koordinataxlarna och så vidare.

```

61 plot('%Y-%m', 61, df_date, Y,
62      'Dagliga Covid-19 Fall Normaliserat', 'Datum',
63      'Antal Fall', 'Daglig trend')
64
65 n_input = 14
66 n_out = 14
67 n_features = len(df.columns)
68
69 train, test = df[:-n_input], df[-n_input * 2:] # 75% och 25%
70 print(train.shape, test.shape)

```

Källkod 5: Fastställa konstanta värden och dela datasetet i en träningsdel och en valideringsdel.

Ovanstående stycke inleds med att anropa funktionen som definierades i föregående stycke för att framställa en illustrativ graf. Följande rader preciserar två konstanter för antalet dagar som ska användas till indata respektive utdata för modellen. Sist åsidosätts även datan till den träningsdel som modellen tränas utefter och en valideringsdel som modellen ska prövas utifrån. Både tränings- och valideringsdelen utgörs av en tvådimensionell form enligt [tidsseriens längd, antalet variabler]. Formen i detta exemplet för respektive del är [48, 56] för träningen och [28, 56] för valideringen.

```

71  scaler.fit(train)
72  train = scaler.transform(train)
73  test = scaler.transform(test)

```

Källkod 6: Anpassa skalären på tränings- och valideringsdelen.

Här anpassas skalären på nytt till träningsdelen och skalar ner både tränings- och valideringsdelen och sparar dessa i varsin variabel.

```

74  def split_sequence(sequence, n_input, n_out):
75      X, y = list(), list()
76      in_start = 0
77
78      sequence = sequence.reshape((sequence.shape[0],
79                                    sequence.shape[1]))
80      # step over the entire history one time step at a time
81      for _ in range(len(sequence) + 1):
82          # define the end of the input sequence
83          in_end = in_start + n_input
84          out_end = in_end + n_out
85          # ensure we have enough data for this instance
86
87          if out_end < len(sequence) + 1:
88              x_input = sequence[in_start:in_end, :]
89              X.append(x_input)
90              y.append(sequence[in_end:out_end, 0])
91          # move along one time step
92          in_start += 1
93
94      return array(X), array(y)

```

Källkod 7: Definiera funktion som omvandlar datasetet till ett för vägledd inlärning.

Innan en dataserie kan användas till en modell måste den prepareras. Målet är att den djupinlärda AI:n ska lära sig från en funktion som kartlägger en observationssekvens av indata och sedan presenterar en sekvens av utdata. För att möjliggöra detta måste observationssekvensen först transformeras till ett flertal praktiska exempel från vilket modellen kan lära sig utav, en så kallad vägledd inlärning.

Betrakta en given sekvens [10, 20, 30, 40, 50, 60, 70, 80, 90]. Fördelning av sekvensen till flera indata-utdata sekvenser kallas för ett sampel (engelska: sample).

Ett sampel med 3 indatastidssteg och 3 utdatastidssteg kan beskrivas med beteckningarna x för indata och y för utdata. Sampeln kan till exempel anta följande form utifrån den givna sekvensen:

x	y
10, 20, 30,	40, 50, 60
20, 30, 40,	50, 60, 70
30, 40, 50,	60, 70, 80 ...

där varje sekvens indata x har en motsvarande sekvens med de värden som modellen ska prediktera och jämförs med. För första steget i tidsserien mottar modellen indata-sekvensen [10, 20, 30] och ska sedan prediktera tre utdata-värden, i detta fall sanningsvärderna [40, 50, 60] för y . Funktionen som beskriven ovan och definierad i Källkod 7 implementerar detta utförande och delar en given sekvens till flera sampel med ett specifikt antal förbestämda indata-tidssteg och utdata-tidssteg.

```

94 train_x, train_y = split_sequence(train, n_input, n_out)
95 test_x, test_y = split_sequence(test, n_input, n_out)
96
97 print(train_x.shape, train_y.shape, test_x.shape,
98       test_y.shape)
```

Källkod 8: Tillämpa föregående definierad funktion för de olika delarna av datasetet.

Ovanstående stycke tillämpar föregående funktionen och delar således både träningsdelen och valideringsdelen i respektive indata-utdatasekvens. Formen⁵ för delarna är [421, 14, 56] för indatasekvensen och [421, 14] för utdatasekvensen på träningen samt [1, 14, 56] för indatasekvensen och [1, 14] för utdatasekvensen på valideringen där formen utgörs av [indatans storlek, antal tidssteg, antal variabler]. Indatan för exempelvis träningen tar in 421 dagar som indata, delar upp det till flera sampel med 14 tidssteg vardera och upprepar detta för varje av de 56 olika variablerna. Eftersom det är antalet fall som söks i utdata är det bara denna variabel som utdata innehåller.

⁵Med form menas vilken grad av dimensioner varje vektor utgör. En vektor av tredje graden har till exempel formen [x, y, z] med värden i tre olika dimensioner, även känt som 3D.

```

99  train_y = train_y.reshape((train_y.shape[0],
100                                train_y.shape[1], 1))
101 test_y = test_y.reshape((test_y.shape[0], test_y.shape[1], 1))
102 print(train_x.shape, train_y.shape, test_x.shape,
103       test_y.shape)

```

Källkod 9: Omforma delarna från föregående stycke till att passa enligt formen för indata till en cell av LSTM.

Vid konstruktionen av en artificiell intelligens med Keras och TensorFlow måste sampeln utgöras av en vektor med tredimensionell form, men i föregående stycke omformas utdatasekvenserna i tränings- och valideringssamplen till enbart två dimensioner. För att åtgärda detta omformas de tvådimensionella vektorerna till en tredimensionell vektor med längden ett. Den nya formen på utdatasekvenserna blir således [421, 14, 1] för träningen och [1, 14, 1] för valideringen.

```

104 from keras.models import Sequential
105 from keras.layers import LSTM, Dense, PReLU, TimeDistributed,
106 RepeatVector, Dropout
107 from tensorflow.keras.optimizers import Adam
108
109 num_epochs, batch_size = 1000, 64

```

Källkod 10: Importera nödvändiga funktioner och bestämma konstanter.

Importering av nödvändiga funktioner från Keras och TensorFlow samt bestämning av konstanter för antalet epoker och satsstorlek.

```

110 model = Sequential()
111
112 mc = tf.keras.callbacks.ModelCheckpoint(
113     'best_model_multi_v2.h5', monitor='val_loss', mode='min')
114 callback = tf.keras.callbacks.EarlyStopping(
115     monitor="val_loss", min_delta=0, verbose=1, patience=200,
116     mode="min", restore_best_weights=True)

```

Källkod 11: Fastställ funktioner som övervakar och reglerar träningsfasen.

För att förhindra överanpassning samt korta av träningstiden för modellen implementeras två olika optimeringar. Den ena sparar det bästa värdet för utvärderingsmåttet (lägsta MSE) under hela träningsfasen och den andra avslutar träningen när ingen märkvärd förbättring sker, alternativt om modellens prestanda försämras.

```

117 model.add(LSTM(64, activation='PReLU', input_shape=(n_input,
118                                         train_x.shape[2])))
119 model.add(Dropout(0.2))
120 model.add(RepeatVector(n_out))
121 model.add(LSTM(128, activation='PReLU',
122                 return_sequences=True))
123 model.add(LSTM(64, activation='PReLU', return_sequences=True))
124 model.add(Dropout(0.2))
125 model.add(TimeDistributed(Dense(32, activation='PReLU')))
126 model.add(TimeDistributed(Dense(1)))

```

Källkod 12: Fastställning av neuronnätet för modellen.

I ovanstående stycke anges hur modellen för den artificiella intelligensen ska tränas. Den första raden informerar vilken typ av cell det rör sig om, antal neuroner som ska användas för det cellagret, transferfunktionen, storleken på en sampel och sist antalet variabler. Raden därefter bestämmer hur många element i minnesbussen som ska passera glömgrinden, alltså glömmas bort. Detta görs för att undvika överanpassning och används för att reglera inlärningstakten. *RepeatVector*-funktionen gör precis som namnet indikerar, repeterar den inkommande indatan ett visst antal gånger. Detta betyder i praktiken att träningsfasen processar samma indata n antal gånger innan den passerar till nästa lager. Indatan passerar sedermera två nya lager av typen långa korttidsminnesnät och därefter ytterligare en glömgrind. *TimeDistributed*-funktionerna därpå applicerar ett ”tätt” lager (*Dense*) för varje sampel den mottar. Det tätta lagret är djupt forbundet med föregående lager, vilket innebär att cellerna i lagret ansluter sig till varje neuron i föregående lager och återkopplar passerande data. En översikt av neuronlagret som definierat ovan kan ges om följande kod körs:

```
print(model.summary())
```

Vilket ger följande utskrift:

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	33344
dropout (Dropout)	(None, 64)	0
repeat_vector (RepeatVector (None, 14, 64)		0
)		
...		

...		
lstm_1 (LSTM)	(None, 14, 128)	98944
lstm_2 (LSTM)	(None, 14, 64)	49472
dropout_1 (Dropout)	(None, 14, 64)	0
time_distributed (TimeDistr ibuted)	(None, 14, 32)	2112
time_distributed_1 (TimeDis tributed)	(None, 14, 1)	33

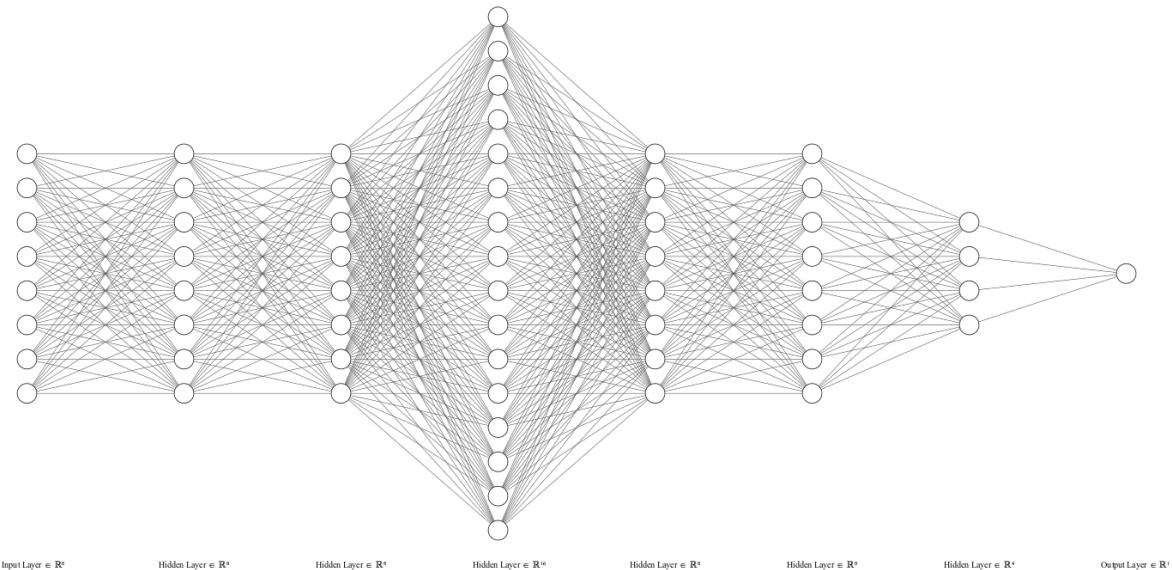
Total params: 183,905

Trainable params: 183,905

Non-trainable params: 0

Neuronnätet för modellen har alltså 183905 möjliga parametrar som tränas och sedan används vid prediktioner.

Neuronnätet kan även visualiseras grafiskt enligt figuren nedan. Notera dock att antalet neuroner för varje lager i nätverket har dividerats med 8 (förutom det sista utdatalagret som antar formen 1) för att möjlighetens skull att kunna se. Ett lager med storleken 8 neuroner har således egentligen 64 neuroner.



Figur 19: Bildlig visualisering av neuronnätet som modellen utgör.

```

127 lr = 1e-3
128 optimizer = Adam(learning_rate=lr, decay=lr/num_epochs)
129 model.compile(optimizer=optimizer,
130                 loss=tf.keras.losses.Huber())
131
132 model.fit(train_x, train_y, epochs=num_epochs,
133 batch_size=batch_size, validation_data=(test_x, test_y),
134         verbose=1, callbacks=[callback, mc])
135
136 from keras.models import load_model
137 saved_model = load_model('best_model_multi_v2.h5')

```

Källkod 13: Modellens tränas.

Funktionsoptimeraren effektiviseras i det ovanstående stycket och i *model.fit*-funktionen tränas modellen. Modelleringen sker på så sätt att för varje epok ackommoderar neuronerna sig utefter ett lämpligt neuronnät baserat på träningssekvenserna som därefter prövas på valideringssekvenserna. Till sist sparas den bästa modellen som anpassades under träningsfasen.

```

138 loss_per_epoch = model.history.history['loss']
139 val_loss_per_epoch = model.history.history['val_loss']
140
141 plot(None, None, range(len(loss_per_epoch)), loss_per_epoch,
142 'Medelkvadratfelet For Varje Epok', 'Epok',
143 'Medelkvadratfel', 'Modellforlust', formater=False,
144 line_2=True, x_axis_2=range(len(val_loss_per_epoch)),
145 y_axis_2=val_loss_per_epoch, legend_2='Valideringsforlust')
146
147
148 prediction = saved_model.predict(test_x)
149 prediction = prediction.reshape(
150 prediction.shape[0] * prediction.shape[1],
151 prediction.shape[2]))
152 prediction = prediction[:, 0]
153
154 for i in range(len(prediction)):
155     if prediction[i] <= 0:
156         prediction[i] = 0
157
158 test_y = test_y.reshape(test_y.shape[0] * test_y.shape[1])

```

Källkod 14: Modellen utför en prediktion för valideringsdatan.

I ovanstående stycke nyttjas TensorFlows inbyggda predikteringsfunktion genom att framlägga en indatasekvens. I detta fall används indatasekvensen för valideringen.

Eftersom det endast är antalet fall som söks i utdata, väljs enbart denna variabel. I datasetet som saknar daglig uppdatering av antalet fall, inklusive datasetet som används för detta arbetet, där vissa dagar i veckan har 0 nya fall (på grund av bristfällig rapportering) tenderar modellen att prediktera värdet under 0. Med detta i åtanke bestäms därför alla negativa värden som 0.

```

159 from sklearn.metrics import mean_squared_error
160
161 print(f'MSE: {round(mean_squared_error(test_y,
162                                     prediction), 7)}')
163 print(f'RMSE: {round(mean_squared_error(test_y,
164                                     prediction, squared=False), 7)}')
```

Källkod 15: Modellens prediktion utvärderas.

MSE och RMSE-funktionerna tillämpas i ovanstående stycke och jämför den predikterade datasekvensen mot respektive sanningsvärdet.

```

165 new_cases = df['new_cases'].values
166
167 new_cases = new_cases.reshape((new_cases.shape[0], 1))
168 new_cases_copies = np.repeat(new_cases, train_x.shape[2],
169                               axis=-1)
170 new_cases_normalized = scaler.transform(
171                               new_cases_copies)[:, 0]
172
173 date_test = df_date[-n_input:]
174
175 prediction_date = []
176 for i in range(n_input):
177     prediction_date.append(date_test[i])
```

Källkod 16: Omforma data till senare och spara prediktionsdatum för utskrift.

För att passera genom skalären måste antalet fall i datasetet (som är endimensionellt) vara av samma dimension som vad skalären först anpassades till. Stycket ovan uträttar detta samt skapar en lista för prediktionsdatum till framtidshypotesen.

```

178 plot("%Y-%m", 61, df_date, new_cases_normalized,
179 'Normaliserad J mf relse av Sanningsv rde och Hypotes',
180 'Datum', 'Antal Fall', 'Sanningsv rde', line_2=True,
181 x_axis_2=prediction_date, y_axis_2=prediction,
182 legend_2="Hypotes")
183
184 plot("%Y-%m-%d", 1, prediction_date, test_y[-n_input:],
185 'Normaliserad J mf relse Av Sanningsv rde Och Hypotes',
186 'Datum', 'Antal Fall', 'Sanningsv rde', line_2=True,
187 x_axis_2=prediction_date, y_axis_2=prediction,
188 legend_2="Hypotes")
189
190
191 prediction_list = df[-n_input:].values
192 x = scaler.transform(prediction_list)
193 x = x.reshape((1, prediction_list.shape[0],
194                 prediction_list.shape[1]))
195
196 forecast = saved_model.predict(x)
197 forecast = forecast.reshape(
198     forecast.shape[0] * forecast.shape[1], forecast.shape[2]))
199 forecast_copies = np.repeat(forecast, train.shape[1], axis=-1)
200 forecast = scaler.inverse_transform(forecast_copies)[:, 0]

```

Källkod 17: Prediktera värden för framtida datum.

Stycket ovan samlar in de 14 sista tidsstegen från datasetet, skalar ner dem till värden mellan 0 och 1 samt använder den sparade modellen för att prediktera antalet framtida fall med TensorFlows inbyggda funktion. När detta är gjort skalias värdena om till prediktionens slutgiltiga värden.

```

201 def predict_dates():
202     last_date = df_date.values[-1]
203     prediction_dates = pd.date_range(
204         last_date, periods=n_out + 1).tolist()
205     prediction_dates.pop(0)
206     return prediction_dates
207
208
209 forecast_dates = predict_dates()
210
211 for i in range(len(forecast)):
212     forecast[i] = abs(forecast[i])

```

Källkod 18: Förbereda framtida prediktioner.

En grafisk illustrering av framtidshypotesen kräver en funktion som skapar framtida datum. Stycket ovan utför detta och alla negativa värden från predikteringen bestäms återigen till 0.

```

213 plot("%Y-%m", 61, df_date, df['new_cases'],
214 'Framtida Prediktioner Med LSTM', 'Datum',
215 'Antal Fall', 'Sanningsvärdet', color_2="red",
216 line_2=True, x_axis_2=forecast_dates,
217 y_axis_2=forecast, legend_2="Framtidshypotes")
218
219 plot("%Y-%m-%d", 1, forecast_dates, forecast,
220 'Framtida Prediktioner Med LSTM', 'Datum',
221 'Antal Fall', 'Framtidshypotes', color='red')
222
223 print(f'7-Dagars Medelvärdet: {abs(round(
224 sum(forecast) / n_out))} Antal Fall')
```

Källkod 19: Framställ grafer för de framtida prediktionerna.

Sist framställs grafer för värdena av de framtida prediktionerna och programmet avslutas.

3.3 Justering av hyperparametrar

Justering av de hyperparametrar som preciserats allt efter i föregående sektion utfördes genom ”försök och misstag” (engelska: trial and error), alltså genom att pröva sig fram och se vilka värden som åstadkommer ett gynnsamt resultat. Några exempel på de variabler, som ofta är konstanter, som justeras utefter arbetsprocessen är *num_epochs*, *batch_size*, *lr* (learning rate), och storleken (antalet neuroner) för de olika lagrena i neuronnätet. Det finns inga förutbestämda regler för hur, vilka eller hur mycket de olika hyperparametrarna ska justeras för att öka riktigheten på resultatet, vilket är just svårigheten med detta moment. För varje exekvering av programmet har valet av hyperparametrarnas använda värden evaluerats med hjälp av utvärderingsmåttet. Återgavs ett sämre resultat än föregående exekvering kan det vara ett tecken på att någon variabel har justerats åt fel håll och behöver åtgärdas eller återgå till det tidigare värdet. Det är genom att finjustera hyperparametrarna som modellen åstadkommer ett bättre resultat och är den del som tagit längst tid under arbetsprocessen. Det slutgiltiga resultatet för varje parameter som användes för modellen presenteras i Tabell 5.

Hypervariabel	Förklaring	Slutgiltigt värde
split_date	Den dag i tidsserien som modellen ska tränas till och med.	Se olika värden för avsnittet resultat
n_input	Antalet dagar av data som modellen matas in med varje tidssteg	14
n_input	Antalet framtida dagar som modellen predikterar varje tidssteg	14
num_epochs	Antalet epoker som modellen tränas för	Se olika värden för avsnittet resultat
batch_size	Storleken som tidsserien delas upp i varje epok	64
1_layer_neurons	Antalet neuroner för första lagret i neuronnätet	64
2_layer_neurons	Antalet neuroner för andra lagret i neuronnätet	128
3_layer_neurons	Antalet neuroner för tredje lagret i neuronnätet	64
dense_layer_neurons	Antalet neuroner för det täta lagret i neuronnätet	32
dropout	Andelen data som passerar glömgrinden	0.2
lr (learning rate)	Mått på hur snabbt modellen ska lära sig	1.00E-03

Tabell 5: Slutgiltiga värden för hyperparameterjusteringen med förklaringar för varje variabel.

3.3.1 Val av datum

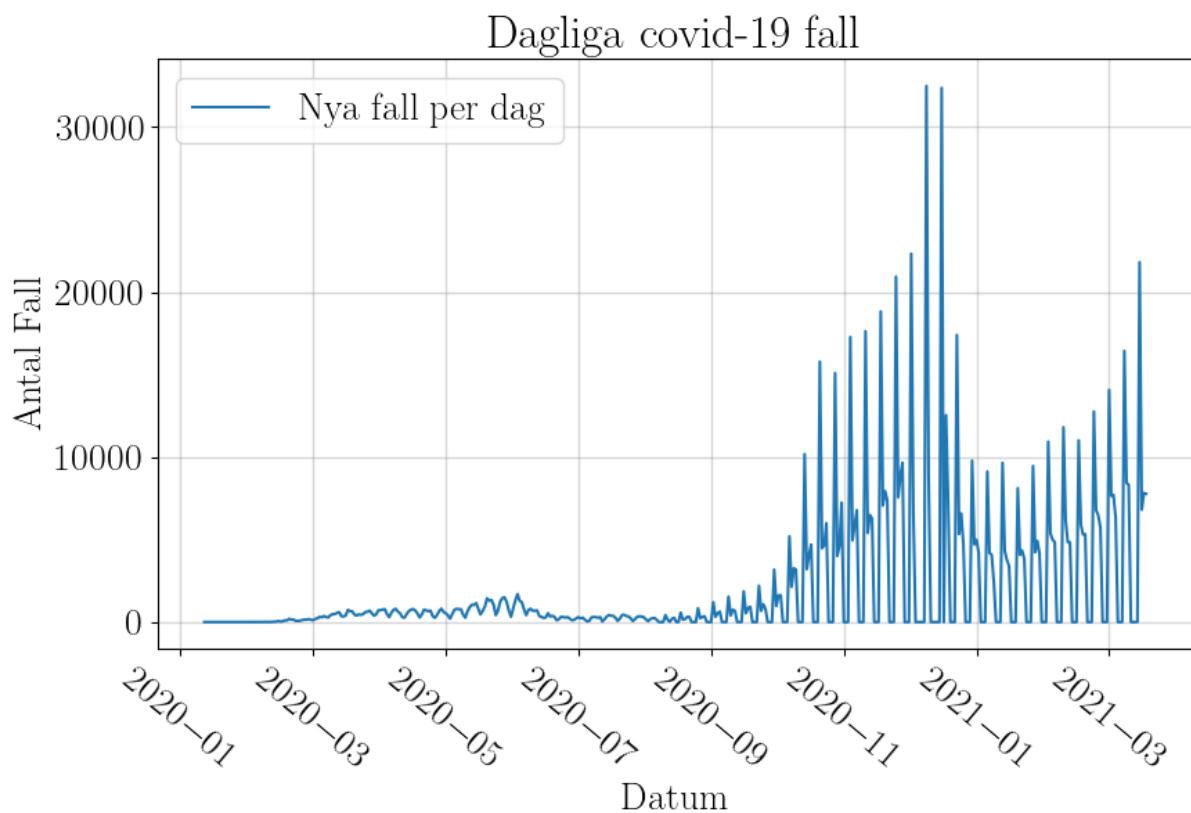
För att få en så rättvisande bild som möjligt av modellens förmåga och omfang krävs att ett flertalet olika situationer representeras i resultatet för undersökningen. Därför har flera olika datum för tidsserien antagits och presenterats för de olika datumen i resultatdelen. Olika datum har således valts för de olika ”vågerna” av covid-19 under pandemien. Däremot har inget datum före den andra vågens nedgång (2021-01), eller för vågen uppkommen av omikronvarianten tagits hänsyn till (där det då hittills högsta anträffade värdet av antalet fall ökade från 30000 till 140000 inom loppet av veckor). Detta beror på grund av bristen på tillräckligt utförlig data för att om möjligens lyckas modellera en funktion med hög träffsäkerhet, varför ett datum av den karaktären ej är av intresse för slutsatsen. Med det sagt är det inte omöjligt att tillräcklig data ges i framtiden för att modellera en funktion för omikronvarianten.

4 Resultat

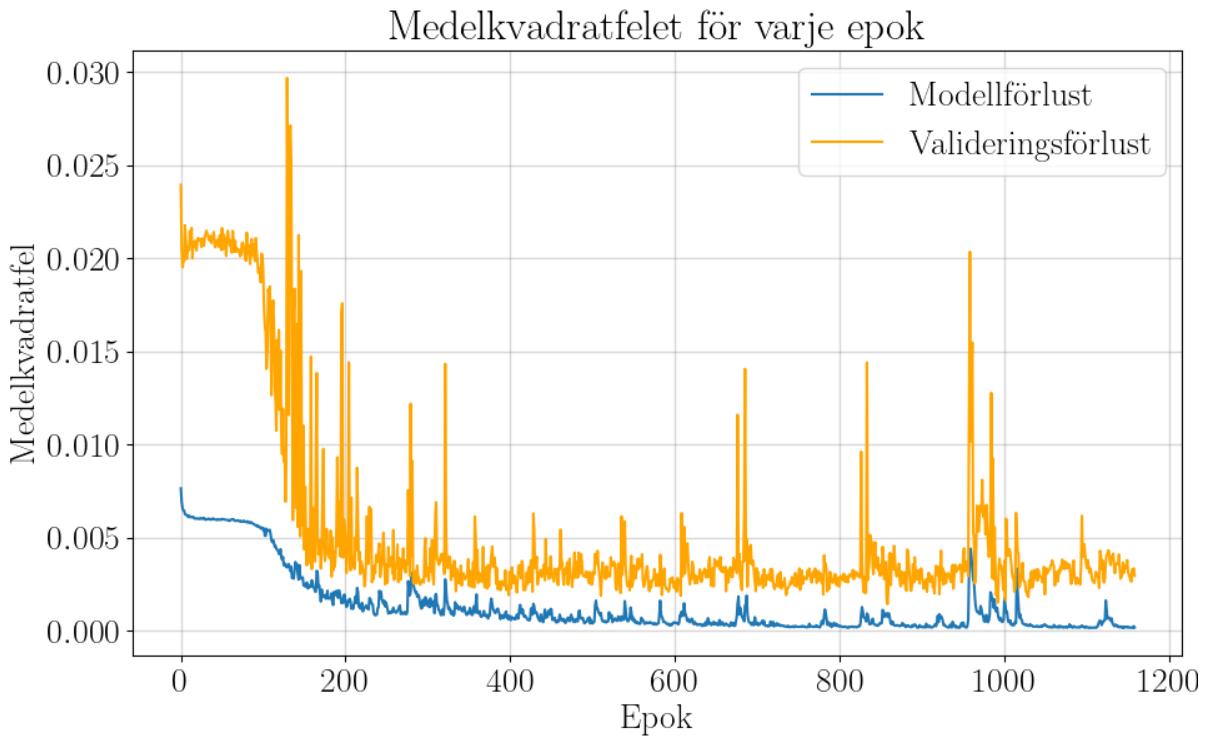
Detta avsnitt innehåller resultatet av den träning och de prediktioner som framställdes av modellen. För varje enskild situation som modellen tränats inför presenteras modellförlusten, jämförelser mellan hypotesen och testdataen samt modellens slutliga framtidsprediktioner.

4.1 Modell för 2021-04-09

Nedan presenteras resultatet för den modell som tränades fram till och med datumet 2021-04-09, vilket motsvarar dag nummer 435 i tidsserien.



Figur 20: Graf visande tidsserien av antalet fall som användes för att träna modellen för dag 435.



Figur 21: Graf visande utvecklingen av medelkvadratfelet under träningen av modellen för dag 435.

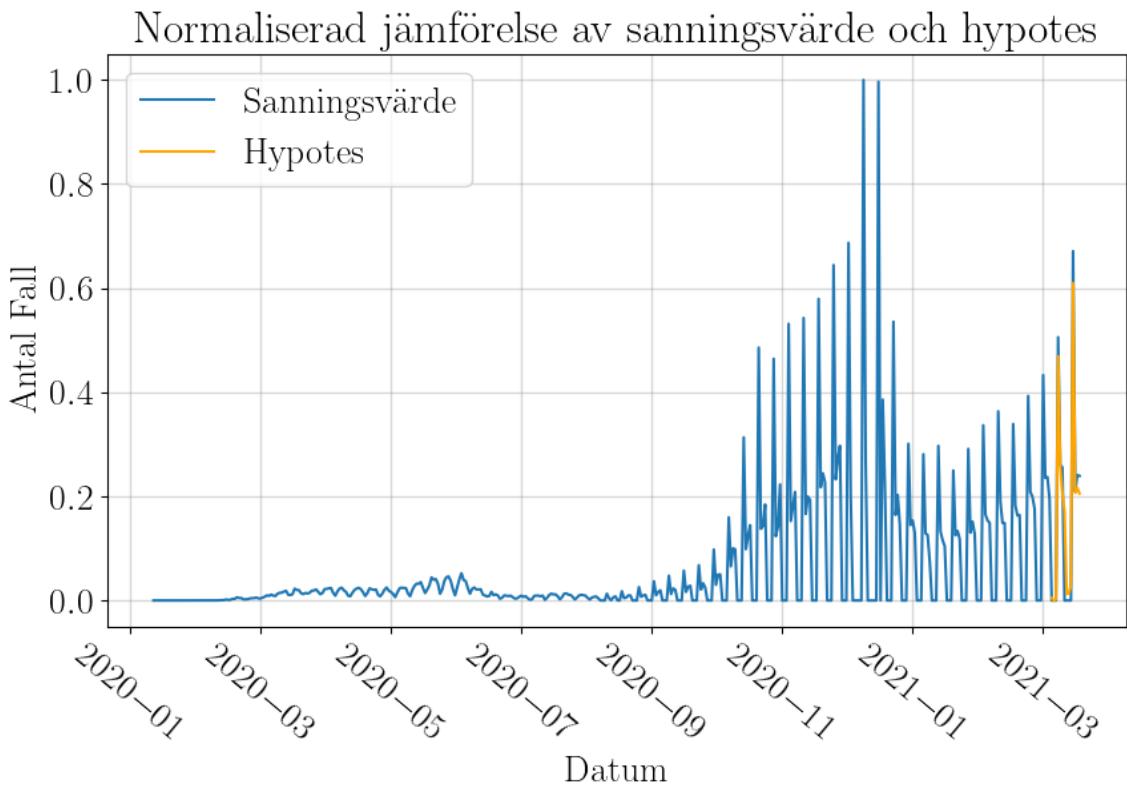
Nedan visar utskriften för träningen av modellen för dag 435.

```
Restoring model weights from the end of the best epoch: 859.
Epoch 859/1500

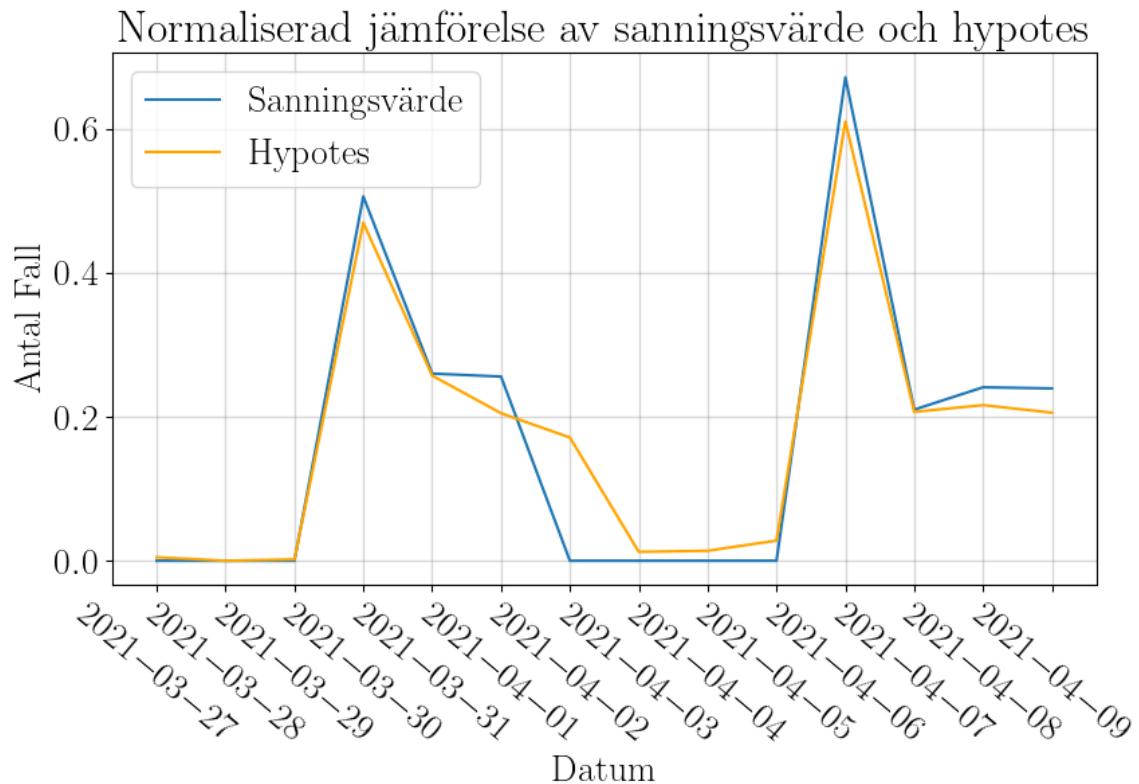
1/7 [===== >.....] - ETA: 0s - loss: 5.1818e-04
3/7 [===== >.....] - ETA: 0s - loss: 5.7656e-04
5/7 [===== >.....] - ETA: 0s - loss: 8.0877e-04
7/7 [=====]

- 0s 41ms/step - loss: 7.2743e-04 - val_loss: 0.0014
```

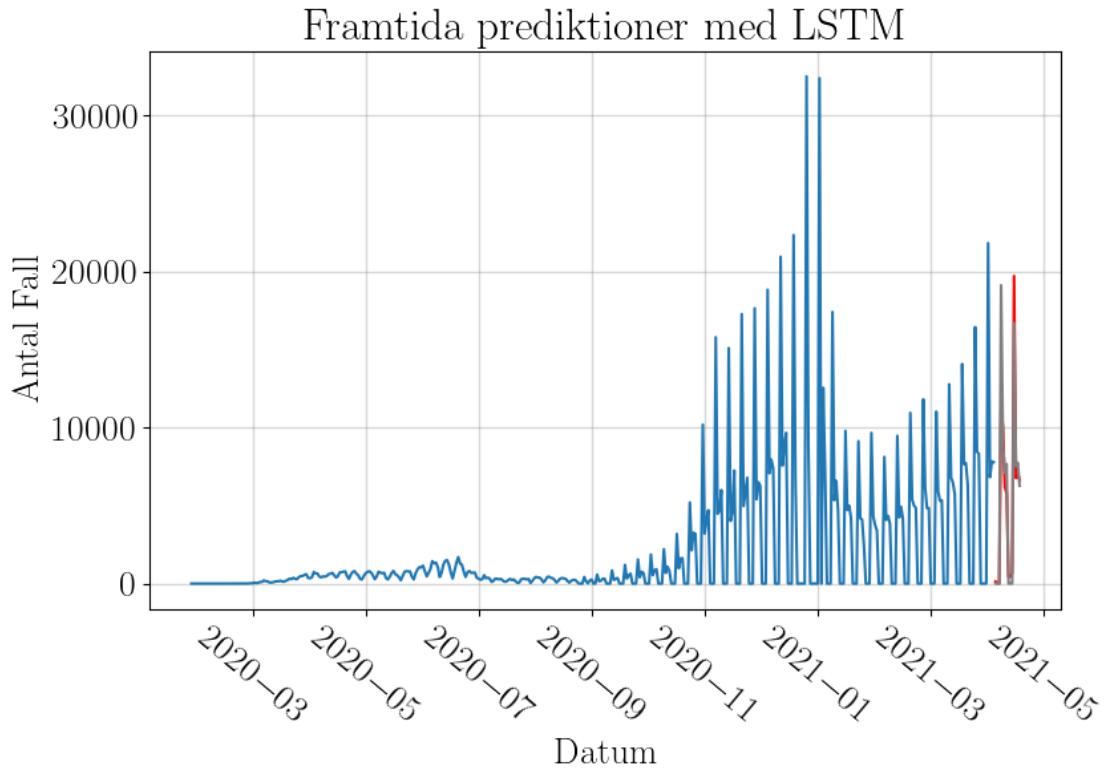
Medelkvadratfelet för träningsdata och testdata motsvarade $7.2743\text{e-}04$ respektive 0.0014. Träningen avslutades tidigt och vikterna återställdes för den bästa epoken 859.



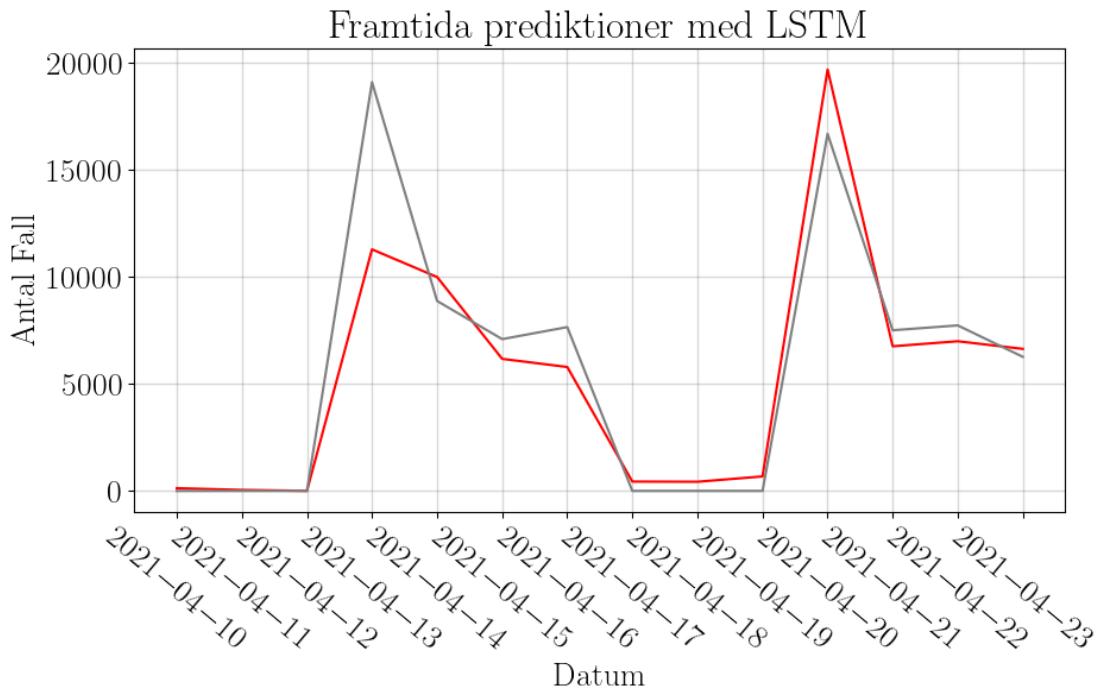
Figur 22: Graf visande jämförelse mellan sanningsvärden och antal predikterade fall av modellen för dag 435 under valideringsprocessen.



Figur 23: Förstorad version av graf visande jämförelse mellan sanningsvärden och antal predikterade fall av modellen för dag 435 under valideringsprocessen.



Figur 24: Graf visande jämförelse mellan sanningsvärdet och framtida predikterade fall av modellen för dag 435. Röd representerar antalet predikterade fall och grå representerar sanningsvärdet för antalet fall.

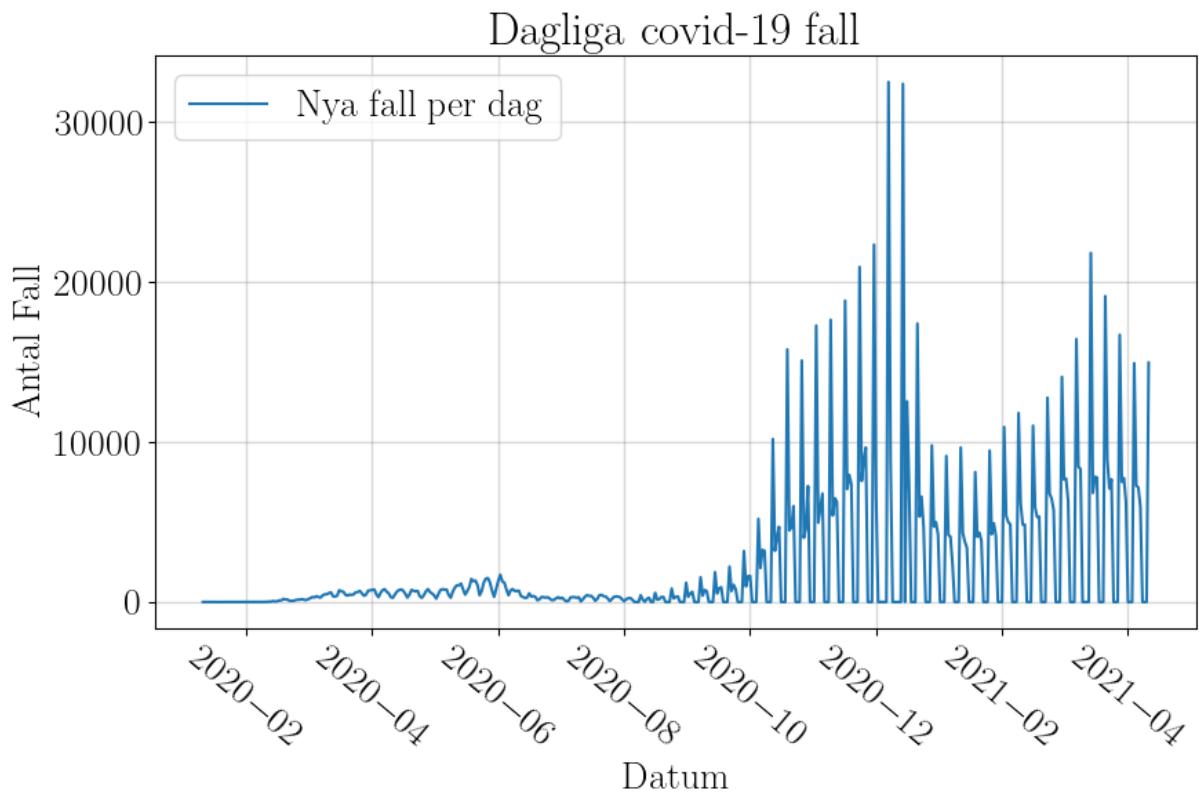


Figur 25: Förstorad version av graf visande jämförelse mellan sanningsvärdet och framtida predikterade fall av modellen för dag 435. Röd representerar antalet predikterade fall och grå representerar sanningsvärdet för antalet fall.

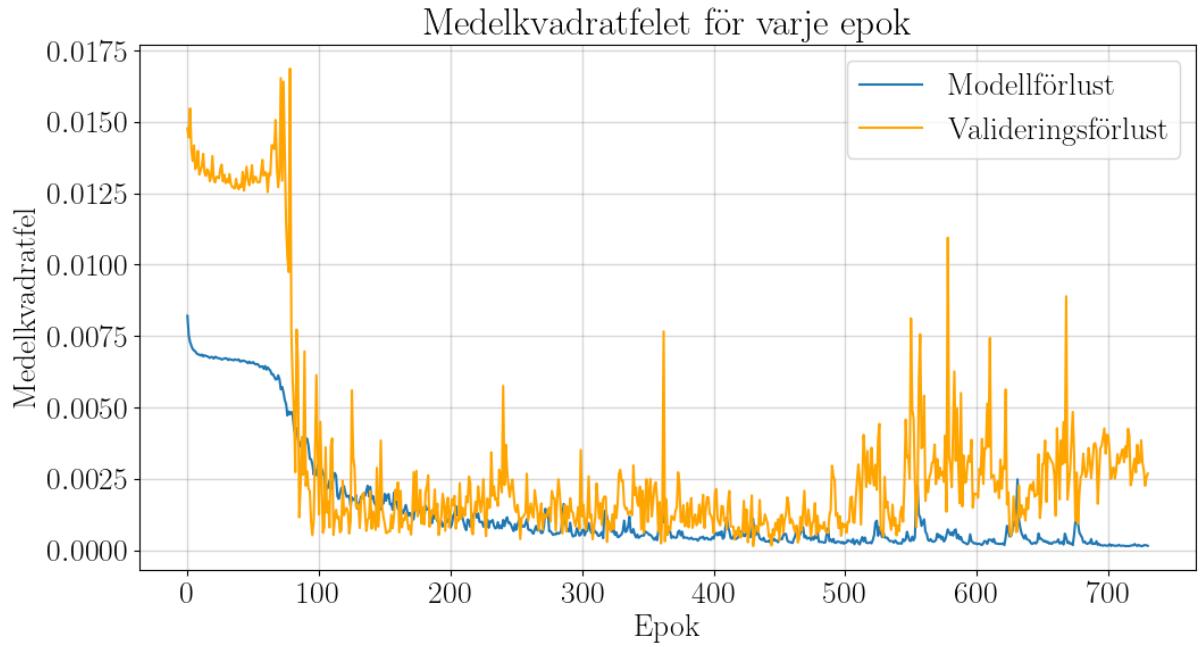
MSE för de predikterade fallen under valideringsprocessen var 0.0028472 och RMSE 0.0533594. MSE för de predikterade fallen för framtida prediktioner var 0.0052649 och RMSE 0.0725593.

4.2 Modell för 2021-05-04

Nedan presenteras resultatet för den modell som tränades fram till och med datumet 2021-05-04, vilket motsvarar dag nummer 460 i tidsserien.



Figur 26: Graf visande tidsserien av antalet fall som användes för att träna modellen för dag 460.



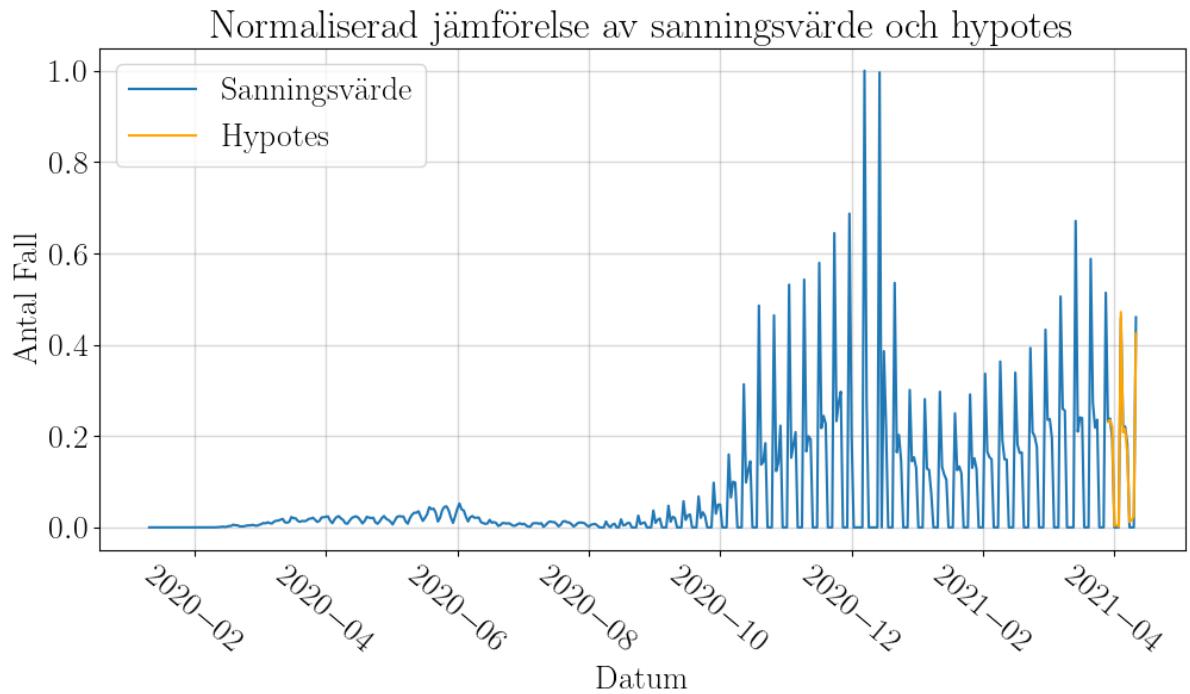
Figur 27: Graf visande utvecklingen av medelkvadratfelet under träningen av modellen för dag 460.

Nedan visar utskriften för träningen av modellen för dag 460.

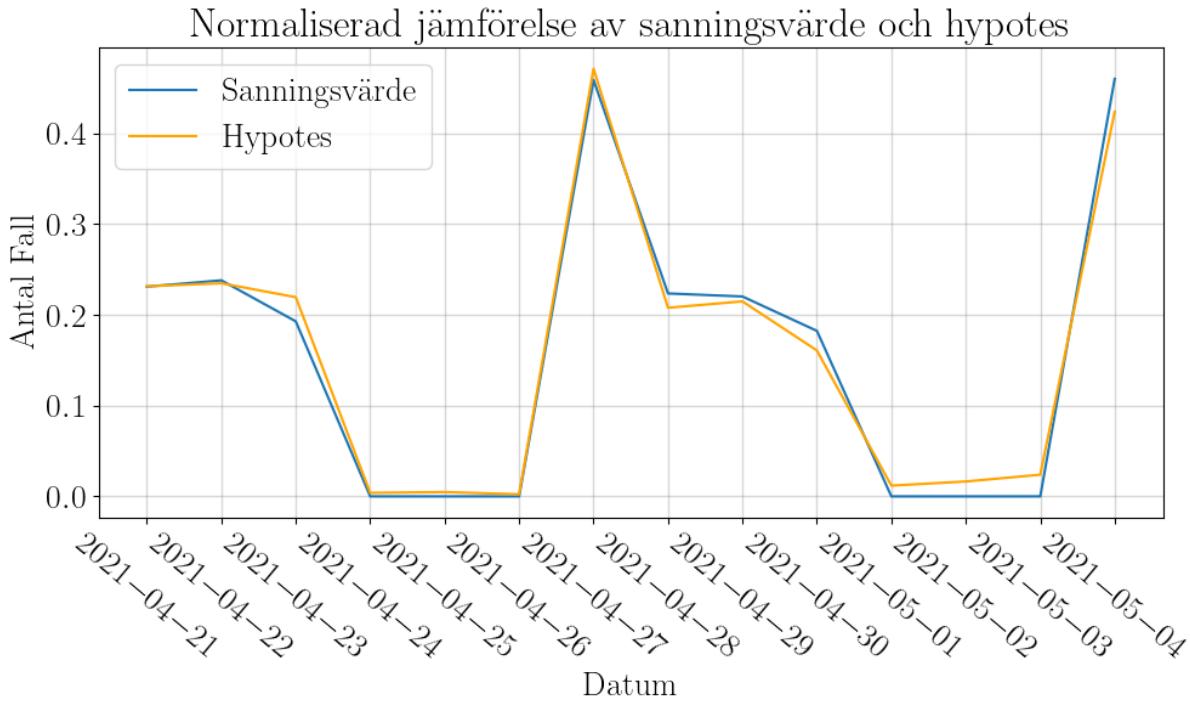
```
Restoring model weights from the end of the best epoch: 431.

Epoch 431/1500
1/7 [===== >.....] - ETA: 0s - loss: 0.0011
3/7 [===== >.....] - ETA: 0s - loss: 0.0016
5/7 [===== >.....] - ETA: 0s - loss: 0.0011
7/7 [=====]
- 0s 40ms/step - loss: 0.0011 - val_loss: 1.4117e-04
```

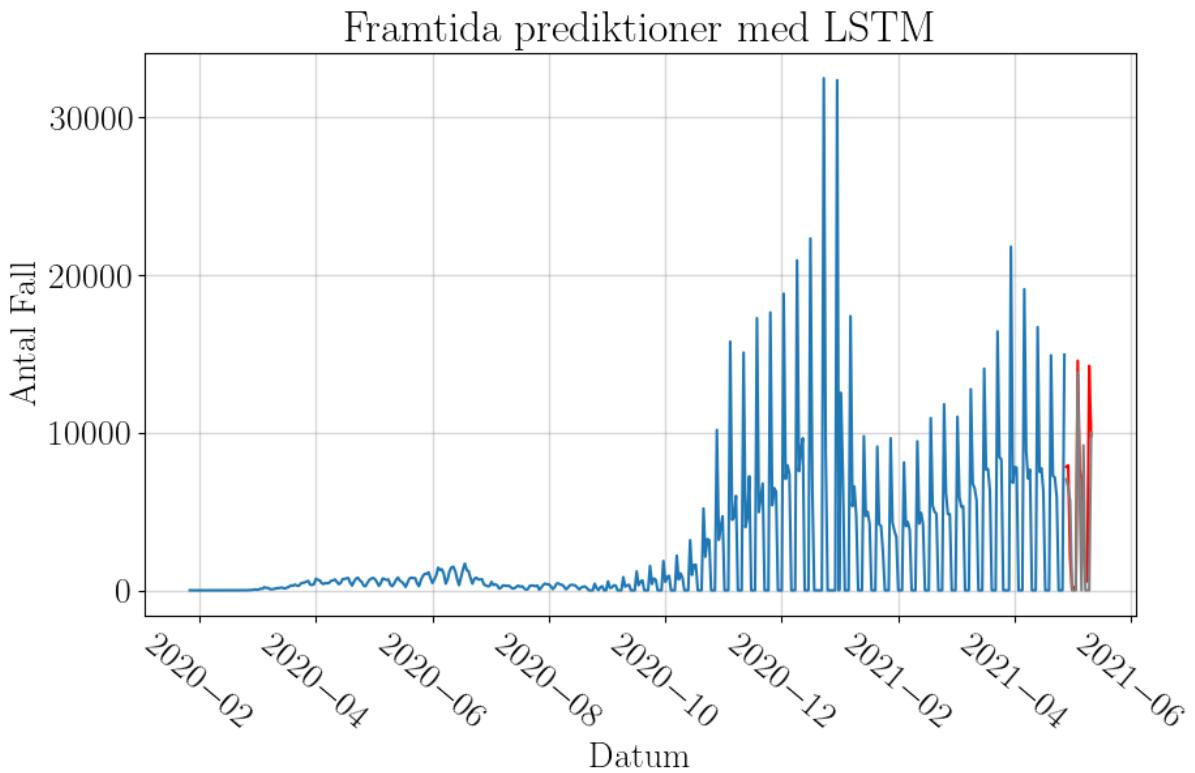
Medelkvadratfelet för träningsdatan och testdatan motsvarade 0.0011 respektive 1.4117×10^{-4} . Träningen avslutades tidigt och vikterna återställdes för den bästa epoken 431.



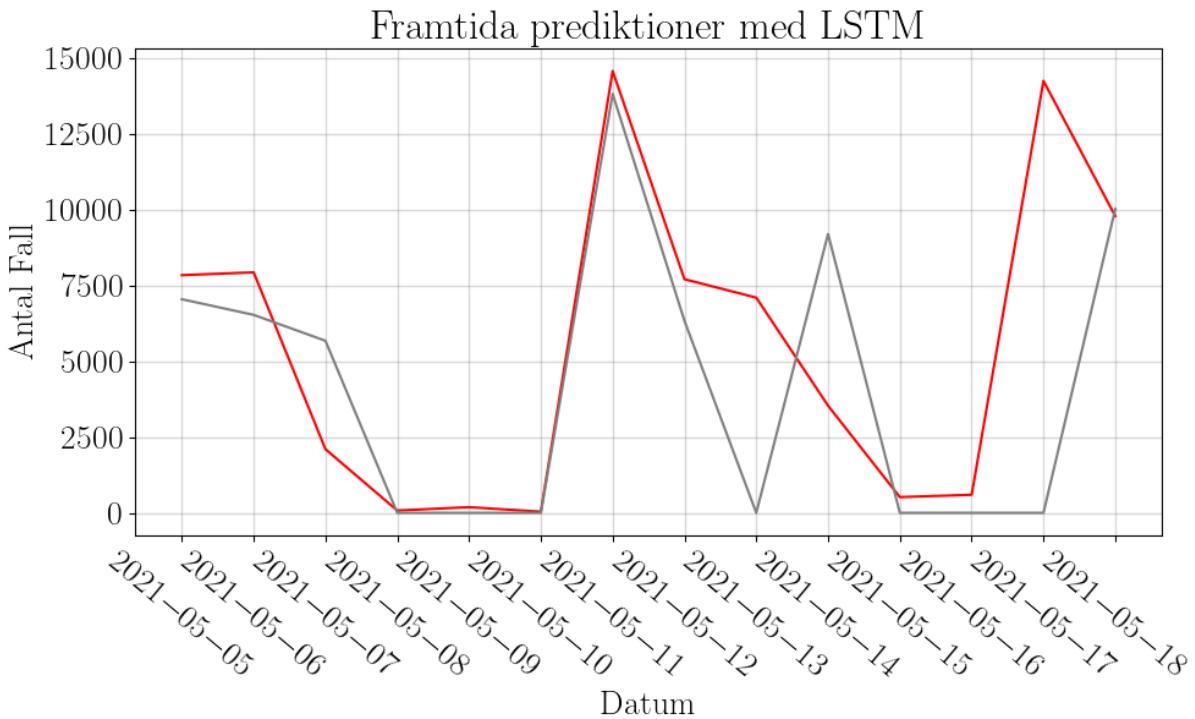
Figur 28: Graf visande jämförelse mellan sanningsvärden och antal predikterade fall av modellen för dag 460 under valideringsprocessen.



Figur 29: Förstorad version av graf visande jämförelse mellan sanningsvärden och antal predikterade fall av modellen för dag 460 under valideringsprocessen.



Figur 30: Graf visande jämförelse mellan sanningsvärdet och framtida predikterade fall av modellen för dag 460. Röd representerar antalet predikterade fall och grå representerar sanningsvärdet för antalet fall.

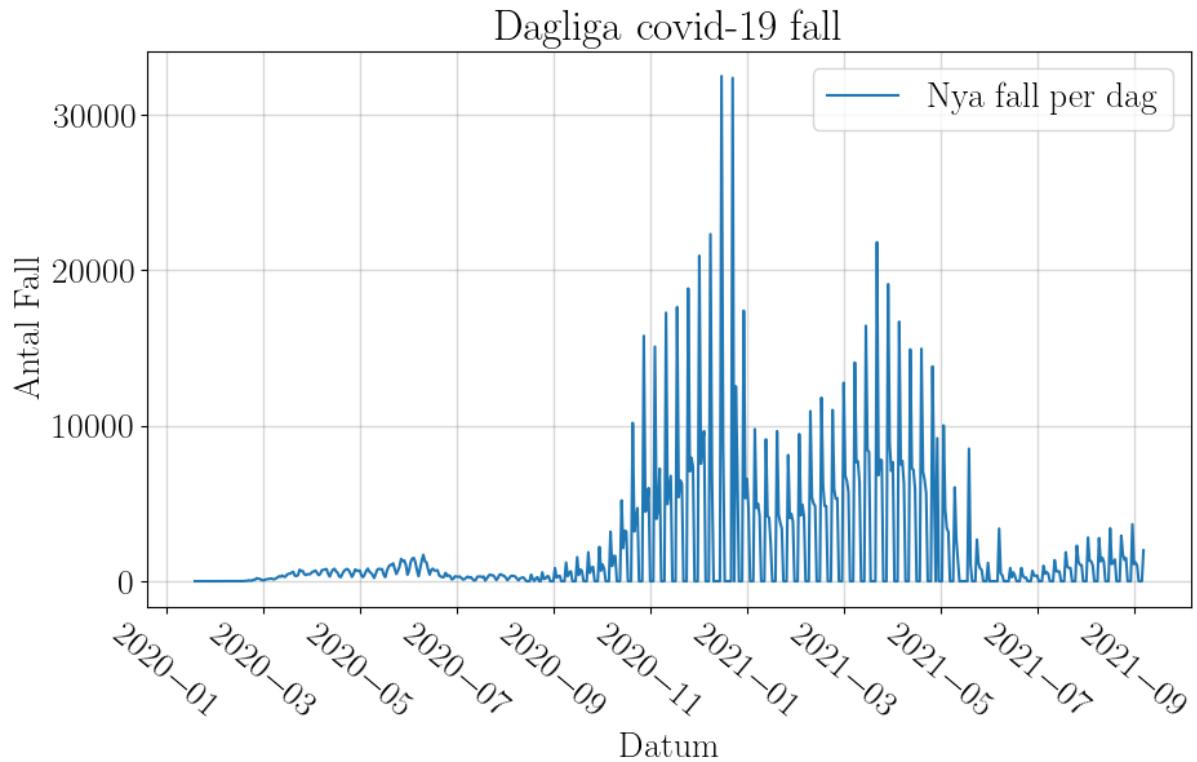


Figur 31: Förstorad version av graf visande jämförelse mellan sanningsvärdet och framtida predikterade fall av modellen för dag 460. Röd representerar antalet predikterade fall och grå representerar sanningsvärdet för antalet fall.

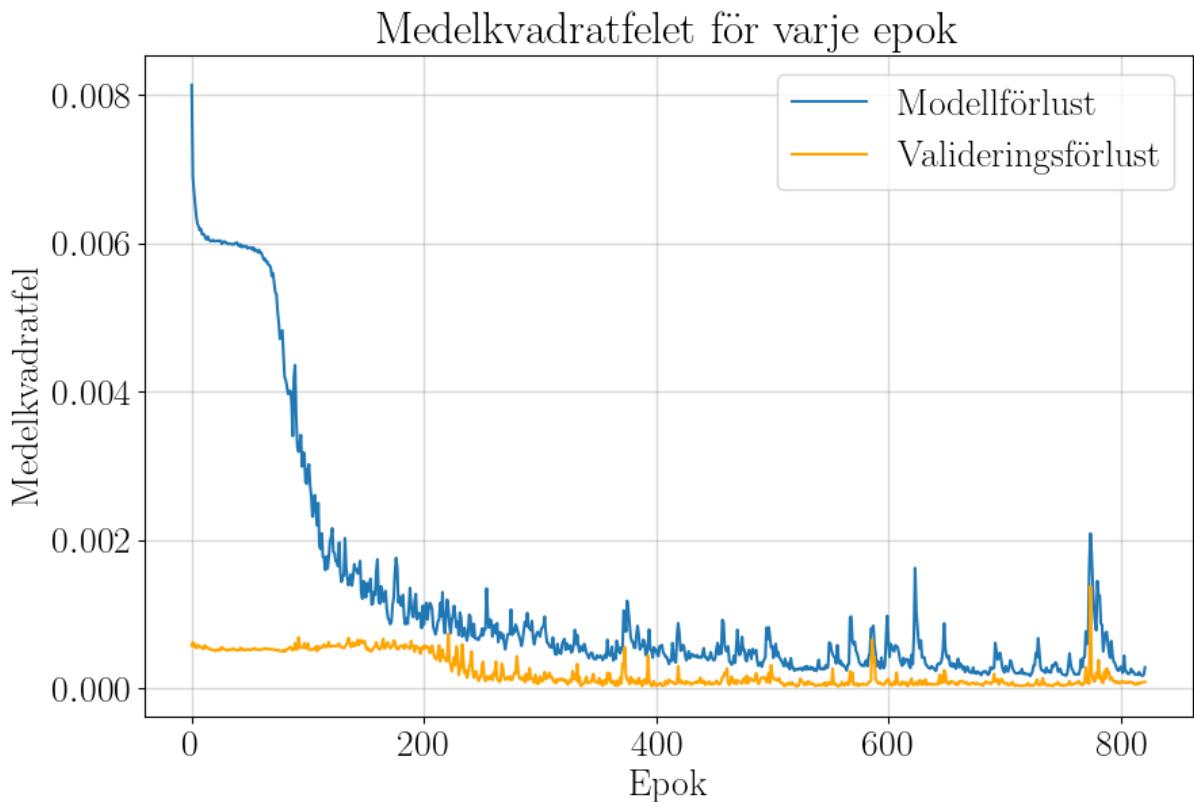
MSE för de predikterade fallen under valideringsprocessen var 0.0002823 och RMSE 0.0168028. MSE för de predikterade fallen för framtida prediktioner var 0.0205524 och RMSE 0.1433612.

4.3 Modell för 2021-09-20

Nedan presenteras resultatet för den modell som tränades fram till och med datumet 2021-09-20, vilket motsvarar dag nummer 600 i tidsserien.



Figur 32: Graf visande tidsserien av antalet fall som användes för att träna modellen för dag 600.



Figur 33: Graf visande utvecklingen av medelkvadratfelet under träningen av modellen för dag 600.

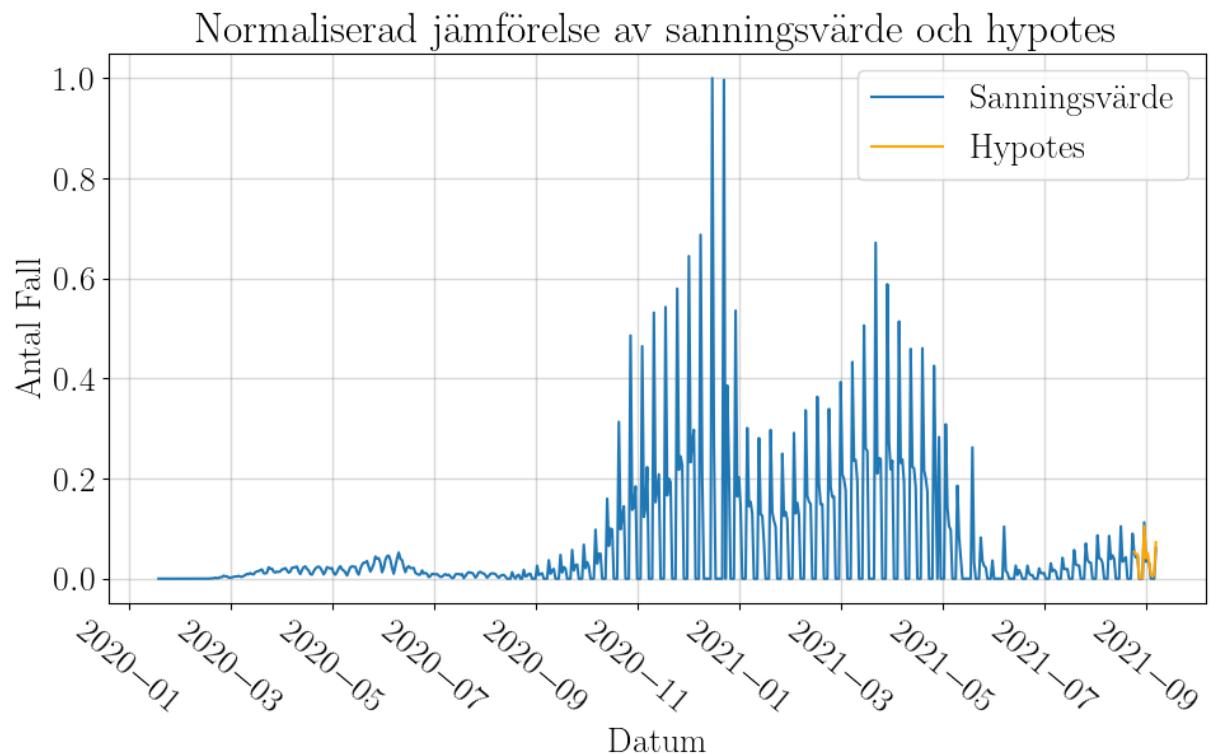
Nedan visar utskriften för träningen av modellen för dag 600.

```
Restoring model weights from the end of the best epoch: 522.

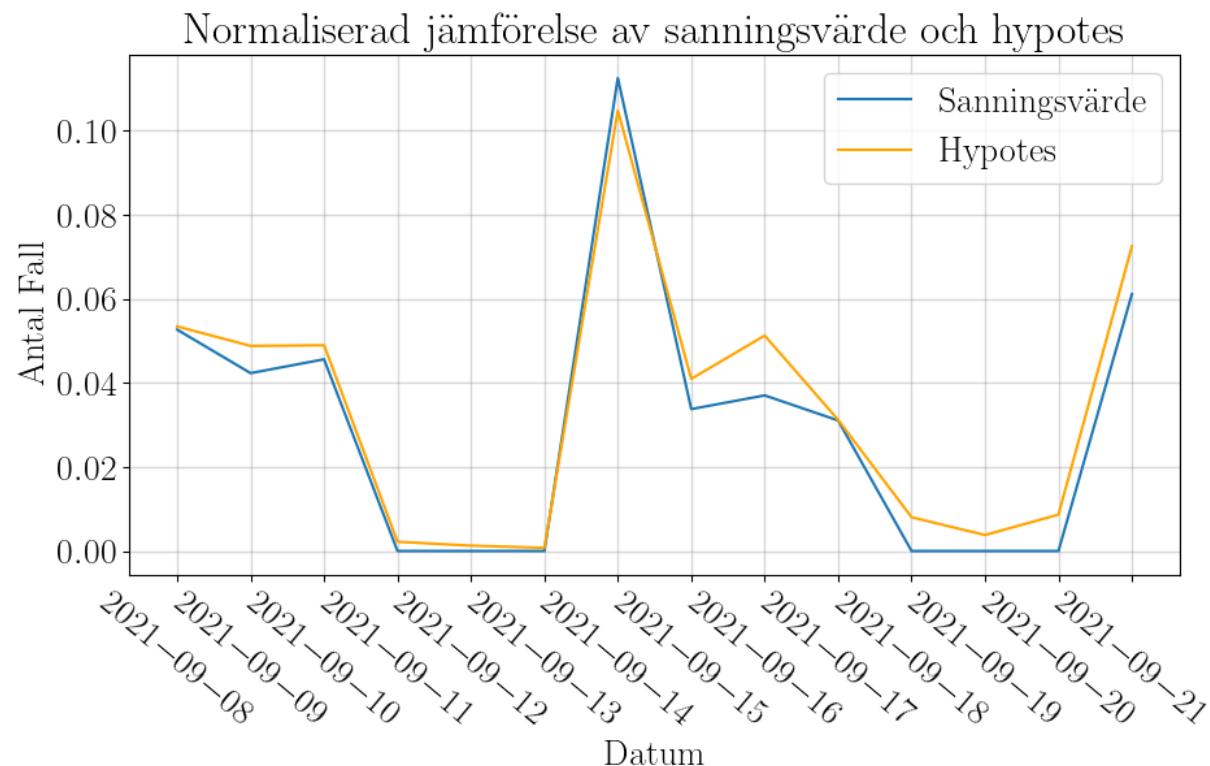
Epoch 522/1500
1/9 [==>.....] - ETA: 0s - loss: 2.4100e-04
3/9 [======>.....] - ETA: 0s - loss: 2.0416e-04
5/9 [======>.....] - ETA: 0s - loss: 2.2472e-04
7/9 [======>.....] - ETA: 0s - loss: 2.5453e-04
9/9 [=====]

- 0s 36ms/step - loss: 2.6256e-04 - val_loss: 4.2083e-05
```

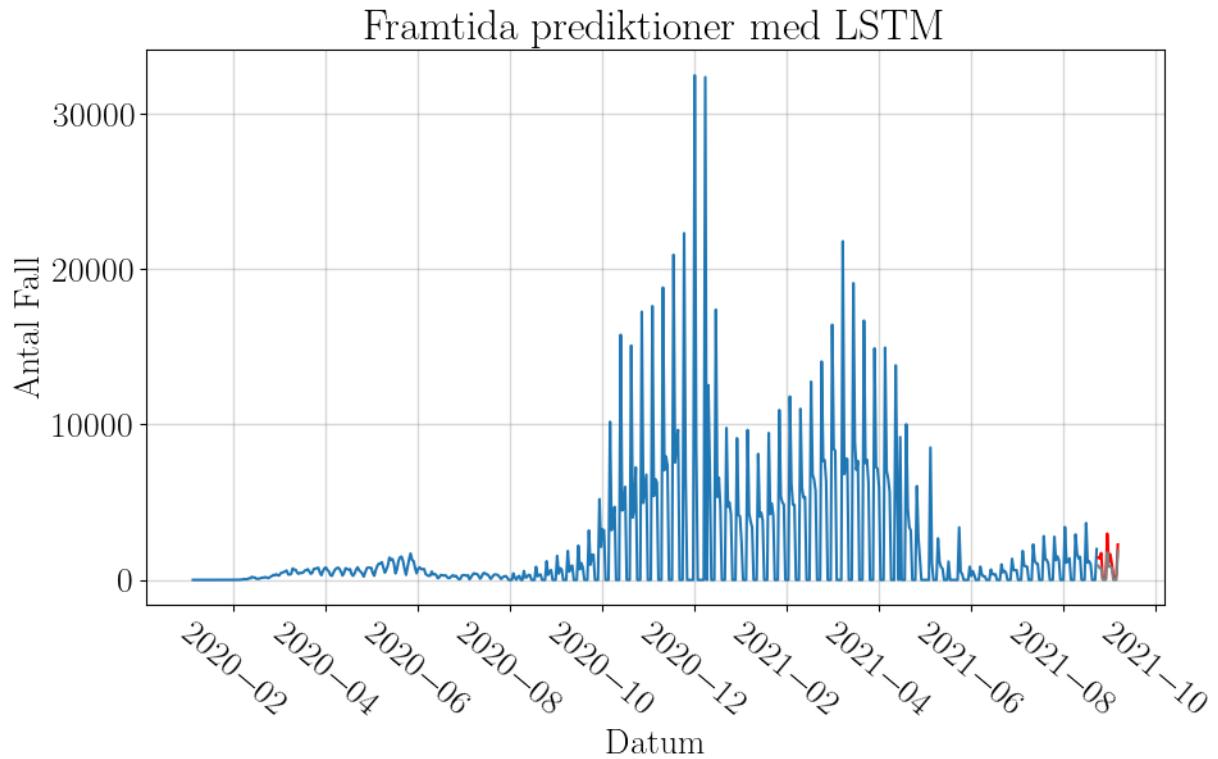
Medelkvadratfelet för träningsdata och testdata motsvarade $2.6256\text{e-}04$ respektive $4.2083\text{e-}05$. Träningen avslutades tidigt och vikterna återställdes för den bästa epoken 522.



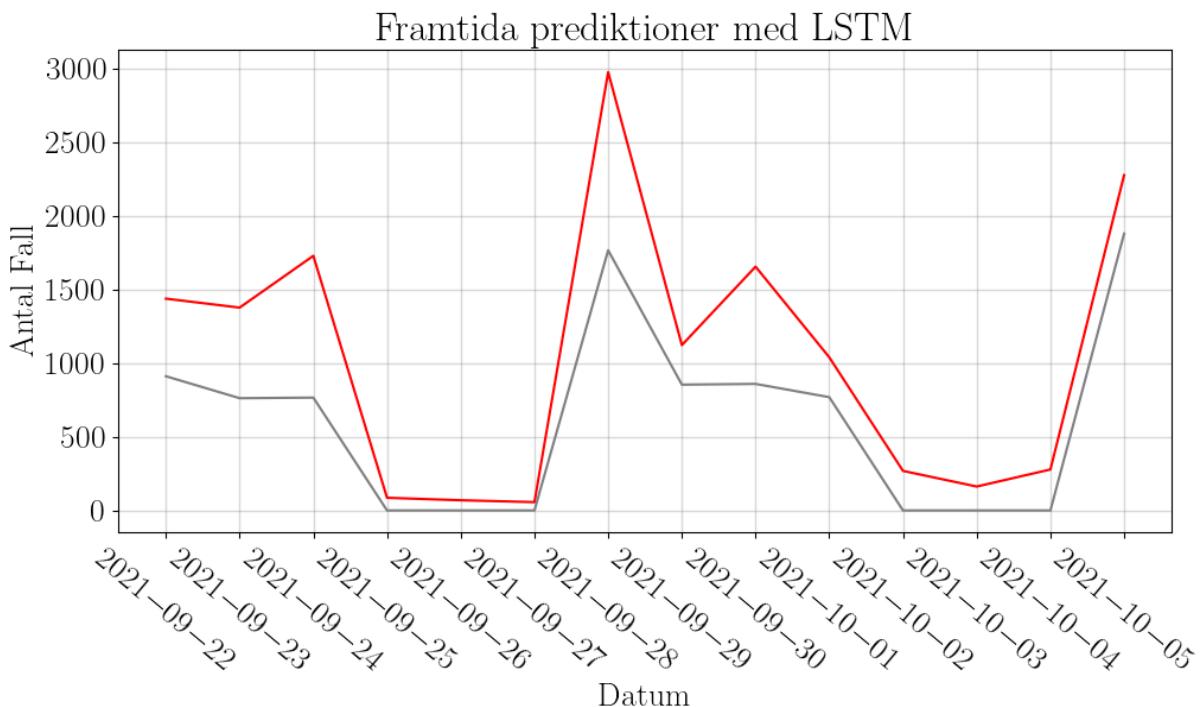
Figur 34: Graf visande jämförelse mellan sanningsvärden och antal predikterade fall av modellen för dag 600 under valideringsprocessen.



Figur 35: Förstorad version av graf visande jämförelse mellan sanningsvärden och antal predikterade fall av modellen för dag 600 under valideringsprocessen.



Figur 36: Graf visande jämförelse mellan sanningsvärdet och framtida predikterade fall av modellen för dag 600. Röd representerar antalet predikterade fall och grå representerar sanningsvärdet för antalet fall.



Figur 37: Förstorad version av graf visande jämförelse mellan sanningsvärdet och framtida predikterade fall av modellen för dag 600. Röd representerar antalet predikterade fall och grå representerar sanningsvärdet för antalet fall.

MSE för de predikterade fallen under valideringsprocessen var 4.7e-05 och RMSE 0.0068572. MSE för de predikterade fallen för framtida prediktioner var 0.0002824 och RMSE 0.0168052.

4.4 Jämförelse av utvärderingsmått

	Valideringsprocess		Framtida prediktioner	
Modell	MSE	RMSE	MSE	RMSE
435	0.0028472	0.0533594	0.0052649	0.0725593
460	0.0002823	0.0168028	0.0205524	0.1433612
600	4.70E-05	0.0068572	0.0002824	0.0168052

Tabell 6 visande utvärderingsmåtten för de olika modellerna som tränats.

Genom att studera ovan tabell ser vi att modellen som tränades för 600 dagar var den som presterade avsevärt bäst, med lägst MSE både i valideringsprocessen och för de framtida prediktionerna. Ingen av modellerna gav ett medelkvadratfel som översteg 0.1 för någon av processerna.

5 Diskussion

Åtskilliga antalet modeller har skapats i hopp om att lyckas hämma, eller åtminstone förbereda sig inför spridningen av covid-19 genom att förutspå variabler såsom antalet fall eller intensivvårdsplatser inom den närmsta tiden. Den i särklass viktigaste faktorn för att lyckas modellera en funktion utifrån ett givet dataset är att det faktiskt finns ett återkommande mönster i datan och att den inte utgörs av slumpmässiga förändringar. En viktig analys borde i sådana fall handla om en tidsserie för antalet insjuknade i covid-19 ens följer ett sådant tydligt mönster. Om bara statistik för just antalet fall tas i beaktande är svaret nej, vilket konstateras genom att studera grafen för antalet fall där varje ny våg har slagit alla tidigare rekord flerfaldigt, utan någon till synes självklar anledning. Om ändå ytterligare variabler såsom antalet utförda tester, dödsfall eller fullvaccinerade tas hänsyn till är det lätt att luras av att självklara samband faktiskt stämmer överens. Att fler utförda tester innebär att fler personer är sjuka räcker inte för att med säkerhet dra slutsatsen att ett tydligt mönster går att hitta. Lösningen ges i stället genom att studera de olika virusvarianterna och deras egenskaper. Genom att kombinera flera oberoende variabler med statistik för olika virusvarianter går det att hitta mönster i datan för antalet sjuka i covid-19. Svårare är det ändå att i förväg veta vilken ny virusvariant som har muterat sådant att den ökar smittspridningen, något som emellertid går att uppnå genom att studera proteinstrukturer. Och tack vare att forskare klassificerar vissa virusvarianter som oroväckande finns denna data att tillgå vem som helst.

Datasetet som används i arbetet inkluderar utförlig statistik för de 20 främsta virusvarianterna, utöver de övriga variablerna. Det finns alltså en teoretisk möjlighet att modellera en funktion som satisfierar frågeställningen. Genom att studera de olika resultaten som framlagts kan slutsatsen dras att det går att framställa en modell som med hög noggrannhet kan prediktera antalet framtida fall av covid-19. Jämförelsen av utvärderingsmåtten styrker detta påstående vidare, där det längsta medelkvadratfelet enbart uppgick till 0.0002824 och inget värde översteg 0.1 för något av resultaten som lades fram. Att alla modeller mättades och avbröt träningen kan tyda på att de inte finjusterats fullt ut, något som förklaras i att detta skedde i grova drag. Ytterligare en slutsats som kan dras är att modeller tenderar att prestera bättre ju mer data de tränats för, och för data som inte ger upphov till stora plötsliga förändringar. Det är ändå dock att de

vikt att poängtera att ingen av de modeller som framtagits har baserats på data som kan anses otillräcklig, då det som konstaterat i sektion 3.3.1 är underförstått att en sådan funktion omöjligen kan modelleras och därav ej är av intresse. Det är alltså inte garanterat att modellen kontinuerligt kan användas för att producera ett kvalitativt resultat i alla godtyckliga situationer.

6 Slutsats

Den slutsats som kan dras med avseende på de resultat som presenterats gällande frågeställningen är att en modell för prediktion av antalet framtida fall av covid-19 kan tas fram med hjälp av artificiell intelligens med tillfredsställande resultat. En modell av sådan karaktär kan ge insiktsfull information om hur samhället bör förbereda sig under en pandemi, och kan utnyttjas för att ge en föraning om den kommande smittspridningen. För att ha en möjlighet att uppnå högkvalitativa resultat krävs emellertid ett dataset inkluderande ett stort omfång olika variabler över en lång tidsperiod. Med det sagt bör de prediktioner som en modell frambringar, oavsett tidigare resultat, tas i åtanke med största möjliga aktsamhet och ej ligga till grund för större viktigare beslut. Eftersom covid-19 inte längre utsätts för aktiv testning är däremot all framtid data ogiltig och vikten av framtida prediktioner försätter, däremot kan föregående resultat användas för att ge insiktsfull om hur framtida pandemier bör hanteras.

7 Källförteckning

- [1] Region Örebro län. *Covid-19 inte längre en allmänsfarlig och samhällsfarlig sjukdom.* <https://www.regionorebrolan.se/sv/aktuellt/covid-19-inte-langre-en-allmanfarlig-och-samhallsfarlig-sjukdom/>. Hämtad 2022-04-01.
- [2] Stockholms universitet. *Tidsserier.* <https://www.statistics.su.se/forskning/tidsserier>. Hämtad 2022-04-01.
- [3] William S Chandler. *NASA's Surface Meteorology and Solar Energy Web Portal (Release 6.0).* https://www.researchgate.net/figure/Daily-averaged-temperature-time-series-data_fig2_268328855. Hämtad 2022-04-01.
- [4] Dave Kuhlman. *A Python Book: Beginning Python, Advanced Python, and Python Exercises.* Platypus Global Media, 2011.
- [5] Guido Van Rossum. *The History of Python.* <https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html>. Hämtad 2022-04-01.
- [6] Python. *Introduction (API).* <https://docs.python.org/3/c-api/intro.html>. Hämtad 2022-04-01.
- [7] Google Developer. *Introduction to TensorFlow.* <https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit>. Hämtad 2022-04-01.
- [8] Keras. *About Keras.* <https://keras.io/about/>. Hämtad 2022-04-01.
- [9] Världshälsoorganisationen. *Tracking SARS-CoV-2 variants.* <https://www.who.int/en/activities/tracking-SARS-CoV-2-variants/>. Hämtad 2022-04-01.
- [10] Ola Danielsson. "Så utvecklas nya virusvarianter". I: (2022).
- [11] Sharon Peacock. "Here's what we know about the new variant of coronavirus". I: (2020).
- [12] Science Media Center. *Expert reaction to South African variant of SARS-CoV-2, as mentioned by Matt Hancock at the Downing Street press briefing.* <https://www.sciencemediacentre.org/expert-reaction-to-south-african-variant-of-sars-cov-2-as-mentioned-by-matt-hancock-at-the-downing-street-press-briefing/>. Hämtad 2022-04-01.

- [13] Centers for Disease Control och Prevention. *What You Need to Know About Variants*. <https://www.cdc.gov/coronavirus/2019-ncov/variants/about-variants.html>. Hämtad 2022-04-01.
- [14] Center for Infectious Disease Research och Policy. *Lung tissue study sheds light on fast Omicron spread*. <https://www.cidrap.umn.edu/news-perspective/2021/12/lung-tissue-study-sheds-light-fast-omicron-spread>. Hämtad 2022-04-01.
- [15] Stuart Russell och Peter Norvig. *Artificial Intelligence: A Modern Approach, Global Edition 4th*. 2021.
- [16] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc., 2019.
- [17] Sten Henriksson. Nationalencyklopedin. *cybernetik*. <https://www.ne.se/uppslag/sverk/encyklopedi/1%C3%A5ng/cybernetik>. Hämtad 2022-02-17.
- [18] Karl Johan Åström. Nationalencyklopedin. *reglerteknik*. <https://www.ne.se/uppslagsverk/encyklopedi/1%C3%A5ng/reglerteknik>. Hämtad 2022-02-17.
- [19] WatElectronics.com. *What is a Feedback Amplifier : Working, Types Its Characteristics*. <https://www.watelectronics.com/what-is-a-feedback-amplifier-working-types-its-characteristics/>. Hämtad 2022-02-17.
- [20] Michael Bernico. *Deep Learning Quick Reference: Useful hacks for training and optimizing deep neural networks with TensorFlow and Keras*. Packt Publishing Ltd, 2018.
- [21] Jitendra R. Raol och Sunilkumar S. Mankame. "Artificial neural networks: A brief introduction". I: (1996).
- [22] Christopher Olah. *Understanding LSTM Networks*. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Hämtad 2022-02-17.
- [23] S. Rao Saahil Afaq. *Significance Of Epochs On Training A Neural Network*. <https://www.semanticscholar.org/paper/Significance-Of-Epochs-On-Training-A-Neural-Network-Afaq-Rao/c010c03972a0b37cd41cab710877595b3576512f>. Hämtad 2022-04-01.

- [24] m.fl. Aditya Devarakonda. *AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks*. <https://arxiv.org/abs/1712.02029>. Hämtad 2022-04-01.
- [25] GeeksforGeeks. *Python / Mean Squared Error*. <https://www.geeksforgeeks.org/python-mean-squared-error/>. Hämtad 2022-04-01.

8 Bilagor

8.1 Bilaga 1

```
new_cases, total_cases, new_cases_smoothed, total_deaths, new_deaths,  
new_deaths_smoothed, total_cases_per_million, new_cases_per_million,  
new_cases_smoothed_per_million, total_deaths_per_million,  
new_deaths_per_million, new_deaths_smoothed_per_million,  
reproduction_rate, icu_patients, icu_patients_per_million, hosp_patients,  
hosp_patients_per_million, weekly_icu_admissions,  
weekly_icu_admissions_per_million, weekly_hosp_admissions,  
weekly_hosp_admissions_per_million, new_tests, total_tests,  
total_tests_per_thousand, new_tests_per_thousand, new_tests_smoothed,  
new_tests_smoothed_per_thousand, positive_rate, tests_per_case,  
total_vaccinations, people_vaccinated, people_fully_vaccinated,  
total_boosters, new_vaccinations, new_vaccinations_smoothed,  
total_vaccinations_per_hundred, people_vaccinated_per_hundred,  
people_fully_vaccinated_per_hundred, total_boosters_per_hundred,  
new_vaccinations_smoothed_per_million, new_people_vaccinated_smoothed,  
new_people_vaccinated_smoothed_per_hundred, stringency_index,  
handwashing_facilities, excess_mortality_cumulative_absolute,  
excess_mortality_cumulative, excess_mortality,  
excess_mortality_cumulative_per_million, AY.4.2, B.1.1.529, B.1.1.7,  
B.1.1.7+E484K, B.1.351, B.1.427/B.1.429, B.1.525, B.1.526, B.1.617.1,  
B.1.617.2, B.1.617.3, B.1.620, B.1.621, C.37, Other, P.1, P.3,
```

Bilaga 1: Samtliga kolumner och de oberoende variabler som inkluderats i datasetet för träningen av modellen. Varje enskild kolumn är separerad av ett kommatecken.