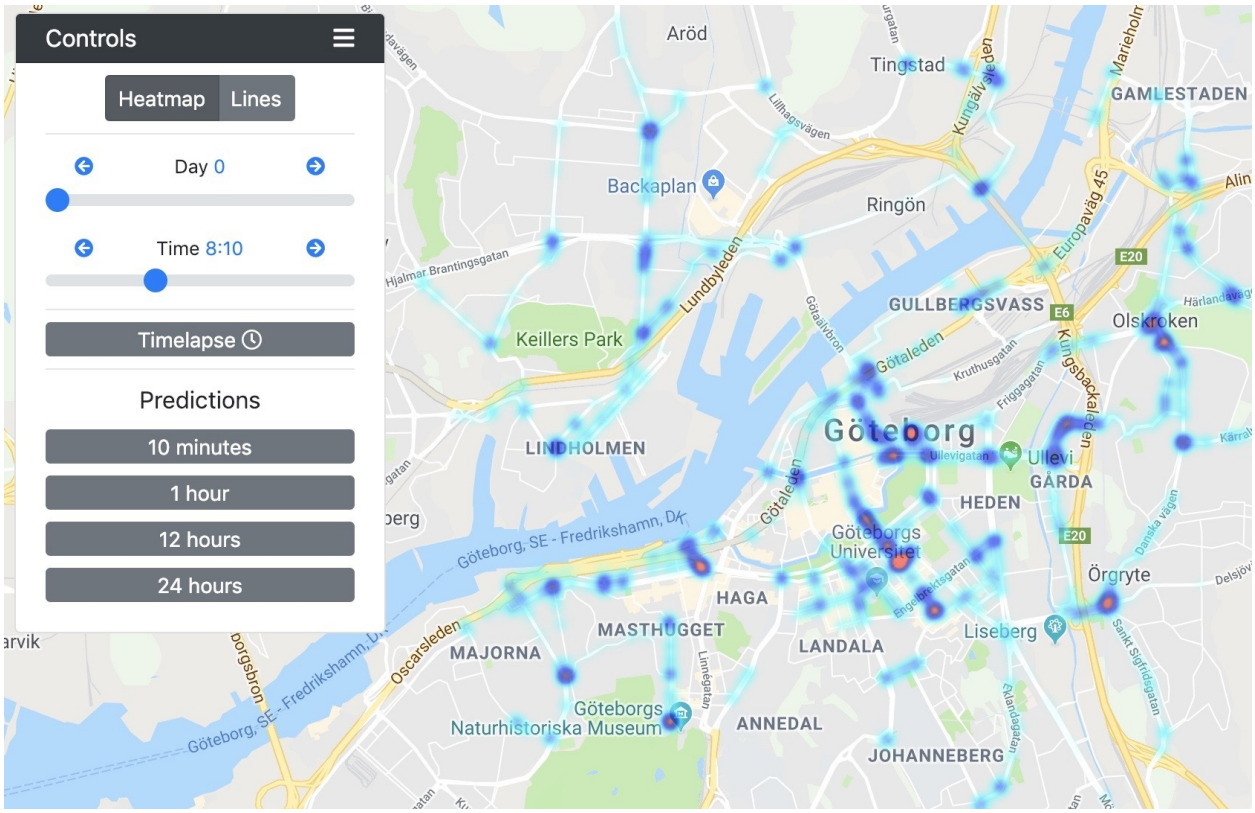
Maskininlärning tillämpat för framställning av trafikprognoser

Visualiserat i en webbapplikation



Lars Andersson, Linus Aronsson, Aron Bengtsson

Daniel Detlefsen, Oskar Lenschow, Erik Tran

Sammanfattning

På senare tid har det skett en stor ökning av tillgänglig data inom alla möjliga områden. Detta inkluderar data om trafikflödet på en viss plats vid en viss tidpunkt, vilket är speciellt intressant för detta projekt. Insamling av trafikdata kan göras på många olika sätt, exempelvis genom att läsa av GPS-sensorer i människors mobiltelefoner eller genom fordonsdetektor-system. Genom att utnyttja denna stora mängd av tillgänglig data så redogör denna rapport för ett lösningsförslag på hur man kan förutspå framtida trafiksituationer med hjälp av maskininlärning. Resultatet visualiseras i en webbapplikation. För att åstadkomma detta så har vi samlat trafikdata i Göteborgsregionen i realtid med hjälp av ett API från företaget HERE. Efter en del undersökning så visade det sig att den maskininlärningsteknik som bör användas är artificiella neurala nätverk (ANN). Det mest lämpliga ANN i detta fall är ett så kallat Recurrent Neural Network (RNN) eftersom datan inkommer i en tidssekvens.

Innehåll

1. Inledning

Artificell intelligens (AI) är en typ av intelligens som uppvisas av maskiner. Många forskare och AI-läroböcker definierar detta område som "studien och utformningen av intelligenta agenter", där en intelligent agent är ett system som har förmågan att uppfatta sin omgivning och beräkna åtgärder som maximerar sina chanser att framgångsrikt uppnå sina mål [1]. Detta akademiska studieområde utforskas intensivt bland flera aktörer, och några av de främsta målen inom AI-forskningen är exempelvis inlärning, resonemang, planering, kommunikation samt förmågan att flytta och manipulera objekt. För detta projekt kommer inlärning vara målsättningen. Liknande projekt som applicerar detta, som också kallas Machine Learning (ML), kräver en stor mängd nyanserad data som relaterar till den intelligenta agentens huvudsakliga uppgift.

Genom att kombinera en rik datauppsättning, uppbyggd på realtidsdata från Göteborg, samt styrkan av AI, så utforskar det här projektet möjligheterna att framställa trafikprognoser i form av en webapplikation.

1.1 Bakgrund

Projektet är i samverkan med Chalmers och Cybercom och har en projekttid på sju veckor. Cybercom är ett IT-konsultbolag som arbetar för att hjälpa företag och organisationer att ta tillvara den uppkopplade världens möjligheter och stärka deras konkurrenskraft. Genom teknisk expertis och god affärsinsikt erbjuder de innovativa och hållbara lösningar.

1.2 Syfte

Syftet med projektet är att skapa en applikation för att kunna förutse framtida trafiksituationer med hjälp av ML-teknik och underlätta vid trafikplanering. Applikationen ska visualisera en uppskattad trafiksituation över ett framtida scenario, vilket kan ge upphov till att vilja utforska möjliga tillämpningsområden. Exempelvis skulle det bli möjligt att hitta den effektivaste rutten till en önskad destination, upptäcka orsaken till trafikstockning, eller bestämma den potentiellt lämpligaste tidpunkten att stänga av en väg för vägarbete.

1.3 Frågeställningar

Uppdragsgivarens önskemål var att få översikt över framtida trafikmönster i Göteborg. Detta kunde förslagsvis från uppdragsgivaren representeras med hjälp av ett

färgdiagram med funktionen att snabbspola förloppet. Genom följande frågeställningar har dessa önskemål konkretiserats.

- Hur fungerar applikationen?
- Vilka datasets behövs för att förutse trafiksituationer med hög noggranhet?
- Hur bör trafikprognosen visualiseras?

1.4 Avgränsningar

I överenskommelse med handledaren från Cybercom bestämdes att projektets omfattning bör hållas öppet. Anledningen till detta är för att skapa utrymme för analys av projektets användningsområden för en eventuell vidareutveckling inom ramen av ett examensarbete, våren 2019. Detta beslut påverkar framför allt punkt tre från 1.3.

2. Teori

Den data som använts i detta projekt är lagrad som en tidssekvens. Mer specifikt är trafikflödet inom en viss region i Göteborg uppmätt och lagrad var tionde minut. Den här typen av data är vad som brukar kallas för tidsserier och är ett välstuderat område [2]. Avsnitt 2.1 börjar därför med att introducera detta koncept och beskriver sedan traditionella statistiska metoder för prognosframtagningar. I avnistt 2.2 introduceras maskininlärning som på senare tider visat sig kunna göra prognoser baserat på tidsserier med betydligt större noggrannhet. Den maskininlärningsteknik som framförallt använts i detta projekt är Artificiella Neurala Nätverk (ANN). Två typer av ANN introduceras, Feedforward Neural Networks (FFNN) i avsnitt 2.2.1, och Recurrent Neural Networks (RNN) i avsnitt 2.2.2. I avsnitt 2.3 ges en kort beskrivning om databashanteringen som valts.

2.1 Tidsserier

En tidsserie är en sekvens X av mätningar x_t från en observerbar variabel x vid successiva tidpunkter med lika tidsintervall mellan varje punkt. Analys av tidsserier görs inom många olika områden som exempelvis statistik, signalbehandling och väderprognosframtagning [2]. Syftet med analysen är att hitta eventuella tidsberoende mönster som leder till bättre förståelse av tidsserien. Dessa mönster kan exempelvis möjliggöra att en tidsserieprediktion. Detta innebär att man baserat på tidigare data i tidsserien tar fram en prognos för hur tidsserien kommer att se ut i framtiden. Detta användningsområde är specifikt intressant för projektet och diskuteras mer i avsnitt 2.1.1.

2.1.1 Traditionell prognosframtagning

Det finns ett flertal olika metoder som traditionellt använts för prognosframtagningar baserat på tidsserier. De två kanske enklaste metoderna förklaras kortfattat med ett exempel. Anta att man har en tidsserie $X = \{x_1, x_2, ..., x_T\}$ med T stycken mätningar. Man vill nu ta fram en prognos för k tidssteg in i framtiden vilket betecknas som $\hat{y}_{T+k}$. Den första metoden tar då fram en prognos genom att ta medelvärdet av alla tidigare mätningar enligt $\hat{y}_{T+k} = (x_1 + x_2 + ... + x_T)/T$. Den andra metoden brukar kallas det *naiva tillvägagångssättet* [3] och tar fram en prognos enligt $\hat{y}_{T+k} = x_T$. Dessa ger såklart väldigt begränsade resultat och används ofta mest som en baslinje för att avgöra om andra mer komplexa metoder är vettiga eller ej. Ett exempel på en mer komplex metod är *Autoregressive integrated moving average* (ARIMA) [4] som ej förklaras mer i detalj här.

Att avgöra hur väl en metod för prognosframtagning presterar kan göras på många sätt. I denna rapport har metoden *Mean absolute error* (MAE) använts [5]. Den är

definierad enligt ekvation (2.1),

$$MAE = \frac{1}{T}\sum_{i=1}^{T}|y_i - \hat{y}_i| \tag{2.1}$$

där $\hat{y}_i$ motsvarar den i:te prognosen, och y_i motsvarar dess faktiska värde. MAE motsvarar den genomsnittliga noggrannheten för alla prognoser i. $MAE = 0$ är det absolut bästa eftersom detta innebär att alla prognoser är lika med de värden som förväntades.

På senare tid har det visat sig att olika maskininlärningsalgoritmer har i många sammanhang kunnat ge bättre resultat än de traditionella metoderna. Detta har i huvudsak möjliggjorts av två anledningar. Den första är på grund av den stora mängd historisk data som nu finns tillgänglig. Den andra är förbättrad datorprestanda [6]. Maskininlärning beskrivs mer i detalj i avsnitt 2.2.

2.2 Maskininlärning

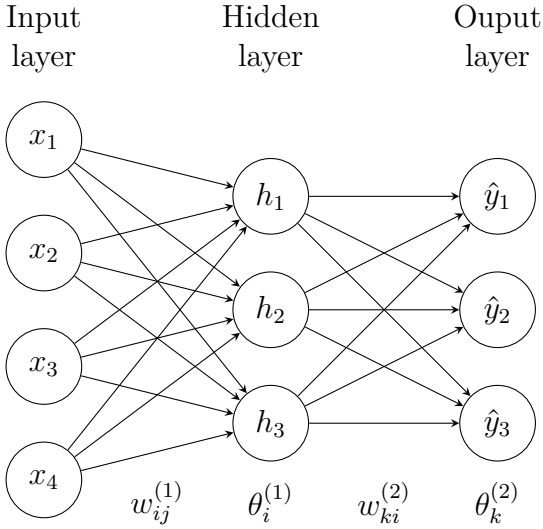
Maskininlärning är studien om algoritmer och data som datorsystemet använder för att förbättra sin lärdom om en specifik uppgift. Thomas M. Mitchell ger följande definition av maskininlärning [7]:

> "Ett datorprogram sägs lära av erfarenhet E givet en uppgift T och ett prestandamått P om dess prestanda på uppgiften T, mätt med P, ökar med erfarenhet E"

Några exempel på vad uppgiften T skulle kunna motsvara är klassifikation, regression och maskinöversättning [8]. Prestandamåttet P är ett mått på hur väl datorsystemet har lärt sig den givna uppgiften T (exempelvis MAE som nämnts tidigare). Erfarenheten E syftar på den typ av data som datorsystemet har tillgång till under inlärningen. Träningsdatan bör vara stor i mängd om systemet ska förväntas generalisera denna till ny testdata som den inte sett tidigare. Det finns olika typer av erfarenheter E, exempelvis *supervised learning* och *unsupervised learning*. I den förstnämnda består datan av färdiga exempel på hur en viss indata är associerad med en given utdata. I den sistnämnda är endast indatan given och maskininlärningsalgoritmen måste själv försöka hitta eventuella mönster i datan [8].

2.2.1 Feedforward Neural Network

En typ av maskininlärningsteknik är Artificiella Neurala Näverk (ANN), som främst använts i detta projekt. ANN är inspirerade av biologiska neurala nätverk som utgör våra hjärnor [9]. Eftersom det finns tillgång till både indata (trafikflödet i en viss tidpunkt) och den förväntade utdatan (trafikflödet en viss tid i framtiden) så går det att applicera *supervised learning*. I detta avsnitt introduceras ett så kallat *Feedforward Neural Network* (FFNN) vilket innebär att alla vikter mellan neuroner inte formar några cykler (se figur 2.1).



Figur 2.1: Ett FFNN med ett gömt lager. x_j motsvarar ingångssignalerna, h_i motsvarar tillstånden av neuronerna i det gömda lagret och $\hat{y}_k$ motsvarar utsignalerna. $w_{ij}^{(1)}$ motsvarar vikten mellan x_j och h_i. På samma sätt är $w_{ki}^{(2)}$ vikten mellan h_i och $\hat{y}_k$. $\theta_i^{(1)}$ är tröskelvärdet för neuron h_i i det gömda lagret. $\theta_k^{(2)}$ är tröskelvärdet för neuron $\hat{y}_k$ i utgångslagret.

Ett ANN tar emot data i en dimension $\boldsymbol{x}^{(\mu)} \epsilon \mathbb{R}^n$ och genererar ett resultat i en potentiellt olik dimension $\hat{\boldsymbol{y}}^{(\mu)} \epsilon \mathbb{R}^m$ med hjälp av en uppsättning parametrar. Exponenten μ illustrerar att nätverket tar emot flera uppsättningar av indata under inlärningsprocessen. Resultatet jämförs sedan med det förväntade värdet $\boldsymbol{y}^{(\mu)} \epsilon \mathbb{R}^m$ för den givna indatan, och uppdaterar parametrarna utifrån eventuella fel. Förhoppningen är då att nästa gång samma indata skickas in så skall nätverket ge ett svar som är lite närmare det förväntade värdet. Upprepning av denna process är vad som menas med att nätverket lär sig över tid [8]. Inlärningsprocessen beskrivs nedan utifrån ett simpelt FFNN som är illustrerat i figur 2.1.

Frampropagering

Första steget är räkna ut utgångsvärden $\hat{y}_k^{(\mu)}$ baserat på givna ingångsvärden $x_j^{(\mu)}$. Detta görs genom att propagera framåt genom nätverket [10]. I detta fall innehåller nätverket ett gömt lager, och första steget blir därför att räkna ut tillstånden hos de gömda neuronerna h_i. Detta görs genom att räkna ut den viktade summan av alla insignaler $x_j^{(\mu)}$ baserat på vikterna $w_{ij}^{(1)}$ samt tröskelvärdena $\theta_i^{(1)}$ enligt (2.2) [10].

$$h_i = g(\sum_{j=1}^{4} w_{ij}^{(1)} x_j^{(\mu)} - \theta_i^{(1)}) \tag{2.2}$$

Med de gömda tillstånden h_i uträknade kan nu utsignalerna $\hat{y}_i^{(\mu)}$ räknas ut på samma sätt enligt (2.3).

$$\hat{y}_k^{(\mu)} = f(\sum_{i=1}^{3} w_{ki}^{(2)} h_i - \theta_k^{(2)}) \tag{2.3}$$

Funktionerna f och g är så kallade aktiveringsfunktioner. Den vanligaste aktiveringsfunktionen kallas ReLU och är definierad som $p(x) = max(0, x)$ [10]. f och g skulle båda kunna vara lika med p men inte nödvändigtvis. Den aktiveringsfunktion som väljs i varje lager kan alltså vara olika [10].

Kostnadsfunktion

Då nätverket tränas skickar man in olika uppsättningar av insignaler $\boldsymbol{x}^{(\mu)}$ och har för varje uppsättning μ lagrat det förväntade värdet $\boldsymbol{y}^{(\mu)}$ (*supervised learning*). $\boldsymbol{x}^{(\mu)}$ skickas alltså in i nätverket, sedan räknas $\hat{\boldsymbol{y}}^{(\mu)}$ ut genom frampropagering som sedan jämförs med det förväntade värdet $\boldsymbol{y}^{(\mu)}$. Nästa steg är att ta reda på hur långt ifrån utsignalerna är ifrån det förväntade värdet med någon typ av kostnadsfunktion. Ett exempel på en sådan funktion är den så kallade kvadratiska kostnadsfunktionen [10] som är definierad enligt (2.4) för det givna nätverket i figur (2.1).

$$H = \frac{1}{2} \sum_{j=1}^{3} (y_j^{(\mu)} - \hat{y}_j^{(\mu)}) \tag{2.4}$$

Man kan se i ekvation (2.4) att funktionen H minimeras (är lika med noll) precis då $\hat{\boldsymbol{y}}^{(\mu)} = \boldsymbol{y}^{(\mu)}$, vilket är precis det nätverket försöker lära sig. Målsättningen är alltså att minimera denna funktion.

Bakpropagering

Minimeringen av kostnadsfunktionen görs med en algoritm som kallas *bakpropagering* [10] som innebär att man istället propagerar baklänges genom nätverket och uppdaterar alla vikter och tröskelvärden. Denna algoritm minimerar kostnadsfunktionen med en teknik som kallas *gradient descent* [10]. Kortfattat så tas den partiella derivatan av funktionen H med respekt till alla vikter ($w_{ij}^{(1)}$ och $w_{ki}^{(2)}$) samt alla tröskelvärden ($\theta_i^{(1)}$ och $\theta_k^{(2)}$) var för sig. Denna derivata säger då i vilken riktning som dessa parametrar skall uppdateras för att komma närmare ett minimum av H. Efter varje uppdatering så görs frampropagering igen vilket resulterar i ett nytt värde på $\hat{\boldsymbol{y}}^{(\mu)}$ som förhoppningsvis dragit ned kostnadsfunktionen en liten bit. Denna process upprepas tills att kostnadsfunktionen är minimerad. Om uppsättningen indata μ som nätverket har tillgång till för inlärning är tillräckligt stor så är nu förhoppningen att det lärt sig att generalisera detta till helt nya uppsättningar av indata som den ej sett tidigare.

Hyperparametrar

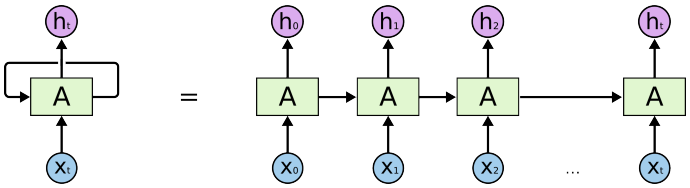
Utöver de parameterar som redan nämnts så finns en del andra parametrar som inte används i själva inlärningsprocessen. Istället är dessa parametrar förinställda innan inlärningen börjar och brukar kallas *hyperparametrar* [8]. Detta kan exempelvis vara antalet gömda lager och antalet neuroner i varje lager. Exempelvis kan fler gömda lager leda till att nätverket har förmågan att lära sig mer komplexa mönster. Detta är vad som kallas *deep learning* och är i nuläget den viktigaste tekniken inom maskininlärning och AI generellt [8]. Fler lager innebär såklart också att inlärningsprocessen blir långsammare. En annan hyperparameter är något som kallas epoker. En epok innebär att nätverket har gått igenom alla uppsättningar μ av indata en gång. Ofta räcker inte detta för att minimera konstandsfunktionen utan antalet epoker är ofta fler än en. För många epoker leder till andra konsekvenser som att inlärningsprocessen blir långsammare och något som kallas *overfitting* som minskar nätverkets förmåga att generalisera till ny data [11]. Att välja dessa parametrar är därför någonting som ofta görs genom att experimentera och försöka hitta vad som passar bäst för ett specifikt syfte.

2.2.2 Recurrent Neural Network

Då ett FFNN tränas för att göra prognoser i en tidsserie vet den endast om det uppmätta värdet i nuvarande tidssteg. Med hjälp av ett uppmätt värde vid tidpunkt t som benämns x_t så tränas nätverket på att bestämma värdet k steg in i framtiden som benämns $\hat{y}_{t+k}$. Nätverket förväntas nu kunna göra en sådan prognos för alla värden på t. Man kan nu inse att detta blir problematiskt ifall flera uppmätta värden x_t med ungefär samma värde förväntas ge en prognos $\hat{y}_{t+k}$ som är väldigt olika. Kortfattat så saknar ett FFNN kontext för var i tidsserien den befinner sig, samt är ovetande om tidigare uppmätta värden i tidsserien [12]. Ett *Recurrent Neural Network* (RNN) löser detta problem genom att introducera möjligheten att ha vikter som skapar cykler (se vänster i figur 2.2). Dessa cykler ger möjligheten för information att flöda baklänges vilket ger upphov till ett sorts minne [12]. Detta gör att en prognos $\hat{y}_{t+k}$ kan tas fram baserat på de N tidigare uppmätta värdena $\{x_t, x_{t-1}, ..., x_{t-N}\}$. Bilden till höger i figur 2.2 visar hur detta går till ifall man "rullar ut" nätverket med tiden. Att träna detta nätverk görs också med bakpropagering som nämndes i avsnitt 2.2.1. Mer specifikt används något som kallas *Backpropogation through time* (BPTT) [13]. RNN är alltså användningsbart för data som inkommer i en tidssekvens. Exempel på användsningsområden är just tidsserieprognoser och något som kallas maskinöversättning. Maskinöversättning innebär att en dator översätter text eller tal från ett språk till ett annat. Detta är något som exempelvis Google utnyttjar för deras tjänst Google Översätt [14].

Ett problem med ANN i allmänhet är något som heter *Vanishing Gradient Problem* [15]. Kortfattat innebär detta att nätverket slutar lära sig då man inför fler lager i nätverket. Ett RNN lider extra mycket från detta eftersom antal lager beror på antal tidssteg N som man vill basera sin prognos på. För stora värden på N slutar alltså nätverket att lära sig. En typ av RNN vid namnet *Long Short-Term Memory*

(LSTM) har därför introducerats som lösning på detta problem och har i huvudsak använts i detta projekt [12].



Figur 2.2: Vänster: Ett RNN där man ser att vikten åker tillbaka in i samma neuron (cykel). Höger: Strukturen av samma RNN efter att det "rullats ut" med tiden. [12]

2.3 Databashantering

MongoDB är ett dokumentorienterat databasprogram och klassificeras som ett dataprogram inom NoSQL [16]. Till skillnad från relationsdatabaser, där data lagras i tabellform, lagras data inom NoSQL på annat sätt och är främst avsedd för dokument. MongoDB lagrar varje dokument som BSON (Binary JSON) vilket är en binär representation av JSON-format (JavaScript Object Notation) på dokumenten (se figur 2.3).

```
{
     "firstName": "Jason",
     "lastName": "Smith",
     "age": 25,
     "address": {
         "streetAddress": "21 2nd Street",
         "city": "New York",
         "state": "NY",
         "postalCode": "10021"
     },
     "phoneNumber": [
         { "type": "home", "number": "212 555-1234" },
         { "type": "fax", "number": "646 555-4567" }
     ],
     "newSubscription": false,
     "companyName": null
 }
```

Figur 2.3: Exempel på JSON-format som beskriver en person.

3. Metod

Eftersom maskininlärning skall utnyttjas i detta projekt så är det kritiskt att ha en stor mängd relevant data. Den data som i detta fall anses vara relevant är uppmätt trafikflöde med jämna mellanrum på flera vägar i Göteborg. Anledningen till detta är att produkten är beställd av ett företag som är stationerat i den regionen. Hur datainsamlingen har gått till förklaras därför i avsnitt 3.1.

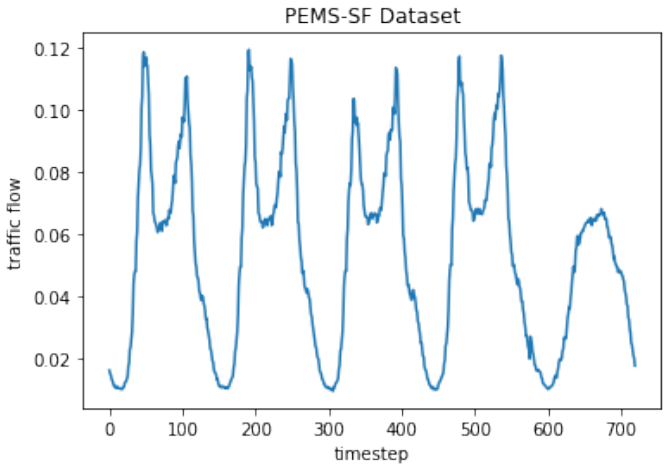
Med data tillgängligt i formatet av en tidsserie så blir nästa steg att analysera denna data, som beskrevs i avsnitt 2.1 om tidsserier. Hur detta gick till beskrivs i avsnitt 3.2. Efter analysen är gjord så skall maskininlärning appliceras med hjälp av tidsserien (enligt teorin från avsnitt 2.2) för att ta fram prognoser för trafikflödet. Det tillvägagångssätt som använts för att åstadkomma detta beskrivs därför i avsnitt 3.3. Till sist beskrivs hur en webbapplikation har skapats för att visualisera prognoserna i avsnitt 3.4.

3.1 Datainsamling

Då det efter mycket sökande verkar som att det inte finns någon lättillgänglig lagrad data för trafikflödet i Göteborg så har egen data samlats in i realtid. Detta gjordes med hjälp av HERE Maps API [17] och förklaras i avsnitt 3.1.2. För att under tiden som denna data samlades in kunna påbörja testning av dataanalys samt prognosframtagningar så användes befintlig trafikdata från San Francisco. Denna datamängd beskrivs kortfattat i avsnitt 3.1.1.

3.1.1 San Francisco Datamängd

Trafikdatan från San Francisco fanns tillgänglig gratis att ladda ned ifrån UCI Machine Learning Repository [18], och kommer ursprungligen ifrån California Department of Transportation [19]. Den består av 144 mätningar av trafikflödet per dag (en mätning var tionde minut) i 440 dagar från 963 olika vägsegment. Trafikflödet beskrivs som ett flyttal mellan noll och ett. Denna data har i huvudsak använts för att dataanalysen samt prognosframtagningar skulle kunna påbörjas tidigt under projektets gång. Då den insamlade datan i Göteborg fanns tillgänglig efter ett par veckors insamling så användes datan ifrån San Francisco endast som typexempel för hur statistiken bör se ut. I figur 3.1 visas trafikflödet för de fem första dagarna. Y-axeln motsvarar det genomsnittliga trafikflödet hos alla 963 vägsegment. X-axeln motsvarar det tidssteg då trafikflödet uppmättes. Det är tio minuters intervall mellan varje tidssteg. Ett tydligt mönster kan observeras för trafiken för de första fyra dagarna. I början av dagen när folk åker till jobbet ökar trafikflödet, sen lugnar det ner sig under dagen fram tills dess att folk åker hem igen. Den sista dagen som visas följer inte samma mönster eftersom det motsvarar en helgdag.



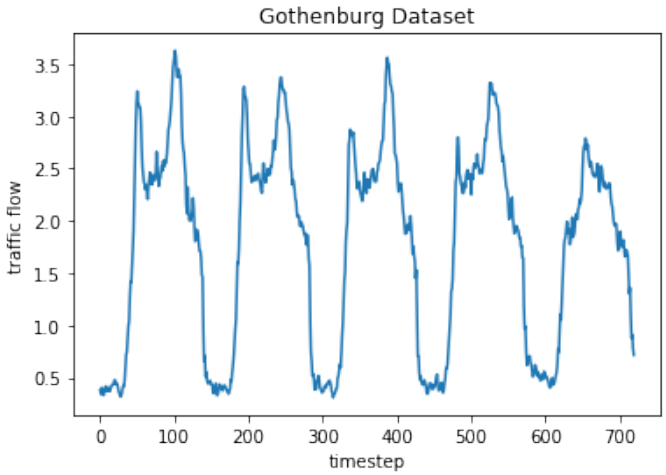
Figur 3.1: Det genomsnittliga trafikflödet under de första fem dagarna baserat på datan från San Francisco [19].

3.1.2 HERE Maps API

HERE Maps API används för att hämta data över Göteborg. Även denna data består av 144 mätningar per dag men endast från 811 olika vägsegment i detta fall. Datan hämtas med en HTTP GET request som innehåller ett antal parametrar med information om vad för returvärden som önskas. Parametrarna som skickas med i förfrågan är en koordinat plus en radie som bildar ett cirkulärt område där datan ska hämtas från. En extra parameter, "shape", skickas även med vilket resulterar i att alla koordinater för vägarna inkluderas i svaret.

Svaret som returneras är i JSON-format (JavaScript Object Notation), vilket underlättar hanteringen av datan då strukturen för JSON är väldigt lik Pythons dictionary-struktur. En annan fördel med att hantera datan som JSON är att MongoDB lagrar all data som BSON, vilket är en binär representation av JSON. Datan som svaret innehåller är uppdelat i större vägar, som i sig är uppdelat i mindre segment (enstaka sensorer) av vägarna. Varje segment innehåller sedan data som trafikflöde, medelhastighet och riktning. Trafikflödet är ett flyttal mellan 0.0 och 10.0, som beskriver hur trafikerad vägen är. Ett högre tal innebär att vägen är mycket trafikerad för stunden, medan ett lägre tal innebär att vägen är mindre trafikerad. Riktning beskriver segmentets riktning och representeras av antingen ett plus- eller minustecken.

Skriptet som hämtar trafikdatan implementerades i Python, och exekveras automatiskt var tionde minut för att sedan lagra datan i databasen. Databasen som används är en molnbaserad version av MongoDB kallad MongoDB Atlas. Detta resulterar i, var tionde minut, en rad i databasen som representerar trafiksituationen vid det tillfället. I figur 3.2 visas det genomsnittliga trafikflödet från 811 vägsegment i Göteborg på samma sätt som visades i figur 3.1 för San Francisco. Den visar samma mönster, fast med mer brus, vilket innebär att den insamlade datan är logisk.



Figur 3.2: Det genomsnittliga trafikflödet under de första 5 dagarna baserat på den insamlade data från Göteborg med HERE Maps API [17].

3.2 Dataanalys

Analys av trafikdatan är en viktig del av projektet eftersom det gör att olika tidsberoende mönster kan identifieras. Detta kan i sin tur underlätta och förbättra appliceringen av vår maskininlärningsalgoritm. Exempelvis behöver det utredas ifall tidsserien innehåller några periodiska tidsmönster och hurvida den inte är helt stokastiskt. Existerar inga tidsmönster så kan ej noggranna prognoser förväntas. Denna analys har gjorts med programmeringsspråket Python. Nedan listas några av de viktigaste Python-bibliotek som utnyttjats för dataanalysen:

- **NumPy** [20]: Möjligör hantering av stora multidimensionella listor och matriser. Innehåller exempelvis en stor samling av matematiska funktioner för att smidigt operera dessa listor.
- **Pandas** [21]: Ett bibliotek för datamanipulation och analys. Den erbjuder datastrukturer och operationer för att hantera numeriska tabeller och tidsserier.
- **Sickit-learn** [22]: Ett bibliotek för bearbetning av datamängder samt implementering av olika maskininlärningsalgoritmer.
- **Matplotlib** [23]: En 2D-grafikmiljö som använts för uppritning av olika diagram som visualiserar trafikdatan på olika sätt.

3.3 Trafikprognoser

I detta projekt har endast LSTM-nätverk med olika hyperparametrar testats för att ta fram trafikprognoser. Implementationen av LSTM-nätverket gjordes med Python-biblioteket Keras [24]. Detta är ett API som ligger ovanpå ett mer välkänt bibliotek med namnet TensorFlow [25], som används för att implementera olika typer av neurala nätverk.

Dataformat

Den data som nu finns tillgänglig är alltså i formatet av en tidsserie. Antal tidssteg av uppmätt trafikflöde ökar hela tiden och benämns därför T i nuläget. För varje tidssteg har trafikflödet vid 811 olika vägsegment uppmätts. Tidsserien blir alltså en tvådimensionell matris av uppmätt data enligt (3.1).

$$X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}, \quad \boldsymbol{x}_t \in \mathbb{R}^{811} \tag{3.1}$$

Där varje element $\boldsymbol{x}_t$ är en vektor av storlek 811 eftersom varje tidssteg har 811 upmätta värden. Dimensionen av X blir således $T \times 811$.

Inlärningsprocessen

Nedan beskrivs den inlärningsprocess som använts för att träna olika LSTM-nätverk på prognosframtagningar givet den tidsserie av data som beskrevs ovan. Detta är baserat på flera olika artiklar skrivna om ämnet av Jason Brownlee [26].

Flera LSTM-nätverk skall tränas på att ta fram en prognos av trafikflödet för alla 811 vägsegment $k \geq 1$ tidssteg in i framtiden baserat på de $N \geq 1$ tidigare uppmätta värdena. Trafikprognosen benämns $\hat{\boldsymbol{y}}_{t+k} \in \mathbb{R}^{811}$ och blir också en vektor av storlek 811 eftersom alla vägsegment skall få en prognos. Framtagningen av denna beskrivs matematiskt enligt (3.2).

$$\{\boldsymbol{x}_{t-N+1}, \boldsymbol{x}_{t-N+2}, \ldots, \boldsymbol{x}_t \mid N \leq t \leq T-k\} \Longrightarrow \hat{\boldsymbol{y}}_{t+k} \tag{3.2}$$

Eftersom *supervised learning* skall appliceras så saknas nu bara de förväntade värdena $\boldsymbol{y}_{t+k} \in \mathbb{R}^{811}$. Under inlärningsprocessen har vi endast tillgång till de element i matrisen X. Det förväntade värdet (som benämns $\boldsymbol{y}_{t+k}$ för att matcha teorin i avsnitt 2.2) blir alltså $\boldsymbol{x}_{t+k}$. De uppsättningar av indata som LSTM-nätverket tränas med blir således alla möjliga tidsordnade delmängder av storlek N ifrån matrisen X (med andra ord alla möjliga värden på t i (3.2)). Detta illustreras i (3.3).

$$\begin{aligned}
\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} &\Longrightarrow \hat{\boldsymbol{y}}_{N+k} \\
\{\boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_{N+1}\} &\Longrightarrow \hat{\boldsymbol{y}}_{(N+1)+k} \\
\{\boldsymbol{x}_3, \boldsymbol{x}_4, \ldots, \boldsymbol{x}_{N+2}\} &\Longrightarrow \hat{\boldsymbol{y}}_{(N+2)+k} \\
&\vdots \\
\{\boldsymbol{x}_{T-N-k+1}, \boldsymbol{x}_{T-N-k+2}, \ldots, \boldsymbol{x}_{T-k}\} &\Longrightarrow \hat{\boldsymbol{y}}_T
\end{aligned} \tag{3.3}$$

Notera att inlärningsprocessen endast går fram till $\hat{\boldsymbol{y}}_T$ eftersom det inte existerar några förväntade värden för tidssteg över T eftersom detta är framtida trafikflöden. Antalet uppsättningar μ av indata blir således $T - N - k$ (d.v.s antal värden som t kan anta i (3.2)). Det är prognoser som går förbi T som är intressanta efter nätverket har lärt sig av den redan existerande datan. Detta är vad som sedan visas grafiskt i webbapplikationen. Detta görs då enligt (3.4).

$$\{\boldsymbol{x}_{T-N+1},\, \boldsymbol{x}_{T-N+2},\, \ldots,\, \boldsymbol{x}_T\} \Longrightarrow \hat{\boldsymbol{y}}_{T+k} \tag{3.4}$$

Heltalen N och k blir nu ytterliggare parametrar som måste väljas. För varje val på dessa två parametrar så tränas en separat LSTM-modell. Heltalet N motsvarar som sagt hur många tidigare uppmätta värden som prognosen skall baseras på. Detta motsvarar antal tidssteg i LSTM-nätverket. Fler tidssteg gör att nätverket blir långsammare att träna, men ger möjligtvis bättre resultat. Heltalet k motsvarar hur långt in i framtiden som nätverket skall tränas att ta prognoser för. I detta projekt har 4 olika LSTM-nätverk tränats. För alla dessa har N valts till 12 av empiriska skäl. k har valts till 1, 6, 72 och 144. Dessa motsvarar prognoser för 10 minuter, 1 timma, 12 timmar respektive 24 timmar in i framtiden. Ungefär 80% av alla tidssteg i matrisen X har använts under inlärningsprocessen. Resterande 20% har använts som test data för att efter inlärningen avgöra hur väl LSTM-nätverket presterar på data som den ej sett tidigare. Alla nätverk har tränats i 30 epoker vilket innebär att de har sett alla $T - N - k$ uppsättningar av indata 30 gånger.

3.4 Webbapplikation

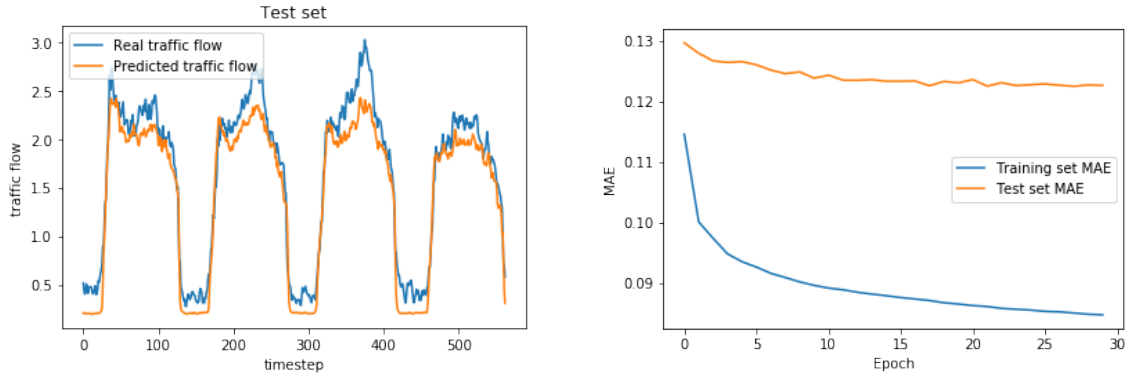
För att kunna illustrera resultatet av den insamlade datan, samt den förutsedda datan skapades en webbapplikation. Ramverket Flask användes för själva back-end delen av webbapplikationen. Flask är ett webbramverk i Python som fungerar som en webbserver, och möjliggör användningen av Pythonscript i själva applikationen. Front-end delen är skriven i HTML, CSS och Javascript samt med ramverken Bootstrap 4 och Google Maps API. Den är sedan uppdelad i två olika vyer, en i form av ett färgdiagram (Känt som heatmap på engelska) där punkter skapas genom att varje nods trafikflöde delas på antalet noder för vägen. Närliggande noder slås sedan ihop till en mörkare punkt där varje nods vikt är dess flöde (se figur 4.6). Den andra vyn är skapad av färgade linjer där varje väg övertäcks av streck, vars färg är baserad på vägens trafikflöde. Högre flöde resulterar i en mörkare färg (se figur 4.7).

4. Resultat

Detta projekt har som tidigare nämnts bestått av tre delar. Den första delen handlade om insamling av relevant data vilket förklarades i avsnitt 3.1. Den andra delen handlade om analys av denna data samt prognosframtagningar med hjälp av maskininlärning. Hur exakta dessa prognoser blev redovisas därför i avsnitt 4.1. Den tredje delen handlade om att skapa en webbapplikation som visualiserar trafikflödet vid en valfri tid, även inkluderat en viss tid in i framtiden på ett färgdiagram med hjälp av prognosframtagningen. Detta redovisas i avsnitt 4.2.

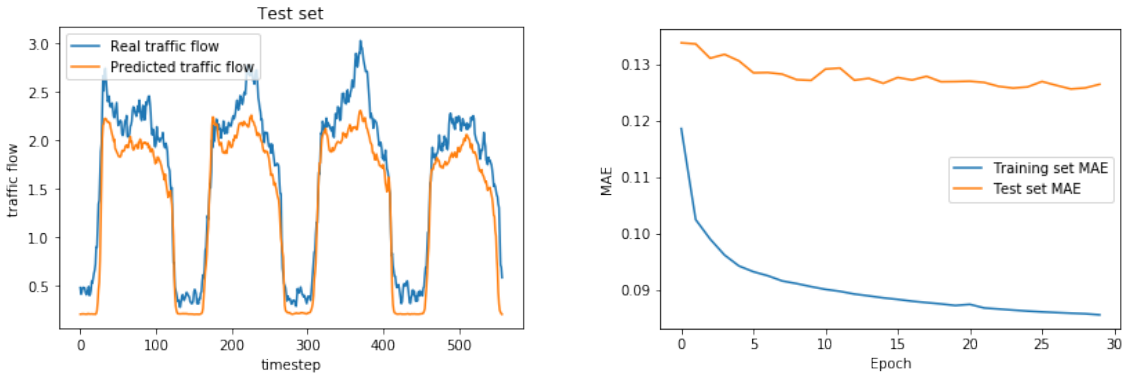
4.1 Trafikprognoser

Fyra olika LSTM-nätverk har tränats på att göra prognosframtagningar. Det första nätverket har tränats på att ta fram prognoser för tio minuter in i framtiden. I figur 4.1 (a) visas hur detta nätverk presterade på test datan genom att jämföra prognoser för varje tidssteg (orange linje) med den faktiska datan för samma tidssteg (blå linje). Figur 4.1 (b) visar uppmätt MAE för både träningsdata (blå linje) och testdata (orange linje) för varje epok under inlärningsprocessen. Det uppmätta MAE för sista epoken för testdatan är den noggrannhet som kan förväntas då riktiga prognoser görs för framtida tidssteg. I figur 4.2, 4.3 och 4.4 visas motsvarande resultat fast för prognoser som togs 1 timma, 12 timmar respektive 24 timmar in i framtiden. För att illustrera skillnaden på den data som samlats in över Göteborg och den redan existerande datan från San Francisco så har även ett nätverk tränats för detta. I figur 4.5 visas därför resultatet för prognoser som tagits 10 minuter in i framtiden baserat på trafikdatan i San Francisco.



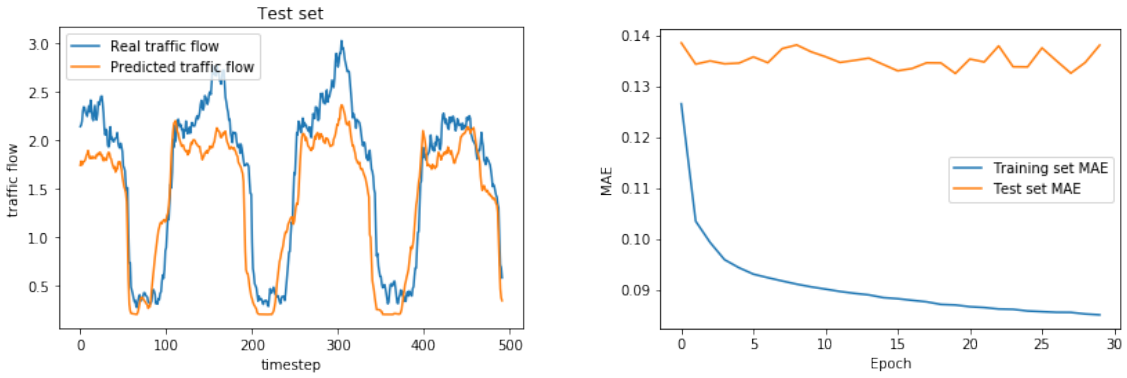
(a) **(b)**

Figur 4.1: Resultat för LSTM-nätverk som tagit prognoser för 10 minuter in i framtiden baserat på de 12 tidigare tidsstegen.



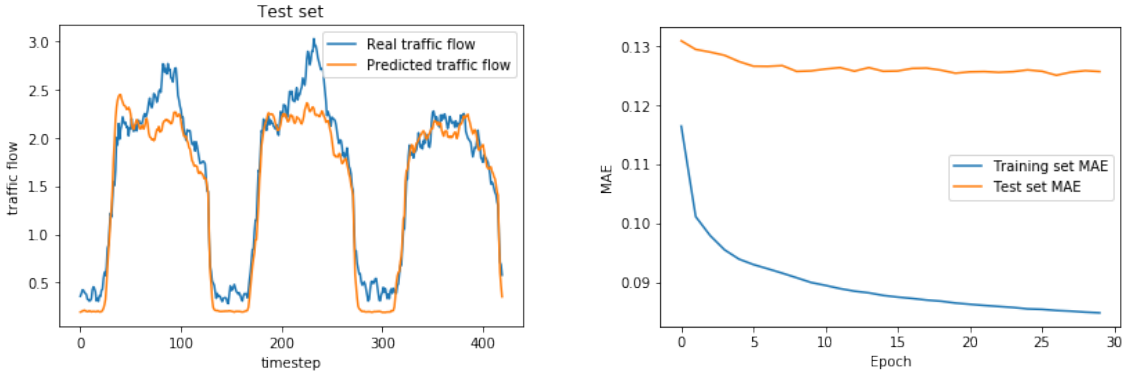
(a) **(b)**

Figur 4.2: Resultat för LSTM-nätverk som tagit prognoser för 1 timma in i framtiden baserat på de 12 tidigare tidsstegen.



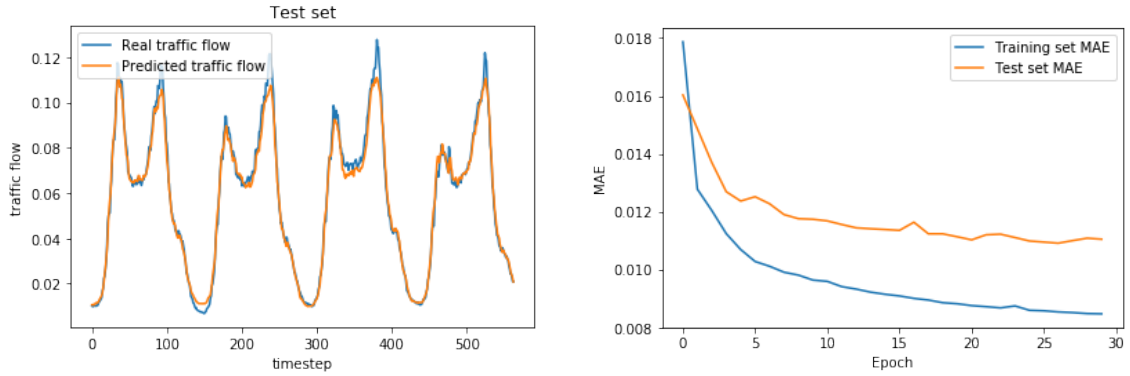
(a) **(b)**

Figur 4.3: Resultat för LSTM-nätverk som tagit prognoser för 12 timmar in i framtiden baserat på de 12 tidigare tidsstegen.



(a) **(b)**

Figur 4.4: Resultat för LSTM-nätverk som tagit prognoser för 24 timmar in i framtiden baserat på de 12 tidigare tidsstegen.

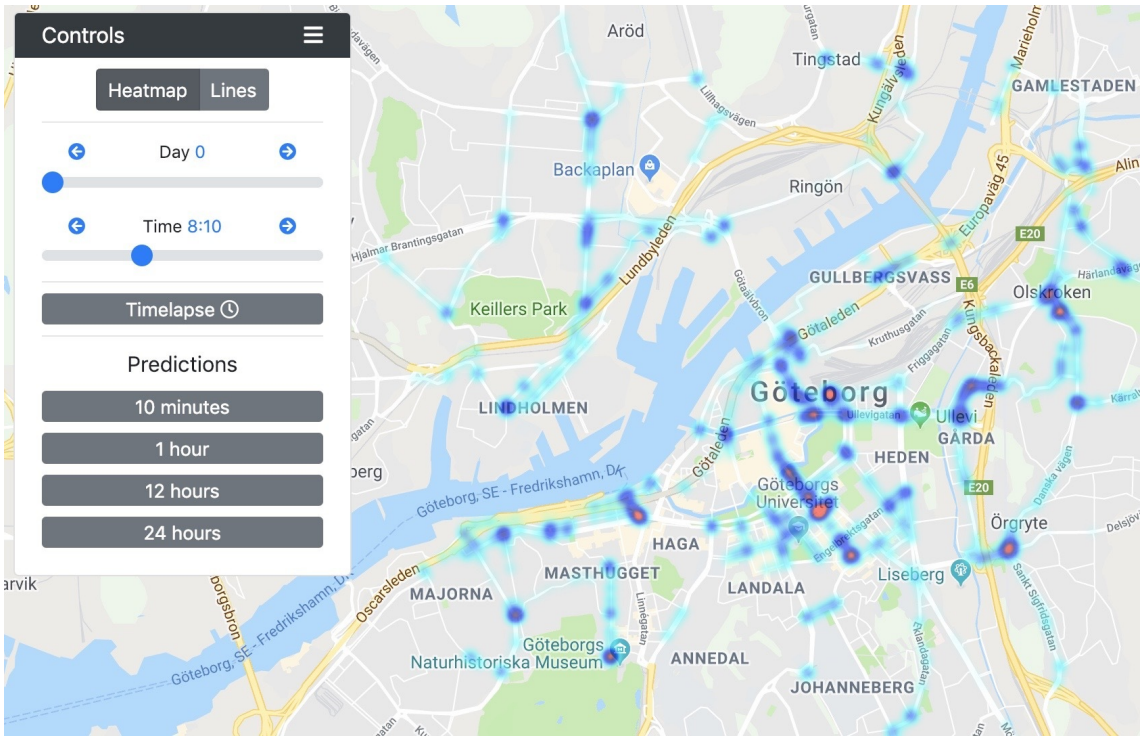


(a) **(b)**

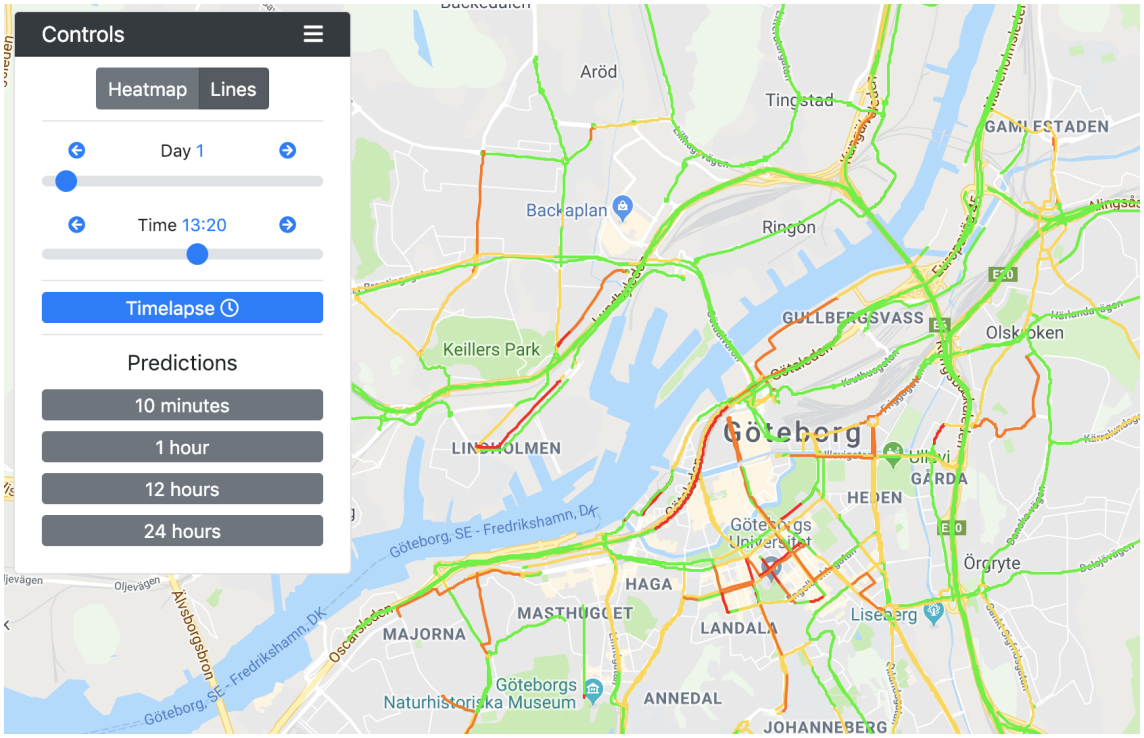
Figur 4.5: Resultat för LSTM-nätverk som tagit prognoser för 10 minuter in i framtiden baserat på de 12 tidigare tidsstegen. Detta resultat är baserat på trafikdatan från San Francisco.

4.2 Webbapplikationen

Resultatet illustreras med hjälp av en webbapplikation med två vyer. Den första vyn är ett färgdiagram (se figur 4.6) och den andra består av färgade linjer (se figur 4.7). Båda vyerna fungerar på liknande sätt under ytan men illustrerar datan på olika sätt. Varje vy har en kontrollruta med "sliders" för att växla mellan de olika tidsstämplarna, samt knappar för att växla mellan de olika vyerna och till den förutsedda datan. Det finns även en knapp för att starta eller stoppa ett time-lapseläge, där applikationen själv går en tidsstämpel framåt varje sekund.



Figur 4.6: Trafikflödet i Göteborg klockan 08:10, dag 0, illustrerat med ett färgdiagram.



Figur 4.7: Trafikflödet i Göteborg klockan 13:20, dag 1, illustrerat med färgade linjer och med timelapseläget aktiverat.

5. Diskussion

Avslutningsvis kommer detta avsnitt diskutera nuvarande system samt några förslag på framtida arbete. I avsnitt 5.1 diskuteras datainsamlingen. I avsnitt 5.2 diskuteras dataanalysen och prognosframtagningen

5.1 Datainsamling

Trafikdatan som hämtas lagras i MongoDB Atlas, som är en molnbaserad databas. Fördelen med MongoDB Atlas, istället för en lokal databas, är att enkelt ha tillgång till datan oberoende av vilken enhet som används. Nackdelarna med MongoDB Atlas är dels att mängden data som kan lagras är begränsad till 512 MB (Megabyte), och dels att kommunikationen med databasen blir långsam när en stor mängd data behöver hanteras. I nuläget lagrar databasen 253.2 MB data och det tar flera minuter att hämta all data från databasen. Tiden det tar att kommunicera med databasen är inte optimal och utgör ett problem när ny data behöver analyseras. Kommunikationen verkar dessutom försämras i och med att den lagrade datan blir större i storlek.

Problemet med databashanteringen kan möjligtvis lösas genom att använda en lokal databas istället för en molnbaserad databas, men detta behöver undersökas vidare. Alternativt kan aggregerade förfrågningar till databasen användas för att endast hämta ny data som inte redan används lokalt.

5.2 Dataanalys & Prognosframtagningar

I figur 4.5 a) ser man att prognoserna för San Francisco blir betydligt mer noggranna än prognoserna för Göteborg. I figur 4.5 b) bekräftas även detta numeriskt då de uppmätta värdena för MAE ligger på ungefär 0.012 jämfört med 0.125 för Göteborg, vilket är drygt tio gånger mindre. Man ser även att MAE för testdatan är betydligt närmare den uppmätta MAE för träningsdatan. Detta innebär att detta LSTM-nätverk har lyckats generalisera det den lärt sig till osedd data mycket bättre. Anledningen till denna skillnad har med hur trafikdatan ser ut. Två skillnader är framför allt avgörande här och har listats nedan.

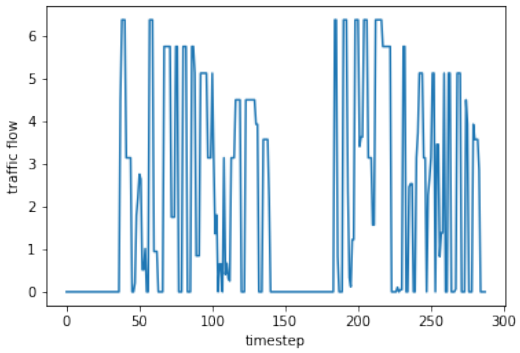
- **Begränsad datamängd**

  Den insamlade datan i Göteborg är inte tillräcklig. Om LSTM-nätverket skall förväntas göra noggranna prognoser för data som den ej sett tidigare så behöver den ha tränats på en mycket stor mängd data. I nuläget finns endast 25 dagar av insamlad data i Göteborg, medan det fanns 440 dagar av data för San

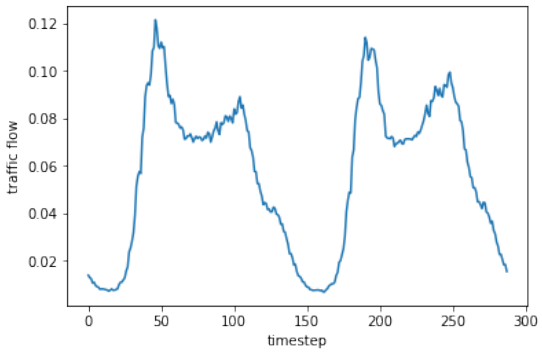
Francisco. Det skall även nämnas att en stor del av den data som samlats in över Göteborg togs under julledigheterna. Denna data skiljer sig mycket från den vanliga datan (ungefär som vardagar och helgdagar skiljer sig).

- **Inkonsekvent data**

  Att trafikdatan över Göteborg är mer brusig/inkonsekvent visades redan i avsnitt 3 om man jämför figur 3.2 med figur 3.1. Detta var dock ett medelvärde för alla vägsegment vilket inte illustrerar bruset tydligt. I figur 5.1 visas trafikflödesskillnaden för ett slumpmässigt valt vägsegment från Göteborg (a) och San Francisco (b) under två dagar. Här syns mycket tydligt att datan i Göteborg är väldigt brusig. Notera dock att alla 811 vägsegment inte visade lika mycket brus som det vägsegment som valdes här. Poängen är att datan i San Francisco förmodligen har bearbetats på olika sätt för att jämna till den så att mer noggranna prognoser kan göras. Exempel på bearbetningar som skulle kunna göras är en metod som kallas *glidande medelvärde* [27]. Detta skulle ta bort mycket av det brus som i nuläget existerar i tidsserien. Denna typ av bearbetning är en del av den fortsatta dataanalys som behöver göras i framtida projekt (med bibliotek som nämndes i avsnitt 3.2). En möjlig anledning till den inkonsekventa datan är att de 811 vägsegment som valts i Göteborg är centralt positionerade. Risken är att trafiken hinner förändras ganska fort inom de 10 minuters intervall som trafikflödet i nuläget mäts. Detta gör att de uppmätta trafikflödena kan hoppa upp och ner ganska plötsligt som ses i figur 5.1 a).



(a) Trafikflödet under två dagar för ett slumpmässigt valt vägsegment av de 811 olika vägsegmenten i Göteborg.



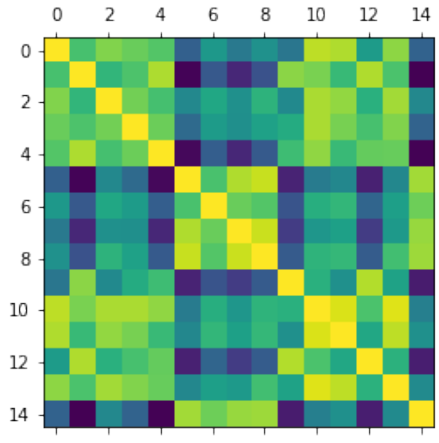
(b) Trafikflödet under två dagar för ett slumpmässigt valt vägsegment av de 963 olika vägsegmenten i San Francisco.

Figur 5.1: Trafikflödesskillnad mellan datan från Göteborg och San Francisco för ett vägsegment under två dagar.

Utöver bättre bearbetning av datamängden för Göteborg så finns en del andra åtgärder som kan appliceras för att öka exaktheten för de tränade LSTM-nätverken. Dessa åtgärder kan alltså dessutom användas för San Francisco-datan för att ytterligare förbättra dessa prognoser. I nuläget tränas varje LSTM-nätverk på att ta fram prognoser för alla dagar vid alla vägsegment med hela datamängden. En förbättring

skulle istället vara att dedikera olika LSTM-nätverk för olika delmängder som är mer lika varandra. Förhoppningen är att det blir enklare att ta fram prognoser för varje delmängd eftersom de följer samma tidsmönster genom hela tidsserien. Ett förslag på en delmängd är dagar under året som följer samma trafikmönster. Exempelvis skulle detta kunna handla om vardagar gentemot helgdagar (se figur 3.1) eller julledighet gentemot vanliga arbetsdagar. Ett annat förslag på en delmängd är att hitta de vägsegment som följer liknande trafikmönster. Detta handlar oftast om vägsegment som ligger fysiskt nära varandra. Att hitta vägsegment som är lika varandra kan exempelvis göras med korrelationsgrafer. I figur 5.2 visas en sådan. Denna beskriver hur trafikflödet under en dag för 14 av de totalt 811 vägsegmenten är relaterade. Exempelvis kan man se att segment 0-4 är relaterade och även segment 5-9. Att manuellt ta ut de vägsegment som är relaterade på detta sätt bland alla 811 olika segment kan bli en jobbig uppgift. Det finns dock maskininlärningsalgoritmer som kan underlätta denna process. I detta fall handlar det om att hitta olika kluster av data i den totala datamängden vilket är passande för *unsupervised learning*. En algoritm som gör detta kallas för *k-means clustering* [28].

Ytterligare strategier för att förbättra exaktheten för prognoserna skulle kunna vara att experimentera med andra typer av ANN eller helt andra maskininlärningsalgoritmer. Exempel på detta är att implementera simpla FFNN, som beskrevs i avsnitt 2.2.1, eller så kallade *Convolutional Neural Networks* (CNN) [29]. En möjlig metod är att kombinera både CNN och LSTM-nätverk i en och samma modell [30]. Något annat som är viktigt för framtida projekt är att implementera de simpla traditionella metoderna för prognosframtagningar (exempelvis det naiva tillvägagångssättet enligt avsnitt 2.1.1). Att räkna ut MAE för dessa metoder ger då en baslinje för vad som faktiskt är en vettig noggrannhet. I nuläget säger de uppmätta värdena på MAE (se avsnitt 4.1) inte så mycket eftersom de ej kan jämföras med något annat.



Figur 5.2: Korrelationsmatris som beskriver hur trafikflödet under en dag för 14 av de totalt 811 vägsegmenten är relaterade. Mer ljus färg indikerar högre korrelation. Diagonalen är gul eftersom varje segment är klart helt relaterat till sig självt.

Referenser

[1] Preparing for the future of artificial intelligence. https://web.archive.org/web/20161013141538/https://www.whitehouse.gov/sites/default/files/whitehouse_files/microsites/ostp/NSTC/preparing_for_the_future_of_ai.pdf. [Online; accessed 9-December-2018].

[2] Wikipedia contributors. Time series — Wikipedia, the free encyclopedia, 2018. [Online; accessed 25-December-2018].

[3] Wikipedia contributors. Forecasting — Wikipedia, the free encyclopedia, 2018. [Online; accessed 21-December-2018].

[4] Wikipedia contributors. Autoregressive integrated moving average — Wikipedia, the free encyclopedia, 2018. [Online; accessed 21-December-2018].

[5] Wikipedia contributors. Mean absolute error — Wikipedia, the free encyclopedia, 2018. [Online; accessed 31-December-2018].

[6] Seth Weidman. The 4 deep learning breakthroughs you should know about. [Online; accessed 10-January-2019].

[7] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[9] Ashley Hamer. Neural networks have advanced beyond our understanding, and that's kind of terrifying. [Online; accessed 10-January-2019].

[10] Michael Nielsen. Neural networks and deep learning. [Online; accessed 10-January-2019].

[11] Wikipedia contributors. Overfitting — Wikipedia, the free encyclopedia, 2018. [Online; accessed 5-January-2019].

[12] Christopher Olah. Understanding lstm networks, 2015. [Online; accessed 22-December-2018].

[13] Wikipedia contributors. Backpropagation through time — Wikipedia, the free encyclopedia, 2018. [Online; accessed 25-December-2018].

[14] Yonghui Wu and Mike Schuster et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[15] Wikipedia contributors. Vanishing gradient problem — Wikipedia, the free encyclopedia, 2018. [Online; accessed 25-December-2018].

[16] MongoDB Inc. What is mongodb?, 2018. [Online; accessed 10-January-2019].

[17] HERE maps API. https://www.here.com/en. [Online; accessed 28-November-2018].

[18] M Cuturi. UCI machine learning repository. PEMS-SF data set. https://archive.ics.uci.edu/ml/datasets/PEMS-SF, 2011. [Online; accessed 28-November-2018].

[19] California department of transportation. http://pems.dot.ca.gov/. [Online; accessed 28-November-2018].
[20] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. [Online; accessed 26-December-2018].
[21] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.
[22] F. Pedregosa and G. et al. Varoquaux. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
[23] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.
[24] François Chollet. keras. https://github.com/fchollet/keras, 2015.
[25] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
[26] Jason Brownlee. Time series forecasting with the long short-term memory network in python. 2017.
[27] Wikipedia contributors. Moving average — Wikipedia, the free encyclopedia, 2018. [Online; accessed 4-January-2019].
[28] Wikipedia contributors. K-means clustering — Wikipedia, the free encyclopedia, 2018. [Online; accessed 5-January-2019].
[29] Wikipedia contributors. Convolutional neural network — Wikipedia, the free encyclopedia, 2019. [Online; accessed 10-January-2019].
[30] Xingjian Shi et al. Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2014.