Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Petter Bjørseth
Kevin Dao
Ludvig Ellevold

Bruk av maskinlæring til å predikere fremtidige interessepunkter i et dataspill

Bacheloroppgave i Dataingeniør
Veileder: Ole Christian Eidheim
Mai 2019

NTNU
Kunnskap for en bedre verden

Petter Bjørseth
Kevin Dao
Ludvig Ellevold

Bruk av maskinlæring til å predikere fremtidige interessepunkter i et dataspill

Bacheloroppgave i Dataingeniør
Veileder: Ole Christian Eidheim
Mai 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

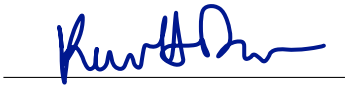
Forord

Denne bacheloroppgaven er utført av tre studenter på NTNU våren 2019 i samarbeid med spillselskapet Riddlebit Software. Riddlebit begynte med utviklingen av spillet Setback i 2017 og søkte i den anledning en gruppe studenter som ville bidra til å utvikle et avansert tilskuersystem til spillet.

Vi valgte denne oppgaven fordi vi så på den som en interessant oppgave med mange utfordringer. Vi ønsket å lære nye aspekter innenfor to felter vi ikke har arbeidet mye med før. Både maskinlæring og spillteknologi opplever stor vekst for tiden og derfor så vi på denne oppgaven som en god mulighet til å lære relevant og spennende teknologi som kan brukes videre i arbeidslivet.

Vi ønsker å rette en stor takk til vår veileder ved NTNU, Ole Christian Eidheim, som har kommet med innspill og oppfølging gjennom prosjektet. Vi ønsker også å takke Jonathan Jørgensen ved Riddlebit som har fulgt oss tett og gitt oss gode tilbakemeldinger og forslag til hvordan vi skal kunne løse oppgaven. Ikke minst vil vi takke de andre ansatte som jobber hos Riddlebit for å ha utviklet spillet og gitt oss denne muligheten. Til slutt vil vi takke familie og venner for støtte og tilbakemeldinger under prosjektet.

Petter Bjørseth | Kevin Dao | Ludvig Ellevold



Trondheim, 20.05.2019

Oppgavetekst

Oppgaven går ut på å utforske hvordan maskinlæring kan anvendes til å predikere fremtidige interessepunkter basert på data om tilstanden i et spill. Det er ønskelig med en generell og robust modell, med krav om at den kan kjøres i samarbeid med en spillserver. Fokuset ligger på hvilke modeller som presterer best, samt hvor langt fram i tid de kan predikere.

Hensikten med oppgaven er å lage produkt som kan brukes til å forbedre tilskueropplevelsen. Programmering og det praktiske arbeidet er en naturlig del av prosessen, og maskinlæringen er fra et anvendt perspektiv.

I den originale oppgaveteksten var det fokus på å lage en forbedret spectating-opplevelse. I dette prosjektet ble det kun fokusert på hvordan maskinlæring kunne anvendes til dette formålet, og ikke selve implementasjonen. Se vedlegg C for original oppgavetekst.

Sammendrag

E-sport er et fenomen hvor utøvere konkurerrerer mot hverandre i dataspill. E-sport har vokst voldsomt de siste årene, og følges av millioner verden over. I spill som Dota 2, League of Legends og Counter-Strike: Global Offensive kan utøvere leve av å spille profesjonelt, og i enkelte land sendes kamper direkte på TV.

Med økt interesse stilles også høyere krav til gode tilskuersystemer. Tilskuere ønsker å se de mest interessante hendelsene i en kamp mens de pågår. I høyhastighetsmiljøer som e-sport kan det bli utfordrende å operere slike systemer manuelt.

I denne rapporten ønsker vi å utforske muligheten for å automatisk predikere fremtidige interessepunkter i et dataspill. Vi ønsker å bruke maskinlæring til å finne ut hvor på en spillbane det vil være mest interessant å observere om et gitt antall sekunder. Dette er ønskelig slik at tilskuersystemer i dataspill kan automatiseres.

Forskning på maskinlæring har vist gode resultater med tidsserieprediksjoner, spesielt ved bruk av Recurrent Neural Networks, mer spesifikt Long Short Term Memory modeller. Det er også mye utvikling med bruk av maskinlæring innen spill, fra klassiske som sjakk til avanserte dataspill som Dota 2.

Flere ulike maskinlæringsteknikker er testet ut i dette prosjektet. Convolutional Neural Networks viste ingen evne til å håndtere problemet. RNN-teknikkene viser lovende resultater. LSTM er teknikken som viser mest lovende resultater, og ser ut til å kunne predikere 20 tidssteg fram i tid med en viss nøyaktighet, med muligheter for opp til 40 tidssteg.

Abstract

E-sport is a phenomenon where competitors compete against each other in video games. E-sport has experienced enormous growth the last few years, and is being watched by millions worldwide. In games like Dota 2, Leauge of Legends and Counter-Strike: Global Offensive players can make a living playing professionally. Some countries also broadcasts these competitions on live television.

With the growth of e-sports the demand for good spectating systems increases. Spectators wish to be shown the most interesting events as they unfold. In such high speed environments it becomes increasingly difficult for humans to operate the spectating systems.

In this paper we wish to explore the possibility of automatically predicting posisitions in the game where interesting events will happen in the future. To accomplish this we will be using machine learning to predict where these events will occur a given time in the future. These positions can then be incorporated with a spectating system for a fully automatic experience.

Research using machine learning for timeseries predictions has shown promising results, especially Recurrent Neural Networks, and more specifically Long Short-Term Memory-models. There is a lot of research on the usage of machine learning in games, from classic ones like chess, to modern video games like Dota 2.

In this paper we have compared different machine learning techniques. Convolutional Neural Networks did not show any promise of accomplishing good predictions. Recurrent Neural Networks however showed promising results, with LSTM performing the best. The best model seems to be able to predict relatively accurately 20 timesteps in the future, with possibilities of predicting 40 timesteps in the future.

Innhold

Figurer

Tabeller

1 Introduksjon og relevans

1.1 Bakgrunn

Maskinlæring har de siste årene vokst enormt. Fra å være en nisje som var lite utbredt og kun var i bruk til helt spesielle ting, brukes det nå i utallige fagfelt. I dag kan man se bruk av maskinlæring i selvkjørende biler, ulike typer taleassistanse og i sosiale medier. Bruken av maskinlæring innen dataspill er ennå i en tidlig fase, men har stort potensiale til å vokse i fremtiden. En av grunnene til dette er den enorme utviklingen av datautstyr som bidrar til bedre prosesseringskraft. Det finnes mange måter å anvende maskinlæring i dataspill på; man kan for eksempel programmere adferden til en non-player character (NPC) basert på mønstre fra spillerens handlinger.

Konkurransespilling, eller e-sport har de siste årene vist seg å være et svært vellykket konsept. E-sportindustrien er i følge PwC det hurtigst voksende markedet innen dataspilling og omsetter for flere milliarder kroner hvert år [1]. De mest populære spillene innen denne bransjen tiltrekker seg store mengder tilskuere, både gjennom nettstrømming og live arrangementer. Man kan dermed se et behov for å utvikle tilskuersystemer som integrerer med spillene på best mulig måte.

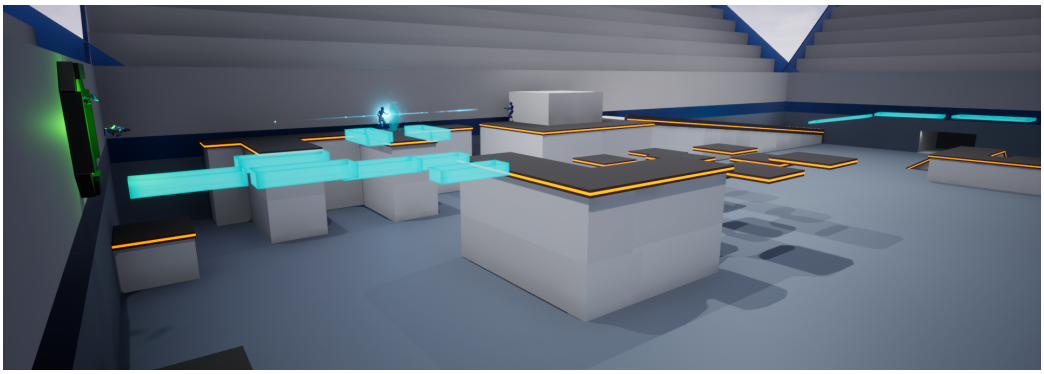
Oppgaven tar utgangspunkt i akkurat dette behovet til å utvikle et mer avansert tilskuersystem enn det som allerede finnes på markedet i dag. Målet med oppgaven er å anvende maskinlæring til å utvikle en modell som klarer å predikere interessepunkter under varigheten av en spillrunde i spillet Setback. Å kunne predikere interessante hendelser ut i fra historisk data kan samtidig være nyttig i andre bruksområder enn bare et tilskuersystem. Det kan for eksempel anvendes for å planlegge strategiske ruter og analysering av motspillere.

1.2 Setback

Setback er et "fast-paced first-person shooter (FPS)" spill. Riddlebit Software begynte utviklingen av spillet i 2017, og har vunnet flere priser innen spillutvikling siden da, deriblant Work-Work-prisen på Norwegian Game Awards 2018. De deltok på SpillExpo samme året med en prøveutgave av spillet.

En spillrunde i Setback begynner med at spillerne slippes ut fra en startsone. Målet er å komme først til mål ved å navigere seg gjennom en bane med hinder, samtidig som man skal bekjempe motspillerene med våpen. Spillerne

har i tillegg en mulighet til å gå i setback, som tilsier at spilleren blir flyttet tilbake til der de var fem sekunder før setback inntraff. Under setbacken kan spilleren verken bevege seg, skyte, eller bli beskutt, og dette varer i to sekunder. Setback er en av hovedfunksjonene i spillet og aktiveres når spilleren ikke har mer liv igjen, eller frivillig går i setback.



Figur 1.1: Eksempel på en bane i spillet Setback

1.3 Problemstilling

Målet for dette prosjektet er å utvikle en prediksjonsmodell som tar utgangspunkt i spillmiljøet i Setback. Vi skal ta i bruk ulike maskinlæringsmetoder og finne ut hvilken som er best egnet til dette formålet. Modellen skal kunne predikere hvor på banen det er mest interessant å følge med ved et gitt tidspunkt i spillet. Dette vil gi en eller flere kameradroner mulighet til å plassere seg slik at de får med seg handlingen. Etter gjennomføring av prosjektet ønsker vi å kunne besvare følgende problemstillinger:

- Hvilke maskinlæringsteknikker kan brukes for å predikere fremtidigere interessepunkter i dataspill?
- Hvor langt fram i tid klarer modellen å predikere interessepunkter som innenfor sin kontekst tilfredsstiller en feilmargin som forventes for et tilskuersystem i et dataspill?

1.4 Rapportstruktur

På grunn av mangel på norsk terminologi vil det i denne rapporten bli brukt en del engelske uttrykk der det er naturlig.

Strukturen på denne rapporten er delt inn i følgende kapitler:

Kapittel 1 Introduksjon gir en introduksjon til innholdet i denne rapporten. Her presenteres bakgrunnen bak oppgaven som er nødvendig for å forstå konteksten og teorien bak problemstillingene.

Kapittel 2 Teori tar for seg den grunnleggende teorien bak maskinlæringsmetodene som er relevante i dette bachelorprosjektet. Kapitlet vil også omhandle tidligere arbeid som er utført med nær tilknytning til de samme problemområdene i denne oppgaven.

Kapittel 3 Teknologi og metode forklarer de ulike stegene som ble utført for å komme fram til resultatet. Det vil inneholde beskrivelser av verktøy som ble brukt og prosessen som førte til disse resultatene.

Kapittel 4 Resultater presenterer resultatene ut i fra gitte metodevalg i det forrige kapittelet.

Kapittel 5 Diskusjon forklarer resultatene opp mot gitte problemstillinger. Her vil resultatene evalueres og diskuteres.

Kapittel 6 Konklusjon og videre arbeid konkluderer arbeidet og gir anbefalinger på hva som kan gjøres i videre arbeid med prosjektet.

1.5 Ordliste og akronymer

Se Vedlegg A for ordliste.

Se Vedlegg B for akronymer.

2 Teoretisk bakgrunn

2.1 Maskinlæring

Maskinlæring er et felt innen kunstig intelligens og har i dag mange ulike bruksområder. Feltet er stort, og stadig under utvikling. Med maskinlæring er det mulig å programmere datamaskiner til å lære å gjøre oppgaver på egen hånd, uten at man trenger å fortelle den eksplisitt hva den skal gjøre.

2.1.1 Flere typer læringsparadigmer

De forskjellige typene maskinlæringsparadigmer har ulike egenskaper. Problemet man ønsker å løse og dataen man har tilgjengelig legger grunnlaget for hvilket paradigme som bør benyttes. Man skiller som oftest mellom disse tre typene:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Supervised learning er en teknikk som krever at man har et sett med inputdata og tilhørende resultat. Målet med supervised learning er å finne en funksjon som, gitt en input, finner korrekt resultat. Dette gjøres ved å trene opp et nevralt nettverk med treningsdata, som består av par av input og resultat. Man ønsker altså å finne en funksjon f slik at $y = f(x)$, gitt at x er input og y er resultatet [2].

Unsupervised learning baserer seg på data som ikke har et tilhørende resultat. Man må derfor lære opp en modell uten å ha en fasit. Dette betyr at algoritmen selv må lære seg hvordan den skal klassifisere dataen. Disse teknikkene baserer seg i stor grad på å samle elementer i et datasett som er like hverandre i grupper, slik at ulike elementer ikke er i samme gruppe. Unsupervised learning brukes mye til cluster analysis for bedrifter som vil analysere data som ikke umiddelbart gir mening.[2]

Reinforcement learning er det tredje paradigmet. Denne typen går ut på å få en agent til å utføre best mulige handlinger i et miljø. En slik modell trenes ved å gi agenten en "belønning" for handlingene den velger, ut fra resultatet handlingene gir. Denne typen læring brukes for eksempel til å

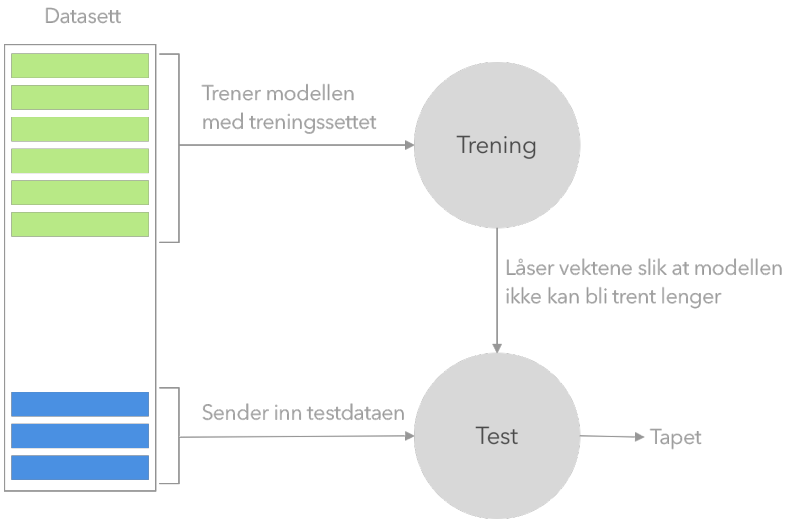
lære datamaskiner å spille sjakk, og har de siste årene vist ekstremt gode prestasjoner ved å slå sjakkmestre [3].

2.1.2 Data

For å trene et nevralt nettverk må man ha data. Hvordan datasettet ser ut påvirker modellen. Dersom man trener på et lite variert datasett så kan modellen bli for spesifikk. Den vil da kun gi fornuftige resultater for data som er tilnærmet lik data som er i dette datasettet. Dette kalles overfitting. Dette er sjeldent positivt, men kan i enkelte scenarioer være foretrukket.

Dersom dataen er variert vil det gi en mer generell modell. Denne vil kanskje ikke være like treffsikker, men kan håndtere flere forskjellige scenarioer.

For å teste hvor god en prediksjonsmodell er, brukes et treningssett og et testsett. Her er det vanlig at datasettet blir fordelt i 70% til treningssettet og 30% til testsettet. Modellen trenes på treningssettet, og når treningen er ferdig evalueres den på testsettet. Dette gjøres slik at man kan teste modellen på data den ikke er trent på.



Figur 2.1: Flytskjerma som illustrerer hvordan modellen trenes og testes

Et problem når det kommer til datasett er at man kan ha veldig mange variabler, også kjent som features. Her kan man bruke prosesser som feature engineering og data analyse for å identifisere hvilke features som påvirker modellen i stor grad, og hvilke som ikke påvirker modellen i det hele tatt.

2.2 Artificial Neural Network

2.2.1 Aktiveringsfunksjon

En aktiveringsfunksjon bestemmer resultatet ut fra en node i et nevralt nettverk. Disse brukes når data traverserer nettverket under forward propagation. Det finnes mange forskjellige aktiveringsfunksjoner, der den mest grunnleggende er identitetsfunksjonen:

$$f(x) = x \tag{2.1}$$

Lineære aktiveringsfunksjoner fungerer bra til enkelte regresjonsproblemer, men kan få problemer når det eksisterer ikke-lineære avhengigheter i dataen. For å løse mer avanserte problemer må vi da ta i bruk ikke-lineære aktiveringsfunksjoner. Det er mange slike, men vi ser kun på noen av de her:

Tanh er en hyperbolsk funksjon, som gir den hyperbolske tangenten til x. Resultatet av *tanh* vil alltid ligge mellom -1 og 1:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2}$$

Sigmoid funksjonen ligger nært 0 for små verdier, og nært 1 for store verdier. Den ligger alltid mellom 0 og 1:

$$sigmoid(x) = \frac{e^x}{e^x + 1} \tag{2.3}$$

Rectified linear unit (ReLU) er 0 hvis x er mindre enn 0, og x hvis x er større enn 0. Den har vist seg å være veldig nyttig i nevrale nettverk:

$$ReLU(x) = max(0, x) \tag{2.4}$$

Leaky ReLU er en variant av ReLU, som i stedet for å være 0 når x er mindre enn null, gjør den x veldig mye mindre:

$$LeakyReLU(x) = \begin{cases} x & hvis\ x > 0 \\ 0.01 \cdot x & hvis\ ikke \end{cases} \tag{2.5}$$

2.2.2 Forward propagation

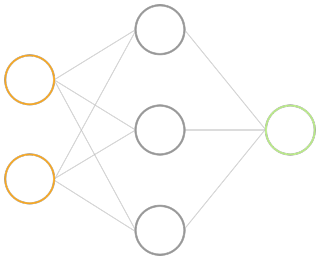
La oss se litt på hvordan et nevralt nettverk fungerer. Først deler man opp nettverket i flere lag. Et nettverk består av et input lag, et output lag og et gitt antall skjulte lag. Disse lagene består av mange nevroner (noder), og disse nevronene er knyttet sammen mellom hvert lag.

Forward propagation betyr å traversere nettverket og finne resultatet av en gitt input. Denne operasjonen brukes både under trening og under bruk av ferdig modell. Forward propagation består av to deler som repeterer seg selv for hvert lag [4]:

- Regne ut $y = \sum(vekt \cdot input) + bias$ inn til et lag
- Bruke aktiveringsfunksjonen: $a = f(y)$

Bias er en konstant som legges til hver node for å justere verdien i forhold til aktiveringsfunksjonen. Dette betyr at verdien til noden y kan bli forskyvet slik at dersom $vekt \cdot input$ ikke tilfredsstiller aktiveringsfunksjonen så vil bias justere y i ønsket retning [5].

Eksempelvis, for å forklare hvordan forward propagation fungerer så kan vi ta for oss et scenario der vi har følgende nettverk:

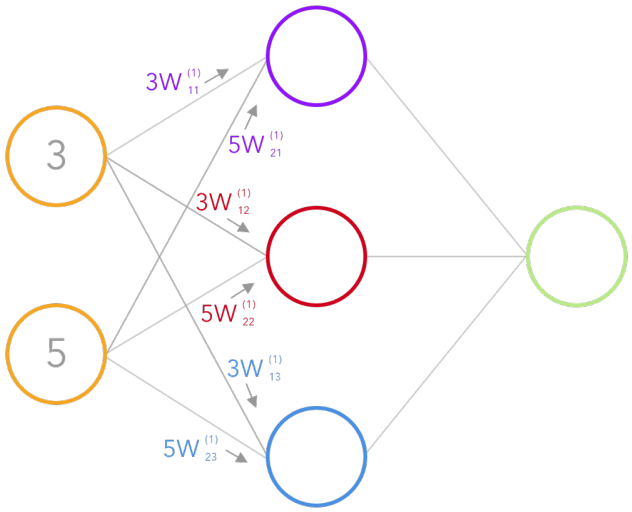


Figur 2.2: Nettverk med 2 noder i input laget (oransje), 3 noder i det skjulte laget (grå) og 1 node i output laget (grønn)

I dette nettverket har vi verdiene:

$$Input(I) \begin{bmatrix} 3 & 5 \\ 5 & 1 \\ 10 & 2 \end{bmatrix} \qquad Output(O) \begin{bmatrix} 75 \\ 82 \\ 93 \end{bmatrix}$$

Vi kan nå ta dette nettverket og traversere første par med inputs inn til det skjulte laget.



Figur 2.3: Figuren viser hvordan første par med input verdier fra input matrisen blir multiplisert vekten W inn til hver node (Bias er ikke tatt hensyn til i dette eksempelet)

.

Når alle parene har blitt regnet ut kan vi skrive det på matriseform for bedre oversikt:

$$Z = \begin{bmatrix} 3W_{11}^{(1)} + 5W_{21}^{(1)} & 3W_{12}^{(1)} + 5W_{22}^{(1)} & 3W_{13}^{(1)} + 5W_{23}^{(1)} \\ 5W_{11}^{(1)} + 1W_{21}^{(1)} & 5W_{12}^{(1)} + 1W_{22}^{(1)} & 5W_{13}^{(1)} + 1W_{23}^{(1)} \\ 10W_{11}^{(1)} + 2W_{21}^{(1)} & 10W_{12}^{(1)} + 2W_{22}^{(1)} & 10W_{13}^{(1)} + 2W_{23}^{(1)} \end{bmatrix} \tag{2.6}$$

Det som har skjedd så langt er at alle input verdiene fra input matrisen har blitt multiplisert med vekten mellom nodene og vi står igjen med en verdi. Nå som vi har denne verdien kan vi se på neste steg som forward propagation gjør, å bruke aktiveringsfunksjonen.

Disse funksjonene tar verdien $Z^{(nn)}$ og regner ut $a^{(nn)}$, hvor a er en gitt aktiveringsfunksjon 2.2.1. Det er disse verdiene som sendes videre til neste lag (tar plassen til input parene) [6].

$$\text{Original input (I)} \begin{bmatrix} I^{(11)} & I^{(12)} \\ I^{(21)} & I^{(22)} \\ I^{(31)} & I^{(32)} \end{bmatrix} \quad => \quad \text{Input til neste lag (a)} \begin{bmatrix} a^{(11)} & a^{(12)} \\ a^{(21)} & a^{(22)} \\ a^{(31)} & a^{(32)} \end{bmatrix}$$

Det er viktig å merke seg at aktiveringsfunksjonene lagres på en stakk, dette fordi at vi trenger disse funksjonene når vi skal utføre backpropagation.

Etter at nettverket har jobbet seg gjennom de skjulte lagene må output laget regnes ut. For å regne ut outputen fra nettverket tar vi resultatene fra aktiveringsfunksjonen i det skjulte laget a som er en 3x3 matrise. Denne ganges med vektene som er mellom det skjulte laget og output laget. Dette er en 3x1 matrise. Se figur 2.3. Vi får da funksjonen:

$$Z^{output} = a \cdot W^{(2)} \tag{2.7}$$

Da gjenstår bare en ting, og det er å sende Z^{output} gjennom aktiveringsfunksjonen:

$$\hat{y} = f(z^{output}) \tag{2.8}$$

Resultatet fra aktiveringsfunksjonen blir resultatet fra nettverket. For å oppsummere forward propagation så betyr det å sende inn en input, la input bli påvirket av nettverkets vekter og bias, bruke aktiveringsfunksjonene, for så å ende opp med et resultat. Det er slik at for å få gode resultater må vektene ha riktig verdi og ved første gjennomkjøring vil vi ikke få et godt resultat fordi vektene har en tilfeldig initiell verdi.

For å få en modell som gir gode resultater må nettverket trenes. Dette betyr at forward propagation brukes for å få et resultat, dette resultatet sammenlignes med antatt resultat gjennom en tapsfunksjon, og deretter traverseres det bakover i nettverket og vektene justeres gjennom backpropagation. Denne prosessen repeteres et gitt antall ganger (epoch).

2.2.3 Tapsfunksjon

Resultatet som nettverket gir må sammenlignes med ønsket resultat for å gi en indikator på hvor godt nettverket er[4]. For å regne ut tapet så finnes det mange forskjellige metoder. Den mest utbredte er mean squared error (MSE) [7]; hovedgrunnen til dette er at store differeranser vekter vesentlig mer enn små differanser. Dette er fordelaktig da man heller ønsker små feil mange ganger, enn veldig store feil noen ganger.

Tapsfunksjonen brukes til å justere vektene i nettverket. MSE kan formuleres slik:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2 \tag{2.9}$$

Der y_i er det faktiske resultatet ved tidssteg i, og p_i er modellens resultat ved tidssteg i.

2.2.4 Gradient Descent

Gradient descent er en optimaliseringsalgoritme. Denne algoritmen brukes til å oppdatere vektene når man trener en maskinlæringsmodell. Målet med algoritmen er å finne et globalt minimum til en funksjon. Innen maskinlæring vil denne funksjonen være tapsfunksjonen som brukes i nettverket. En gradient er enkelt sagt en flerdimensjonal derivert til en funksjon; det er en vektor der hvert element er en partiell derivert til funksjonen. I tre dimensjoner vil dette være:

$$\nabla f = (f_x, f_y, f_z) \tag{2.10}$$

der ∇f er gradienten til f, og f_x, f_y, og f_z er den partiell deriverte av f med hensyn på henholdsvis x, y, og z. I et gitt punkt P vil gradienten til f $\nabla f(P)$ ha retningen til den største økningen av f. Dette vil si at hvis man hadde beveget seg i retningen til gradienten fra punkt P så ville f ha økt.

Gradient descent bruker gradienten til en funksjon til å minimere den. Algoritmen virker slik:

Algoritmen initialiseres med en tilfeldig valgt x_0, som den prøver å konvergere fra. Man finner så gradienten i dette punktet, og tar et steg i motsatt retning av gradienten:

$$x_{n+1} = x_n - \gamma \nabla f(x_n) \tag{2.11}$$

der γ er steglengden. Steglengden sier hvor store skritt man skal ta i retningen man fant med gradienten. Denne må spesifiseres; dersom steglengden er for stor kan man hoppe over minimum, og dersom steglengden er for liten vil algoritmen bruke veldig lang tid på å konvergere. Man gjentar så dette steget til man har oppfylt et spesifikt krav. Det kan være et gitt antall iterasjoner, eller en lav nok funksjonsverdi.

Gradient descent brukes til å minimere tapsfunksjonen når man trener en maskinlæringsmodell. Man bruker altså gradient descent til å oppdatere vektene i nettverket.

Et problem med gradient descent er at den kan konvergere mot et lokalt minimum i stedet for et globalt minimum. Hvis et lokalt minimum ligger i en dyp grop er det ikke gitt at algoritmen klarer å finne veien ut av denne gropen slik at den kan konvergere mot det globale minimumet.

2.2.5 Backpropagation

Backpropagation bruker gradient descent til å oppdatere vektene i nettverket under trening. I forrige avsnitt (kapittel 2.2.4) så vi på hvordan gradient descent finner gradienten til tapsfunksjonen. Denne gradienten brukes så til å oppdatere vektene i nettverket. Man ønsker å oppdatere vektene slik at tapet blir minst mulig. For å finne ut hvordan man må oppdatere vektene, må man gå tilbake gjennom nettverket for å finne ut hvordan hver enkelt vekt påvirker tapet.

Som nevnt tidligere i kapittel 2.2.2 lagres aktiveringsfunksjonene på stakken under forward propagation. Dette gjøres slik at man kan bruke differensiering til å finne ut hvordan hver vekt i nettverket påvirker tapet. Til dette brukes kjerneregelen:

Gitt at

$$h(x) = f(g(x)) \tag{2.12}$$

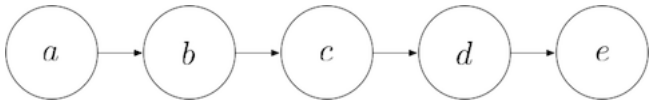
så sier kjerneregelen at

$$h'(x) = f'(g(x)) \cdot g'(x) \tag{2.13}$$

Skrevet på en annen måte: hvis $z = f(y)$ og $y = g(x)$ så er

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \tag{2.14}$$

Dette brukes til å finne den deriverte av tapet med hensyn på vektene. Si man har følgende graf av funksjonene som er lagret på stakken under forward propagation:



Figur 2.4: Graf av funksjonene på stakken [8]

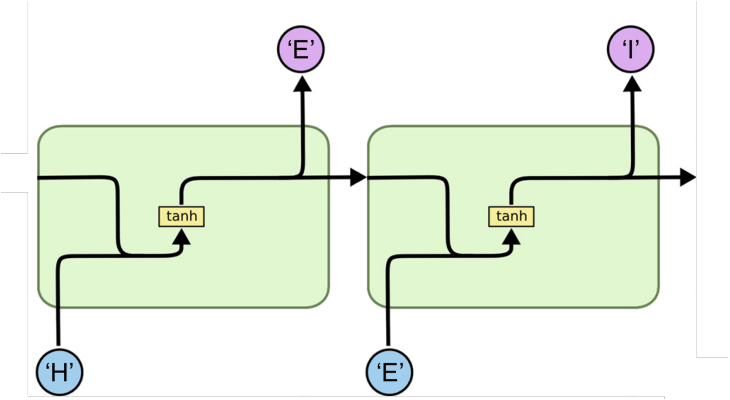
Hvis man her ønsker å finne den deriverte av e med hensyn på a, kan dette gjøres på følgende måte:

$$\frac{de}{da} = \frac{de}{dd} \cdot \frac{dd}{da} = \frac{de}{dd} \cdot \frac{dd}{dc} \cdot \frac{dc}{da} = \frac{de}{dd} \cdot \frac{dd}{dc} \cdot \frac{dc}{db} \cdot \frac{db}{da} \tag{2.15}$$

På denne måten kan vi starte med å differensiere tapsfunksjonen, for å deretter bevege oss bakover i nettverket ved å differensiere hvert ledd, og til slutt finne den deriverte av tapsfunksjonen med hensyn på hver enkelt vekt. Når dette er funnet kan man oppdatere vektene for å minimere tapsfunksjonen.

2.3 Recurrent Neural Network

Recurrent neural network (RNN) er en type nevralt nettverk hvor tidligere tilstander påvirker hvordan neste tilstand vil bli tolket. Hvis man for eksempel ser for seg et nettverk som skal lære seg å skrive 'Hei', vil RNN fungere bra. Her vil den lære at 'e' kommer etter 'h' og 'i' kommer etter 'e'. Forrige tilstand vil altså påvirke neste tilstand. Men den baserer seg ikke bare på forrige tilstand, men vil også bruke den historiske dataen som den ble trent på.



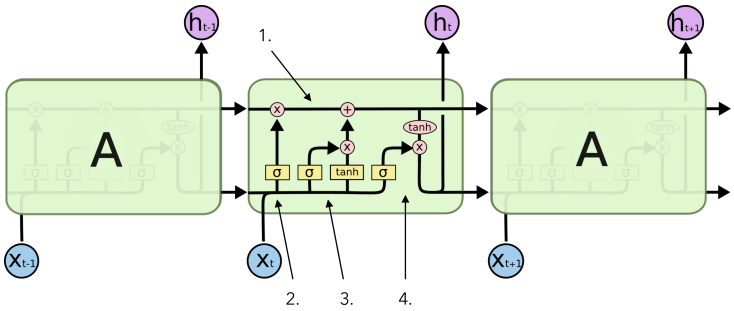
Figur 2.5: Illustrasjon over hvordan RNN fungerer. Tilpasset figur fra [9]

I figur 2.5 kan man se hvordan en tilstand påvirker den neste. Mellom de to tilstandene er det en pil. Denne pilen representerer en dataoverføring fra en tilstand til den andre. Når tilstanden til venstre er ferdig med en prediksjon kan vi ta med data fra den tilstanden videre til neste tilstand (høyre). Dataen beskriver forrige tilstanden og basert på dette så kan vi bruke denne opplysningen til å påvirke hva neste tilstand skal predikere.

Ta tilstanden til venstre. Den får inn 'H' og predikerer 'E'. Når neste tilstand skal predikere så får den inn 'E'. Den ser så at i forrige tilstand ble 'H' til 'E'. Basert på dette så skjønnner denne tilstanden at den skal predikere 'I'.

2.3.1 Long short-term memory

Long short-term memory (LSTM) er en type RNN. Med en langt mer komplisert celle kan LSTM *huske* data fra flere tilstander tilbake en vanlige RNN algoritmer.



Figur 2.6: Virkemåten til en LSTM celle. Tilpasset figur fra [9]

.

1. Cell state: Dette er data som kommer fra tidligere celler.
2. Forget gate: Her avgjøres om man skal "glemme" noe av dataen i cell staten.
3. Update gate: Her bestemmes hvilken data som skal lagres i cell staten til de neste cellene.
4. Output gate: Til slutt avgjøres en output basert på input og cell staten.

Cell state er en kanal hvor data kan passere. Den horisontale linjen i cellen viser hvordan dataen flyter og blir regulert av *forget* og *update gatene*. Det er denne *cell staten* som gjør at LSTM fungerer så bra. Dataen som flyter gjennom cellen gir nettverket mulighet til å huske informasjon som er relevant.

Forget er den første operasjonen som blir gjort. Her brukes en sigmoid funksjon til å bestemme om vi skal kaste vekk dataen som kommer fra forrige tilstand. Dette gjøres ved å se på output fra forrige tilstand h_{t-1} og input til denne tilstanden x_t. Resultatet vil bli en verdi mellom 0 og 1. 0 betyr at alt i *cell staten* skal "glemmes", og 1 betyr at alt skal skal taes vare på. Dersom verdien er mellom 0 og 1 så fjernes bare deler av *cell staten* som tilhører verdien som sigmoid funksjonen gir [9].

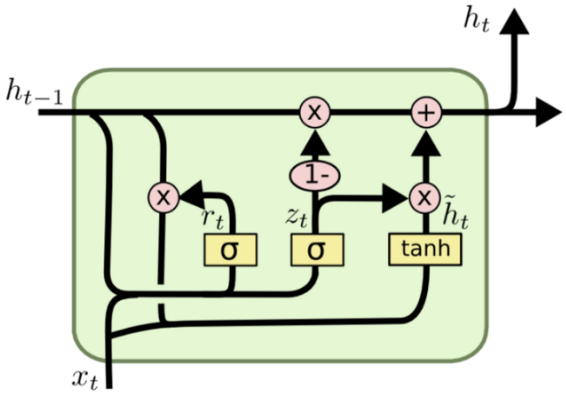
Etter at man har fjernet data som ikke er relevant vil vi lagre ny informasjon som kan være relevant for fremtidige tilstander. Her brukes *update gaten*. Denne gaten består av to deler. Sigmoid delen kalles "input layer gate" og avgjør hvilke verdier man vil oppdatere. Den andre delen er en tanh del. Her genereres vektorer av de nye verdiene. Disse verdiene blir kombinert for å få den nye verdien til *cell staten*. Når dette har blitt gjort oppdateres *cell staten* [9].

Nå som *cell staten* har blitt tatt hånd om gjenstår det å få ut et resultat (h_t).

Her brukes en sigmoid funksjon for å finne ut hvor i *cell staten* dataen som skal returneres ligger, tenk på det som adressen. *Cell staten* sendes gjennom en tanh funksjon for å få den på riktig form, deretter brukes adressen for å hente ut dataen som ligger på denne adressen i *cell staten*. Der er denne verdien som blir h_t [9].

2.3.2 Gated recurrent units

Gated recurrent units (GRU) er også en type RNN. I likhet med LSTM *husker* også GRU fra flere tilstander tilbake en vanlige RNN. Men måten den husker på er annerledes.



Figur 2.7: Virkemåten til en GRU celle [10]

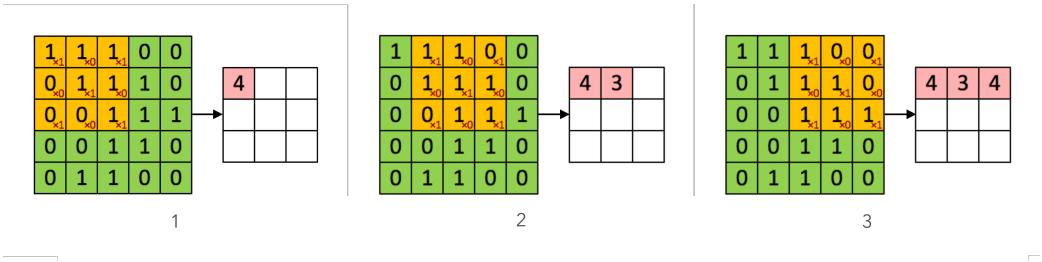
GRU har to porter, en *update* port og en *reset* port. Disse fungerer ganske likt som LSTM da begge påvirker den indre *cell staten*. En stor forskjell er at *cell staten* i en GRU er lik resultatet (h_t), mens i en LSTM så er cell staten og resultatet (h_t) separert [11].

2.4 Convolutional Neural Network

Convolutional neural network (CNN) er en type nevralt nettverk som tar i bruk convolutions og max-pooling for å lære mønstre i data. Denne typen nevrale nettverk er mye brukt på bilder, da den selv kan lære å trekke ut det viktigste fra store mengder data.

2.4.1 Convolutions

Convolutions er en teknikk for å redusere matrisens dimensjon, mens man samtidig bevarer integriteten i dataen.



Figur 2.8: Hvordan convolutions gjøres på en matrise. Tilpasset figur fra [12]

Figur 2.8 viser hvordan convolutions reduserer oppløsningen ved å regne ut en sum. Convolutions består av tre forskjellige komponenter. Matrisen som er farget grønt, et vindu som representerer dataen som er aktuell for en gitt tilstand, og noen konstante tall i rødt som bestemmer om dataen i en gitt posisjon i vinduet skal tas med i beregningen.

Her blir hvert tall i vinduet først ganget med konstanten, noen tall blir da ganget med 1 og noen med 0. Deretter summeres disse tallene opp og blir satt som resultat i den reduserte matrisen. Vinduet flytter seg så med n antall posisjoner, også kalt strides. I dette eksempelet bruker vi $n = 1$. Deretter repeteres prosessen helt til alle elementene i matrisen har blitt regnet ut.

Hvis vi tar eksempelet fra figur 2.8 blir resultatet en 3x3 matrise som gjenspeiler dataen i 5x5 matrisen, men med redusert oppløsning.

2.4.2 Max Pooling

Max Pooling er en teknikk hvor man drastisk reduserer oppløsningen til en matrise ved å kutte vekk elementer. La oss si at vi har en matrise som er 4x4, og vi sender denne gjennom et max pooling lag med størrelse 2x2 som har $stride = 2$.

Det som skjer da er at vi får et vindu med størrelsen 2x2, og dette vinduet hopper 2 elementer etter hver gjennomkjøring. Dette betyr at vi ikke får noen overlapp. Max pooling vil så se på de fire elementene i vinduet og avgjøre hvilket element som har høyest verdi. Resten vil bli ignorert. Siden en 4x4 matrise har 16 elementer og resultatet fra max poolingen blir en 2x2

matrise som har 4 elementer, så kan vi se at i dette eksempelet så har 75% av elementene i matrisen forsvunnet [13].

2.5 Tidligere arbeid

Denne seksjonen presenterer tidligere arbeid gjort innen fagfeltet og er delt inn i arbeid som er relatert til domenet, og arbeid relatert til maskinlæringsmetodene som er brukt i dette prosjektet. Det har vært utfordrende å finne fagstoff innen samme spesifikke problemområde som denne oppgaven omhandler; å predikere interessepunkter i et FPS spill. Vi har derfor sett på andre domener der det forskes på liknende problemstillinger.

2.5.1 Domenerelatert arbeid

Bruk av maskinlæring i skytespill er fortsatt på et tidlig stadium. Et problem som kommer i fokus er hvordan man utvikler ikke-spillerstyrte spillere(NPC) til å fremstille menneskelig atferd så godt som mulig. S. Hladky og V. Bulitko undersøkte dette i 2008 ved å evaluere "hidden semi-Markov" modeller og partikkelfiltre [14] for å predikere motstanderens posisjoner i et videospill. Målet var å kunne forbedre spillerstilen til NPCene. Resultatene viste at "hidden semi-Markov" modellen var mest nøyaktig av disse.

Dette ble også forsket på av B. Geisler i 2002 [15]. Her ble det benyttet ANN, ID3 beslutningstre og Naive Bayes for å finne den beste strategien for NPCer i FPS spill. Input feature-settet bestod av informasjon om omgivelsene rundt NPCen. Dette var antall fiender innen en viss radius, hvor mye liv som spilleren hadde igjen og hvor mye liv motspillere hadde. Ut fra disse attributtene ville den beregne hvilken av fire ulike handlinger den skulle gjøre. Dette var enten å bevege seg mot en retning, se mot en annen, hoppe eller akselerere. Her viste det seg at ANN hadde et lavere antall feilvurderinger på store datasett, mens ID3 og Naive Bayes gjorde det bedre på små datasett.

C. Ballinger, S Liu og S. Louis så i 2014 på hvordan man kunne anvende maskinlæringsmetoder i et RTS (Real Time Strategy) spill til å identifisere spillere basert på erfaringsnivå og deretter predikere spillerens handlinger ut i fra dette [16]. Å først kunne identifisere hvilket nivå spilleren er på, gjør det lettere å kunne predikere handlinger da spillere med ulik spillerfaring vil oppføre seg forskjellig. Dette ble gjort i spillet StarCraft II, som er et komplekst spill med utallige muligheter når det kommer til spillerhandlinger. De klarte ved bruk av et "J48 beslutningstre" å klassifisere 75% av 41 spillere

til riktig ferdighetsnivå. Samtidig oppnådde de en 81% nøyaktighet når de ville predikere den neste handlingen spilleren gjør ut fra 50 mulige handlinger. Dersom spilleren allerede var identifisert på ferdigheter så var dette tallet på 89%.

Et annet komplekst strategispill er Dota 2. OpenAI tar i bruk reinforcement learning og LSTM til å lære maskiner å spille dette spillet [17]. Under en spillrunde i Dota er det fem spillere på lag, og hver av disse spillerne bruker hver sin individuelle LSTM modell. Modellene blir trent ved å simulere spilletid tilsvarende 180 år hver eneste dag. Disse treningene består kun av simulerte spill mot seg selv. Her brukes en ekstremt skalert versjon av "Proximal Policy Optimization" for å lære opp hvert av de fem LSTM som består av et lag med 1024 enheter. OpenAI-laget, bestående av kun disse maskinene, har gjentatte ganger klart å vinne mot profesjonelle menneskelige motstandere.

2.5.2 Metoderelatert arbeid

M. Hagen skrev i sin masteroppgave om rutepredikering av maritim trafikk med bruk av nevrale nettverk [18]. I denne forskningen er det prøvd ut tre typer nevrale nettverk for å forsøke å finne den som best kan predikere posisjoner ett og ti steg fram i tid. Modeller som blir brukt er et standard nevralt nettverk, samt RNN og LSTM. Dataen som blir brukt til å trene modellene er lengde- og breddegrad, i tillegg til farten og kursretning til fartøyet i nåværende tidssteg. I forskningen viser han til at alle modellene gjør gode prediksjoner ett tidssteg fram i tid, men at de gir dårligere resultater lengre fram i tid. Både LSTM og standard RNN slet med å predikere posisjonen til fartøyene 10 steg fram i tid. I rapporten vises det til at dette kan skyldes at kun en liten feil i en prediksjon, kan føre til at feilen i senere prediksjoner øker mye.

L. Drevland og P. Finseth forsket på bruk av flere typer læringsmetoder, blant annet LSTM og GRU for å predikere etterspørselen av bysykler i Oslo til enhver tid [19]. Her hadde alle modellene gode prediksjoner, men det viste seg at LSTM gjorde det noe bedre enn de andre.

I 2016 forsket R. Fu, Z. Zhang og L. Li på hvordan man kunne forbedre daværende løsninger for predikeringen av trafikkflyt [20]. De brukte GRU og LSTM og sammenlignet disse modellene opp mot ARIMA (Auto regressive integrated moving average). Her viste det seg at GRU og LSTM presterte bedre enn ARIMA, samtidig som GRU slo LSTM med små marginer.

Convolutional neural networks ble brukt for å predikere ressursområder i

spillet StarCraft II. S. Lee (m.fl.) forsket i 2016 på hvordan man kunne bruke datasett fra andre baner i spillet for å predikere hvor det var optimalt å generere ressursområder på nye baner i spillet [21]. Metodene som ble brukt var ulike former for CNN som de navnga "simple", "shallow" og "deep" convolutional. Data var i koordinatform før det ble omgjort til bildefiler for modelleringen med CNN. Resultatene fra forskningen var varierte og ga ulike resultater for ulike baner. Mulige svakheter var risiko for overfitting av biaser i treningssettet eller forholdet mellom trenings- og valideringssettet. Samtidig bestod datasettet av 147 baner, som er et relativt lite datasett i proporsjon med antall parametere som ble brukt i nettverket.

3 Teknologi og metode

3.1 Metode

3.1.1 Overordnet metode

Figur 3.1 viser de ulike stegene hver modell går gjennom for å komme til resultatet. Rådataen prosesseres til et format som modellen benytter under trening. Deretter blir datasettene brukt for å predikere et resultat som til slutt skaleres tilbake til korrekt størrelse.

Figur 3.1: Overordnet arkitektur av prosessen

Vi har i dette prosjektet testet ut mange forskjellige metoder. Vi vil i dette kapittelet gå gjennom disse.

3.1.2 Rådata

Vi har i dette prosjektet hatt tilgang til to forskjellige datasett. Dog vi fikk tilgang til datasettet vist i tabell 3.2 veldig seint i prosjektet. Tabell 3.1 viser et tilfeldig utdrag av rådatasettene vi hadde tilgang til.

x	y	z	inSetback
-1004.712097	-1693.432983	428.149994	false
-1004.712097	-1693.432983	428.149994	false
-1017.379456	-1712.126709	428.149994	false
-1017.379456	-1712.126709	428.149994	true
-1017.379456	-1712.126709	428.149994	true

Tabell 3.1: Utdrag fra rådatasett med få attributter av en spiller

x	y	z	inSetback	inAir	armor
-1294.052124	-364.492584	187.149963	false	false	20
-1297.118408	-364.492584	187.149963	false	false	20
-1302.322388	-364.492584	187.149963	false	false	20
-1302.322388	-364.492584	187.149963	false	true	0
-1294.341797	-364.492584	187.149963	false	true	0

Tabell 3.2: Utdrag fra rådatasett med mange attributter av en spiller

Her har man følgende attributter:

- x, y og z er posisjonen til spilleren
- *inSetback* sier om spilleren er i setback eller ikke; en funksjon i spillet der spilleren kan bli satt tilbake i tid
- *inAir* sier om spilleren er i lufta, eller på bakken
- *armor* tilsvarer hvor mye skjold spilleren har, dette sier noe om hvor sannsynlig det er at spilleren havner i setback

  Som du ser er *inAir* og *armor* ikke med i det første datasettet.

Når vi får dataen ligger dataen til hver spiller i en egen fil. Dataen skrives om slik at hver spillrunde har sin egen fil med dataen til alle spillerne i den aktuelle spillrunden. Dette gjør det enklere for oss å hente inn dataen når vi trenger den. Som man kan se på tabellene 3.1 og 3.2 er det også noen boolske verdier. For å gjøre det enklere for modellen å forstå disse verdiene skrives disse om slik at $false$ blir 0, og $true$ blir 1.

3.1.3 Algoritme for interessepunkter

For å trene nettverket så må vi ha en *fasit*. For å lage en fasit for interessepunkter trenger man to ting: å definere hva et interessepunkt er, og en algoritme for å lage en fasit.

Vi valgte å definere et interessepunkt som posisjonen til spilleren nærmest mål gitt at det ikke forekommer en setback. Dersom det forekommer en setback så anses dette som mer interessant. Hvis to spillere er i setback samtidig er det spilleren nærmest mål som blir ansett mest interessant.

Algorithm 1: InterestPoint

```
most_interesting_points = []
/* Går gjennom hver tilstand                                    */
for i ← 1 to len(match) do
    highest_score ← 0
    highest_score_player_index ← −1
    /* Går gjennom hver spiller i gitt tilstand                 */
    for j ← 1 to len(match[i]) do
        if highest_score < calculate_score(match[i][j]) then
            highest_score ← calculate_score(match[i][j])
            highest_score_player_index ← j
        end
    end
    most_interesting_points ← match[i][highest_score_player_index]
end
return most_interesting_points
```

Algorithm 2: CalculateScore

```
in_setback ← false
spiller_setback ← state[3]
/* Sjekker om spiller er i setback                              */
if spiller_setback ≠ 0.0 then
    in_setback ← true
end
/* Finner avstanden til mål                                     */
distance ← 1 − euclidean_distance(player, goal)
/* Dersom spiller er i setback så økes scoren med 1             */
if in_setback then
    distance+ = 1.0
end
return distance
```

Algoritmene sammenligner hver spiller for et gitt tidssteg. Dette ved å regne ut en score for hver spiller. Denne scoren avhenger av hvor langt spilleren er fra mål, og om spilleren er i setback. Spilleren med høyest score blir ansett som mest interessant. Posisjonen til denne spilleren blir så satt som fasit for interessepunker i dette tidssteget. Ved å gjøre dette på en hel match vil vi få en fasit hvor hvert tidssteg inneholder den mest interessante posisjonen. Denne fasiten vil hoppe mye fram og tilbake mellom ulike posisjoner på banen.

Dersom man vil endre på hvordan fasiten skal se ut, og med det endre hvordan modellen predikerer, må man endre på algoritme 2.

3.2 LSTM

3.2.1 Preprossesering

Datasettene som brukes i prosjektet er samlet inn av Riddlebit Software. Datasettene består av data fra henholdvis 10 spillrunder med to spillere og 20 spillrunder med 4 spillere. I løpet av en spillrunde registreres posisjonen til hver enkelt spiller sammen med forskjellige typer attributter i gjeldende tidssteg. Disse datasettene lagres i comma separated values (CSV) format.

Vi har valgt å splitte datasettet vårt i en 70-30% fordeling mellom henholdsvis treningssett og testsett. Da vi har jobbet med begrensede mengder data, har vi valgt å ikke ta i bruk valideringssett.

3.2.2 Skalering

Før vi kan ta i bruk dataen til å trene modellen vår er det noen problemer som må løses. For det første er det mange negative verdier. Med de aktiveringsfunksjonene vi bruker vil dette være problematisk, da noen av de kun fungerer for positive verdier. Siden det kun er x og y koordinatene som har dette problemet, har vi valgt å finne de største negative verdiene som kartet i spillet tillater, og legge dette til x og y verdiene. På denne måten blir posisjonene skalert til å kun være positive.

I tillegg til dette ønsker vi å normalisere dataen. Rådataen har mange store og mange små verdier. Dette kan skape ubalanse i modellen, som gjør det vanskelig for den å finne verdifulle sammenhenger. Mange maskinlæringsteknikker håndterer slike verdier dårlig. Vi har derfor valgt å skalere all dataen til å ligge mellom 0 og 1. For attributtene *inSetback* og *inAir* er dette enkelt, da de allerede ligger i dette intervallet. For posisjonene har vi funnet størrelsen på kartet i x, y og z retning, slik at vi kan dele på disse:

$$\begin{aligned} Skalert\ x &= \frac{x}{største\ mulige\ x\text{-}posisjon} \\ Skalert\ y &= \frac{y}{største\ mulige\ y\text{-}posisjon} \\ Skalert\ z &= \frac{z}{største\ mulige\ z\text{-}posisjon} \end{aligned} \tag{3.1}$$

Maksimal verdi til *armor* er 100, så da deler vi på 100 får å få den mellom 0 og 1.

x	y	z	inSetback	inAir	armor
0.171993485	0.288188427	0.023393745	0	0	0.2
0.171610199	0.288188427	0.023393745	0	0	0.2
0.170959701	0.288188427	0.023393745	0	0	0.2
0.170959701	0.288188427	0.023393745	0	1	0
0.171957275	0.288188427	0.023393745	0	1	0

Tabell 3.3: Fra tabell 3.2 kan treningssettet se slik ut etter preprossesering.

3.2.3 Arkitektur

Modellene består av flere lag. De består av en "Stacked LSTM"; ett LSTM-lag som så sender resultatene sine inn i et annet LSTM-lag. Deretter et Leaky ReLU lag, så et lineært lag, og så et Leaky ReLU lag til, før vi får resultatet.

Dataflyten blir altså slik:

Figur 3.2: Dataflyten i modellene

3.2.4 Tapsfunksjon og optimalisering

For å trene modellene vår bruker vi mean squared error (MSE). Når man skal velge tapsfunksjon har det mye å si om man jobber med et klassifiseringsproblem eller et regresjonsproblem. Måten vi har valgt å bruke LSTM på i denne oppgaven, gjør at den tilnærmes et regresjonsproblem, så tapsfunksjoner beregnet på klassifiseringsproblemer (for eksempel Cross Entropy Loss) passer ikke. MSE er beregnet på regresjonsproblemer, da tapet beregnes på differansen mellom prediksjonen og fasiten.

Til optimalisering (gradient descent og backpropagation) har vi tatt i bruk en versjon av Adam Optimizer kalt AMSGrad. Det har tidligere blitt vist at Adam konvergerer hurtigere enn mange andre optimaliseringsalgoritmer. Rask konvergens har vært viktig for oss, da vi har begrensede ressurser til å trene modellene våre. Desto raskere tapet går mot null, jo mindre trenger vi

å trene modellene våre. Adam har et kjent problem der den i enkelte tilfeller sliter med justering av parametere, for å fikse dette problemet har det blitt videreutviklet en versjon som heter AMSGrad [22].

3.2.5 Metoder

Under utvikling av LSTM-modellene ble det testet ut tre ulike metoder.

Modell 1

Modell 1 er en standard LSTM som predikerer ett tidssteg fram i tid. Når den skal predikere lengre fram i tid, så predikerer den hvert tidssteg fra det den starter på, til det tidssteget den skal predikere. Den benytter seg av hidden layer når den skal predikere, og holder styr på den historiske dataen gjennom hele kampen.

Modell 2

Denne modellen bruker LSTM-cellen, men dataflyten blir mer som et vanlig feedforward network. Den tar inn ett tidssteg, og predikerer et gitt antall tidssteg fram i tid direkte. Det gjøres altså ingen prediksjoner mellom det man predikerer fra, og det som skal predikeres.

Modell 3

Modell 3 er en LSTM-modell som tar inn de 10 foregående punktene når den skal predikere. Det vil si at når den er på tidssteg x, og skal predikere tidssteg $y = x + n$, så bruker den alle tidssteg i intervallet $(x, x - 10)$ for å predikere tidssteg y. Dette gjør at modellen får tilpasset seg den historiske dataen i matchen før den gjør sin prediksjon.

3.3 GRU

For GRU modellen brukes samme preprossesering og skalering som LSTM. Se kapittel 3.2.1 og kapittel 3.2.2.

3.3.1 Arkitektur

Arkitekturen i GRU-modellen består av et GRU-lag, et Leaky ReLU lag, og til slutt et lineært lag. Under trening brukes Adam til optimalisering, og MSE som tapsfunksjon.

Figur 3.3: GRU-modellens arkitektur

3.4 Grid

Denne modellen bruker også LSTM-arkitekturen, men dataen legges inn i en grid før den sendes inn i modellen. Vi legger altså spillerne inn i en matrise som representerer banen i spillet. Dette er et forsøk på å overføre problemet til et klassifiseringsproblem. Håpet er at modellen kan klassifisere hvilken av rutene i griden interessepunktet ligger i.

Grunnen til at det var ønskelig å bruke en grid på denne måten, er at man kunne fått flere forslag til interessepunkter ved hvert tidssteg. Dette skjer ved at modellen for hvert tidssteg vil predikere sannsynligheten for at interessepunktet ligger i hver celle i griden.

3.4.1 Preprossesering

Til denne modellen preprosseseres dataen på en litt annen måte enn det vi gikk gjennom tidligere. Også her gjøres alle verdiene til positive verdier, men de skaleres ikke til intervallet [0, 1]. I stedet skaleres de med en gitt faktor:

$$\begin{aligned} Skalert\ x &= \left\lfloor \frac{x}{faktor} \right\rfloor \\ Skalert\ y &= \left\lfloor \frac{y}{faktor} \right\rfloor \end{aligned} \tag{3.2}$$

Grunnen for å skalere med denne faktoren er at banene er store, og man får dermed ekstremt store matriser om man bruker samme størrelse som banene.

Dette er ikke hensiktsmessig både for minnebruk og treningstid når man har begrensede ressurser, og når modellen skal brukes til predikering.

Før dataen sendes inn i modellen legges hvert tidssteg inn i en egen matrise. Her fjernes også z-dimensjonen, slik at dataen blir 2-dimensjonal. Dette er fordi matrisen blir veldig mye større med en gang man bruker 3 dimensjoner i stedet for 2. I tillegg er ikke z-dimensjonen av like stor relevans på den banen vi har jobbet med.

3.4.2 Arkitektur

Denne modellen bruker følgende arkitektur: et LSTM-lag, et tanh-lag, deretter et Leaky ReLU-lag, så et lineært lag, og til slutt et softmax-lag.

Figur 3.4: Grid-modellens arkitektur

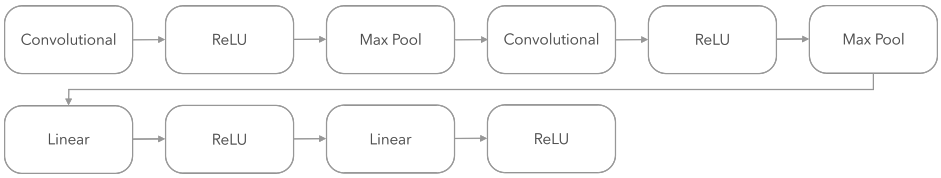
Adam ble brukt til optimalisering, og både Mean Square Error og Cross Entropy Loss ble testet ut som tapsfunksjon.

3.5 LSTM + CNN

Tanken med denne modellen var å bruke en LSTM-modell til å predikere framtidige posisjoner til spillerne på banen, for å deretter bruke en CNN-modell til å finne interessepunkter fra disse posisjonene. Denne modellen bruker samme preprossesering som Grid-modellen over, se kapittel 3.4.1.

3.5.1 Arkitektur

Ser her kun på arkitekturen til CNN-modellen, da LSTM-delen ville ha brukt en av arkitekturene som vi så på i kapittel 3.2.5 om LSTM.



Figur 3.5: CNN-modellens arkitektur

3.6 Teknologi

Utviklingen av maskinlæring de siste årene har ført til at det har kommet mange teknologiske verktøy og gjort det enklere å sette seg inn i feltet.

3.6.1 Maskinvare

Vi har i dette prosjektet fått tilgang til maskinkraft fra Instituttet for datateknologi og informatikk ved NTNU. I vårt tilfelle trengte vi mer prosesseringskraft til å håndtere mengden treningsdata og for at treningen av prediksjonsmodellen skulle gå i et hurtigere tempo. Vi fikk disponert en virtuell Linux maskin bestående av 32 CPU kjerner på Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz med 16GB RAM.

Fordelene ved bruk av den virtuelle maskinen var mange, blant annet at vi fikk kjørt lengre treninger over natten samt at maskinen var tilgjengelig når som helst.

Mot slutten av prosjektet fikk vi også tilgang til et GPU-cluster, dette fordi at vi trengte mer regnekraft en det CPU maskinen hadde tilgjengelig. Vi rakk ikke å ta i bruk dette i noen særlig grad.

3.6.2 Operativsystem og programmeringsspråk

Dette prosjektet ble gjennomført i operativsystemene Linux(Ubuntu 18.04 og Manjaro 18.0.2) og MacOS(Mojave 10.14.4). Python ble valgt som programmeringsspråk tidlig i prosjektløpet. Dette var det mest naturlige valget for oss da vi i tidligere gjennomføringer av maskinlæringsprosjekter også har tatt i bruk Python. Nærmere undersøkelser viser også at Python er det mest utbredte språket innenfor maskinlæringsprogrammering [23].

3.6.3 Pythonpakker

Python har som sagt, gjort det svært lite krevende å utvide programmer med nye funksjoner. Ved å installere og importere pakker til programmet kan man enkelt ta i bruk disse. Vi har i denne oppgaven tatt i bruk følgende pakker:

Matplotlib

For å visualisere resultatene fra prediksjonsmodellene våre har vi tatt i bruk Matplotlib. Pakken er et bibliotek som muliggjør plotting av 2D og 3D figurer og diagrammer av forskjellig type.

NumPy

NumPy gir støtte til bruk og håndtering av kompliserte matriser og innen maskinlæring er dette spesielt viktig. Vi har i dette prosjektet tatt i bruk NumPy for å håndtere de multidimensjonale datasettene som er brukt.

Pandas

Pandas er et verktøy som kan brukes for analysering og behandling av data. Pandas bruker enten "Series" eller "DataFrames" for å strukturere data. I dette prosjektet har vi tatt i bruk DataFrames for å kunne importere data i form av CSV til programmet. DataFrames strukturerer dataen til en todimensjonal tabell og er enkel å modifisere hvis dette er ønskelig.

Pylint

Pylint tilbyr en ekstra kvalitetssjekk på koden som skrives. Dette er svært nyttig når man skriver større programmer og ønsker å ha standardiserte former på koden slik at det forblir enkelt å lese for andre enn oss selv.

PyTorch

Problemstillingene til oppgaven vi har jobbet med var maskinlæringsorientert. Vi måtte derfor velge et bibliotek som passet best for oss. Det stod mellom enten PyTorch eller TensorFlow. Basert på erfaringen fra TensorFlow og

råd fra oppgavestiller så valgte vi å satse på PyTorch. Dette biblioteket er bedre integrert med Python og gir mer lettlest kode.

3.6.4 Verktøy og samskriving

Andre verktøy for koding og samarbeid omfatter følgende:

- IDE; PyCharm og Visual Studio Code
- Overleaf (LaTeX)
- Google Drive
- Git
- GitHub
- TravisCI
- PIP

3.7 Rolle- og arbeidsfordeling

Gjennom prosjektet har vi hatt en strukturert arbeidsfordeling. Hovedområdene bestod først og fremst av utvikling av maskinlæringsmodeller, visualisering, forskning og dokumentasjon. Alle i studentgruppen har jobbet under alle områdene, men det var naturlig at en med mer kunnskap i maskinlæring jobbet mer med dette. Da hver arbeidsdag ble utført sammen i gruppe ble kommunikasjonen og samarbeidet på tvers av områdene svært effektiv.

4 Resultater

I dette kapittelet vil vi gå gjennom resultatene vi har fått i løpet av prosjektet. Siden vi har eksperimentert veldig mye i løpet av prosjektet, vil vi ikke se på alle resultatene vi har fått. Vi vil vise resultater fra noen ulike modeller, som vil legge grunnlag for diskusjon i neste kapittel.

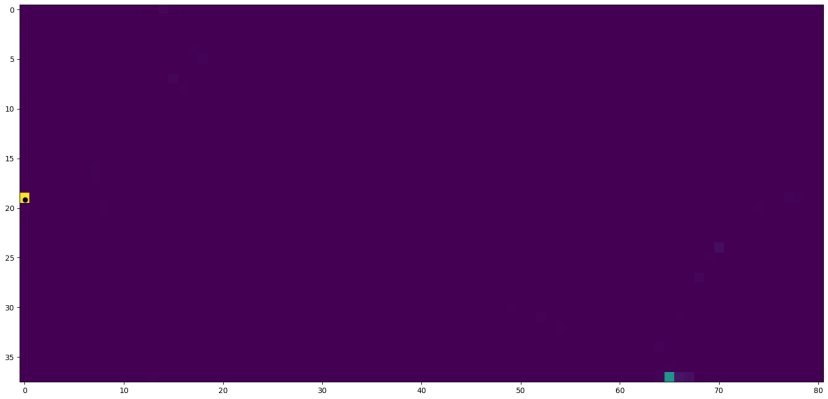
Siden interessepunktene kan hoppe mye fram og tilbake på banen, blir dataen vår ganske kaotisk. Dette gjør at det er relativt vanskelig å lage gode visualiseringer, spesielt når vi også må håndtere tidsdimensjonen. Vi har her valgt å visualisere i to dimensjoner, selv om posisjonene egentlig er i tre dimensjoner. Dette fordi det fort blir rotete å visualisere tre dimensjoner på et ark, og fordi z-aksen ikke er så relevant på denne spillbanen. I visualiseringene til GRU og LSTM vil heltrukken linje representere den faktiske dataen, og prediksjonene blir vist med stiplet linje.

4.1 Grid

Resultatet fra denne modellen er en to-dimensjonal matrise, hvor hvert element er sannsynligheten for at interessepunktet for dette tidssteget er på det punktet i banen som korresponderer til dette elementet i matrisen. Dette visualiseres i figurene under med heatmaps. De laveste verdiene er lilla, og fargene blir lysere desto høyere verdiene blir; de høyeste verdiene er gule (se figur 4.1). Det faktiske interessepunktet er en svart prikk på figuren. Da det er vanskelig å visualisere resultatet fra mange tidssteg på denne måten, viser vi her kun noen utdrag fra en match.

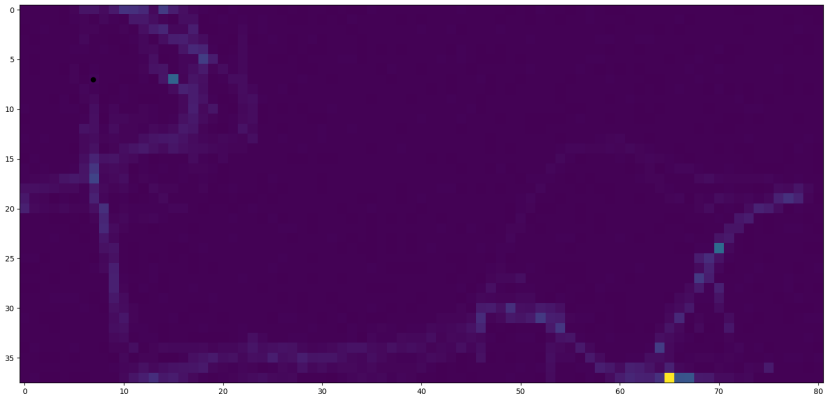
Denne modellen brukte faktor på 100 når den skalerte dataen (se kapittel 3.4.1).

Figur 4.1: Fargeskala for heatmap. Venstre er lave verdier, og høyre er høye verdier.



Figur 4.2: Prediksjon med grid for første tidssteg i en match (4.2)

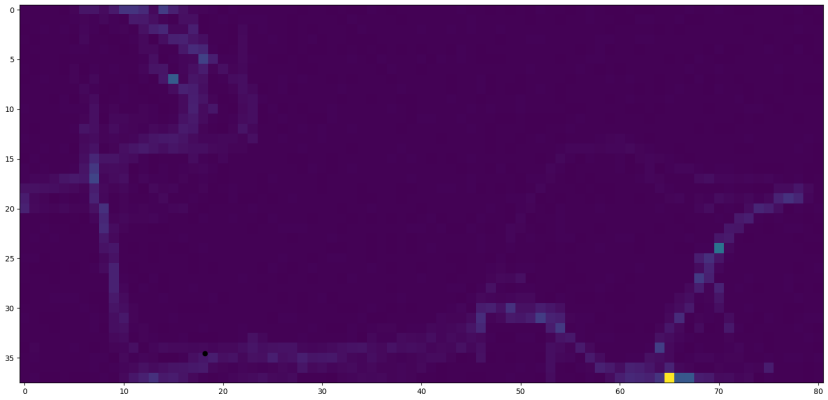
Fra figur 4.2 ser vi at modellen predikerer første tidssteg korrekt, da det er en gul flekk rett under den svarte prikken. Det meste av figuren er lilla, som viser at det er svært liten sjanse for interessepunkter i disse områdene. Vi ser dog noen forhøyde verdier nede i høyre hjørne, langt unna det faktiske interessepunktet.



Figur 4.3: Prediksjon med grid 1000 tidssteg ut i en match (4.3)

Det første man legger merke til i figur 4.3 er at hele banen spillerne bruker å

følge har forhøyde verdier. Da det meste av banen fortsatt er lilla, kan vi se et mønster i lyseblått, som er veien fra start til mål på denne banen. Vi ser også at det samme punktet ned til høyre på figuren som hadde noe høyere verdier ved tidssteg 0 nå har enda høyere verdier, og er modellens beste prediksjon. Dette er meget langt unna det faktiske interessepunktet, som i motsetning ligger i et helt lilla område.

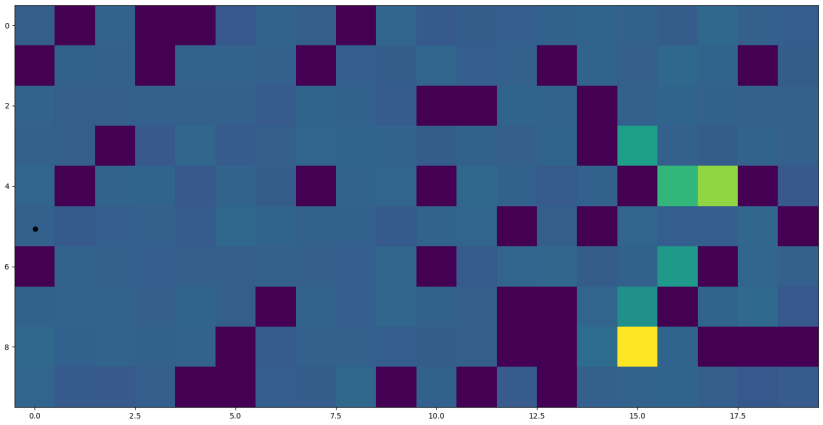


Figur 4.4: Prediksjon med grid 2000 tidssteg ut i en match (4.4)

Figur 4.4 viser at modellens prediksjoner ikke har endret seg fra tidssteg 1000 til 2000.

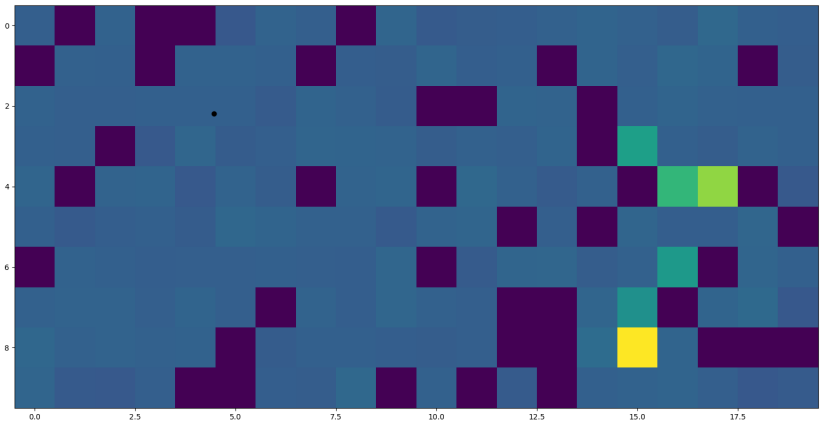
4.2 CNN

Disse resulatene er på samme format som Grid-modellen over (se kapittel 4.1), og visualiseres på samme måte. Denne modellen brukte en faktor på 4 når den skalerte dataen (se kapittel 3.4.1. På grunn av convolutions (se kapittel 2.4.1) og max-pooling (se kapittel 2.4.2) er oppløsningen i resultatene vesentlig lavere.



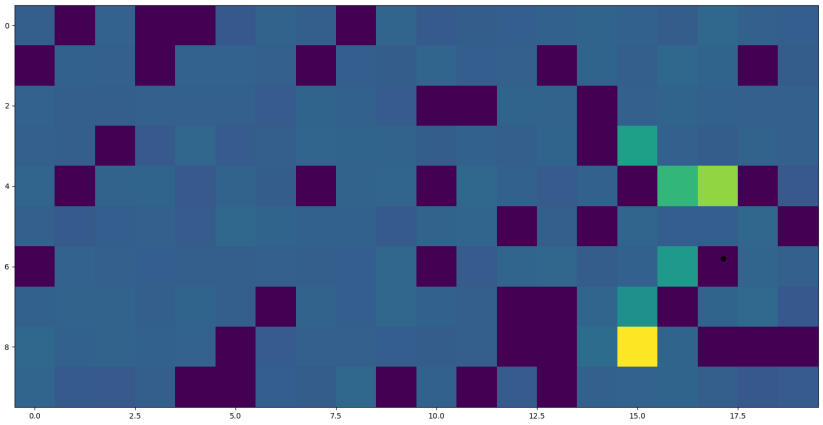
Figur 4.5: Prediksjon med CNN ved første tidssteg i en match (4.5)

I figur 4.5 ser vi at allerede ved første tidssteg er prediksjonene dårlige. Modellen predikerer at interessepunktene ligger nært mål til høyre på figuren, men det faktiske interessepunktet er ved start, helt til venstre på figuren.



Figur 4.6: Prediksjon med CNN 100 tidssteg ut i en match (4.6)

Det er enkelt å se fra figur 4.6 at prediksjonene til modellen er akkurat de samme ved tidssteg 100 som ved tidssteg 0, selv om det faktiske interessepunktet har beveget seg mye.

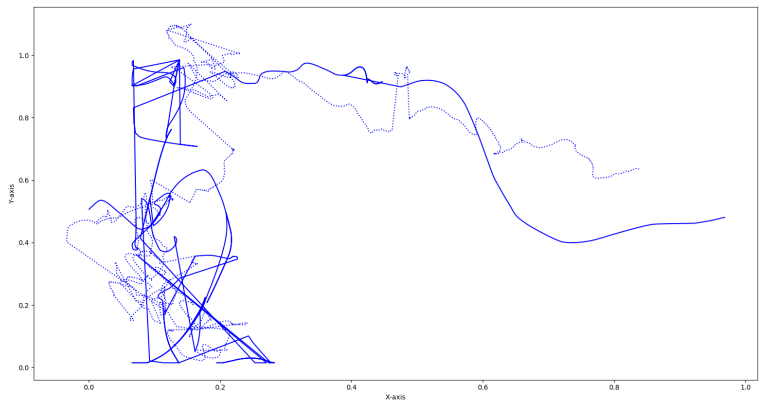


Figur 4.7: Prediksjon med CNN ved slutten av en match (4.7)

Figur 4.7 viser også de samme prediksjonene som de foregående figurene, selv om interessepunktet nå er ved mål.

4.3 GRU

Figur 4.8 viser resultatet fra en GRU-modell på samme format som LSTM-variantene.



Figur 4.8: 1 tidssteg fram i tid (4.8)

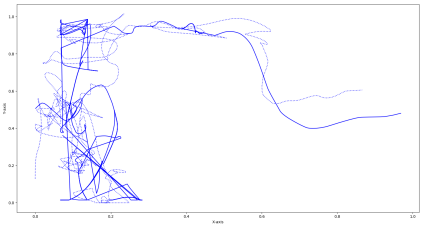
Prediksjonene til denne modellen klarer ikke å følge banen til de faktiske interessepunktene. Allerede tidlig i matchen er prediksjonene dårlige, og det er flere områder på banen der prediksjonene er veldig langt unna.

4.4 LSTM-varianter

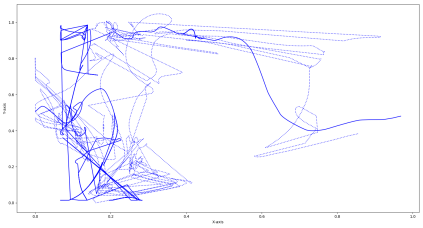
Vi vil her vise resultater fra tre forskjellige modeller. Alle disse er basert på LSTM-teknikker, men behandler dataen på litt forskjellige måter. Vi har gjort eksperimenter med å predikere 1, 20, 40 og 80 tidssteg fram i tid. Dette tilsvarer henholdsvis 0.05, 1, 2 og 4 sekunder fram i tid. Dette høres kanskje lite ut, men i et dataspill som dette er det både unødvendig og urealistisk å predikere lenger fram i tid enn dette.

Vi har valgt ut en match som vi sammenligner alle modellene på. Dette er gjort for å få et godt sammenligningsgrunnlag uten at det blir alt for mange figurer. Denne matchen er valgt da den har mye aktivitet i starten, samt en sving på slutten som gjør det enkelt å se forskjeller mellom modellene. Modellene har ikke trent på denne matchen, og dermed aldri sett denne dataen før. I figurene er start midt på y-aksen helt til venstre, og mål er midt på y-aksen helt til høyre. Den heltrukkede linjen er de faktiske interessepunktene, og den stiplede linjen er prediksjonene til modellen.

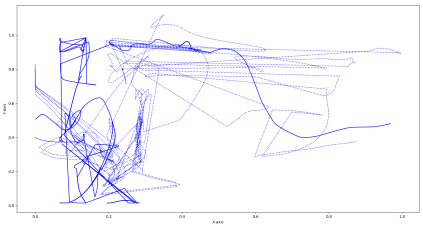
4.4.1 Modell 1



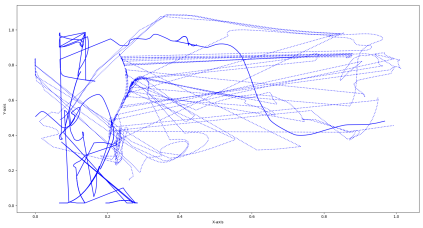
(a) 1 tidssteg fram i tid (4.9a)



(b) 20 tidssteg fram i tid (4.9b)



(c) 40 tidssteg fram i tid (4.9c)

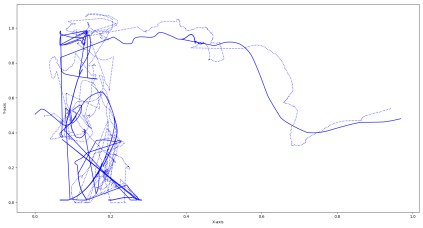


(d) 80 tidssteg fram i tid (4.9d)

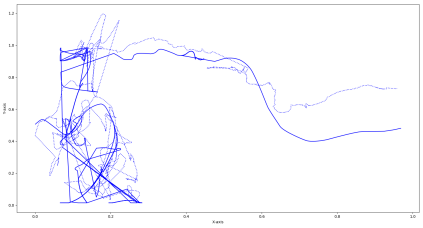
Figur 4.9: Visualisering av prediksjoner for modell 1

Vi ser allerede fra figur 4.9a at prediksjonene til denne modellen ikke er særlig gode. Allerede i starten av kampen er prediksjonene dårlige, og selv om den noenlunde klarer å følge banen til interessepunktene, har den mange punkter hvor den er alt for langt unna. Men da 4.9a noenlunde holdte seg i nærheten av interessepunktene, er 4.9b vesentlig verre. Denne modellene avviker vesentlig fra banen til interessepunktene, og er tidvis ikke i nærheten. I 4.9c og 4.9d blir tendensen fra 4.9b bare verre. Prediksjonene havner lenger og lenger unna de faktiske interessepunktene, og vi får store mengder støy.

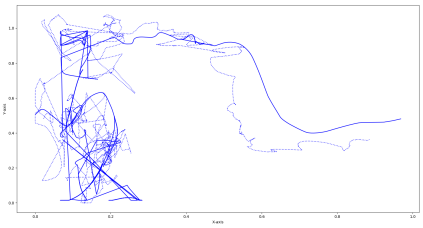
4.4.2 Modell 2



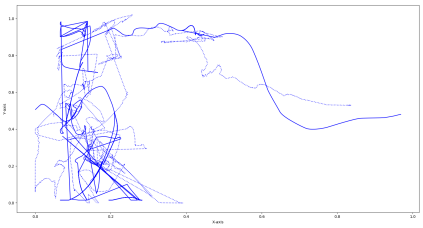
(a) 1 tidssteg fram i tid (4.10a)



(b) 20 tidssteg fram i tid (4.10b)



(c) 40 tidssteg fram i tid (4.10c)

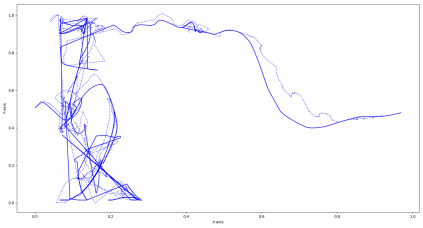


(d) 80 tidssteg fram i tid (4.10d)

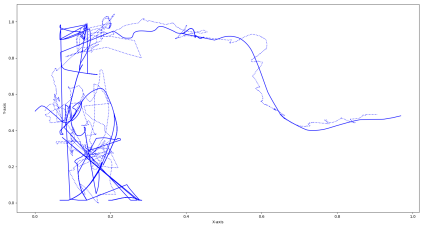
Figur 4.10: Visualisering av prediksjoner for modell 2

Dersom vi ser på figur 4.10a så ser vi at den sliter i områder med mye aktivitet. Typisk så forekommer det en del aktivitet på starten av spillet, mellom $x = 0.0 \rightarrow 0.2$. Denne trenden forekommer også ved flere tidssteg fram i tid. Og i 4.10d så bommer den veldig. Samtidig så holder den seg ganske greit mot slutten av banen. Mellom $x = 0.5 \rightarrow 1.0$ så treffer den bedre, spesielt 4.10a og 4.10b. Når tidsstegene blir større en 20 så reduseres også treffsikkerheten på denne delen av banen.

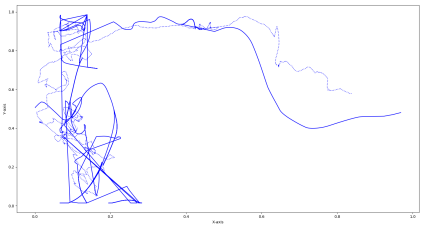
4.4.3 Modell 3



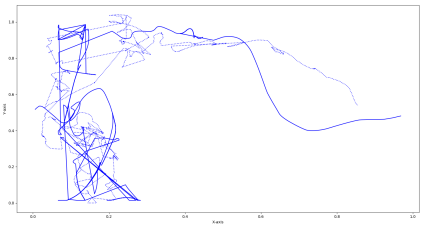
(a) 1 tidssteg fram i tid (4.11a)



(b) 20 tidssteg fram i tid (4.11b)



(c) 40 tidssteg fram i tid (4.11c)



(d) 80 tidssteg fram i tid (4.11d)

Figur 4.11: Visualisering av prediksjoner for modell 3

Vi ser fra figur 4.11a at prediksjonene følger banen til interessepunktene godt. Tidlig i kampen ligger prediksjonene veldig tett med de faktiske punktene. I svingen mot slutten av kampen så bommer den litt, og klarer ikke helt å ta en like dyp sving som fasiten. 4.11b ser vi har litt mer støy gjennomgående gjennom kampen. Den har flere småfeil her og der, og klarer ikke like godt å følge banen tidlig i kampen. Den har også noen plasser hvor den bommer mer, før den henter seg inn til den gitte banen. Det ser derimot ut til at den håndterer den siste svingen noe bedre enn 4.11a.

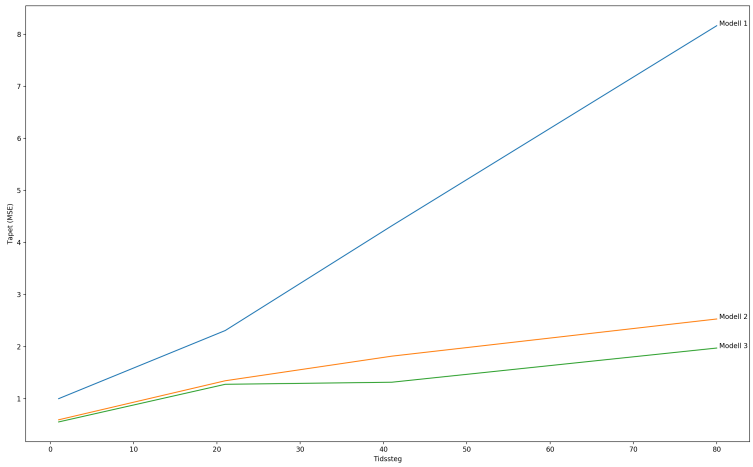
I 4.11c og 4.11d ser vi at resultatene tydelig blir verre. Begge disse har mer støy i starten av kampen, og klarer ikke å følge banen. De ligger stort sett noenlunde i nærheten, men følger ikke interessepunktene ut i svinger. Vi ser også at ingen av disse klarer å følge den siste svingen. Der de to andre modellene lå relativt nært i svingen, bommer disse ganske mye.

4.4.4 Evalueringer

Tabell 4.1 viser evalueringer av de forskjellige modellene ved forskjellige tidssteg. Disse evalueringene er gjennomsnittet av tapet regnet ut med MSE på tre testkamper. Figur 4.12 viser tabellen i en graf.

	Tidssteg			
Modell	1	20	40	80
1	0.99989	2.30860	4.32101	8.16349
2	0.59396	1.34382	1.81638	2.53034
3	0.55293	1.27581	1.31567	1.97221

Tabell 4.1: Evalueringer



Figur 4.12: Graf som viser tapets utvikling over tid

Det første som kan poengteres er at tapet til modell 1 øker nesten proporsjonalt med antall tidssteg fram i tid som predikeres. Økningen er ikke like stor fra 1 til 20 tidssteg, men fra 20 til 80 øker tapet voldsomt med hvert tidssteg. Modell 2 har en relativt jevn økning mellom alle tidssteg. Modell 3 øker noe mellom 1 og 20 tidssteg, øker så nesten ikke fra 20 til 40 tidssteg, før den øker noe igjen mellom 40 og 80 tidssteg.

5 Diskusjon

5.1 Modeller

5.1.1 Grid

I resultatkapittelet så vi at denne modellen så ut til å gi samme resultat tilnærmet uavhengig av input. Det kan være flere grunner til dette, en av de er at vi slet med å få trent denne modellen. Med store baner blir inputen veldig stor, noe som gjør at det tar lang tid for modellen å prosessere hvert element. I tillegg hadde vi begrenset maskinkraft tilgjengelig, så det å få trent en slik modell tilstrekkelig viste seg å være svært vanskelig. Et annet poeng er at hensikten med modellen er å kjøre på en server og gjøre prediksjoner i sanntid, og hvis en modell bruker for lang tid på å utføre hver prediksjon ville dette være lite hensiktsmessig.

For å få trent denne modellen skalerte vi ned banen med en faktor på 100. Dette senker oppløsningen på banen drastisk, og vil gjøre de predikerte interessepunktene mindre nøyaktige. Dette anses derimot ikke som et stort problem, da kameradronene som skal fange opp interessepunktene i spillet vil holde noe avstand, og vil nok fange opp interessepunktet i synsfeltet sitt. Selv om det er ønskelig med høyest mulig nøyaktighet, vil man alltid måtte regne med en feilmargin. Dersom modellen hadde levert gode resultater, hadde vi ikke sett på feilmarginen som et problem.

5.1.2 CNN

Denne modellen hadde mange av de samme problemene som nevnt i kapittel 5.1.1. Begge disse modellene bruker samme type grid, og får samme type problemer med trening og trege prediksjoner.

Som vi så i resultatkapittelet hadde også denne modellen de samme prediksjonene gjennom hele matchen. En annen mulig grunn til dette er at inputen til modellen vil ha ekstremt mange nullverdier i forhold til andre verdier. Når vi trente denne modellen skalerte vi ned banen med en faktor på 4. Dette vil gi en matrise med 1906065 elementer. Da vi trente på et datasett med to spillere, vil to av disse elementene være forskjellig fra 0, og alle andre vil være 0. Dette gjør det nok meget vanskelig for modellen å lære seg mønstre i dataen.

En teknikk vi prøvde for å løse dette problemet var å bytte ut nullverdiene med tilfeldige lave tall. Vi byttet ut nullverdiene med tall mellom 0 og 0.2, for å forsøke å tvinge modellen til å endre prediksjonene sine selv om inputen var tilnærmet lik hver gang. Dette gjorde at modellen klarte å komme med noen prediksjoner, men de var fortsatt langt fra gode nok.

5.1.3 GRU

Vi testet GRU mot LSTM relativt tidlig i prosjektet. Det vi fant da var at med tilsvarendre parametre og treningstid gjorde GRU det dårligere enn LSTM på vårt problem. Vi tok da en avgjørelse på å fokusere på LSTM, og har derfor ikke testet GRU i etterkant av dette. Med begrenset tid og ressurser var det den mest logiske avgjørelsen. Med mer tid og ressurser kunne det vært interessant å se nærmere på hvordan GRU hadde målt seg mot LSTM med den videreutviklingen vi har gjort siden den tid, men vi anser det som høyt usannsynlig GRU hadde utkonkurrert LSTM.

5.1.4 LSTM-varianter

Vi har her testet tre forskjellige modeller som bruker LSTM-arkitekturen. Modell 1 predikerer 1 tidssteg fram i tid, og holder styr på historisk data gjennom hele matchen. Modell 2 predikerer et antall tidssteg fram i tid, men bruker historisk data i mindre grad. Modell 3 predikerer også et antall tidssteg fram i tid, og bruker historisk data fra de 10 siste tidsstegene for å gjøre det.

Modell 1 er desidert den dårligste av disse modellene. Det at den er dårligere til å predikere lengre fram i tid kan nok forklares av at den predikerer alle tidssteg mellom start og slutt. Det gjør at selv en liten feil i en prediksjon kan føre til store feil i senere prediksjoner. På denne måten blir feilen større og større helt til den når det tidssteget den skulle predikere, og da er feilen meget stor.

Modell 2 og 3 er vesentlig bedre enn modell 1, og har nokså like evalueringer. Når man ser på figur 4.10 og 4.11 ser dog modell 3 bedre ut. Figur 4.12 viser også til at modell 3 har et noe lavere tap enn modell 2. Spesielt fra 40 tidssteg og oppover klarer ikke modell 2 å holde følge med modell 3.

Modell 2 er veldig enkel å trene, og bruker lite ressurser. Den trener fort, og vi har fått trent denne modellen til tapet ikke lenger minsker. Modell 3 bruker noe mer ressurser når den trener, og bruker mye lengre tid. På grunn

av problemer med maskinene vi har brukt til trening, har vi ikke fått trent denne modellen så lenge som er ønskelig. Dette betyr at med tilstrekkelige ressurser kan man forvente enda bedre resultater fra denne modellen.

Generelt for alle modellene er at de presterer bedre når de predikerer få tidssteg fram i tid, kontra mange tidssteg fram i tid. Dette er som forventet, da det er enklere å predikere det som skal skje like etterpå enn det som skal skje om en stund. Vi ser dog at modell 3 håndterer det å predikere lengre fram i tid bedre enn de andre.

5.2 Problemstillinger

Problemstillingene vi har jobbet med i dette prosjektet er følgende:

- Hvilke maskinlæringsteknikker kan brukes for å predikere fremtidigere interessepunkter i dataspill?
- Hvor langt fram i tid klarer modellen å predikere interessepunkter som innenfor sin kontekst tilfredsstiller en feilmargin som forventes for et tilskuersystem i et dataspill?

Det finnes uttalige maskinlæringsteknikker. Noen er veldig generelle, andre er tilpasset spesifikke problemdomener. Når man jobber med tidsserier er RNN desidert den mest brukte teknikken. Det ble derfor naturlig for oss å utforske disse teknikkene, og ikke se på så veldig mange andre teknikker. Vi valgte også å teste ut CNN da en bane i dataspill kan tolkes som en grid, noe som CNN er god på å håndtere.

I seksjonen over drøftes noen av modellene som er testet ut i dette prosjektet. Disse bruker ulike teknikker; CNN, LSTM og GRU. Ut fra resultatene vi har gått gjennom ser det ut til at LSTM håndterer dataen vi jobber med best, og gjør de beste prediksjonene. Den dataen vi har jobbet med ser ikke ut til å passe til CNN, da den ikke klarer å lære mønstrene godt nok. GRU kan sammenlignes med LSTM, men ser ikke ut til å være like god.

I denne oppgaven er det en avveining hvor langt fram i tid man ønsker å predikere kontra hvor stor feilmargin man kan akseptere. Man ønsker å predikere langt nok fram i tid til at man kan flytte kameraene til den rette plassen på banen slik at man får med seg det interessante som skal forekomme der, samtidig som at man vil at nøyaktigheten på prediksjonene skal være god nok slik at man flytter kameraene til rett plass.

Til syvende og sist vil det være hvor gode man opplever prediksjonene når

man bruker de i praksis som avgjør dette. Da vi ikke har fått testet modellene våre i spillet, kan vi ikke trekke noen konklusjoner om dette. I figur 4.12 kan vi se utviklingen av tapet med antall tidssteg fram i tid man predikerer. Det vil være uinteressant å predikere 1 tidssteg fram i tid i praksis, siden dette tilsvarer 0.05 sekunder. Med eksperimentene vi har gjort vil det dermed være 20, 40 eller 80 tidssteg som er aktuelt.

For modell 3 ser vi at tapet ikke øker noe vesentlig mellom 20 og 40 tidssteg. Men likevel kan vi i figur 4.11c se at den håndterer den siste svingen i banen vesentlig dårligere. Det er dog mulig at den med noe videreutvikling og mer trening kan bli bedre på dette området.

5.3 Implementasjon

Med maskinvaren som er brukt i dette prosjektet tar det ca. 0.001 sekund å utføre en prediksjon for modell 3. Dette er viktig da det ønskes å utføre, og ta i bruk prediksjonene i sanntid. Prediksjonene er like kjappe på CPU som på GPU, som er gunstig hvis serveren ikke har GPU-kraft.

Modellene må trenes på forhånd. For å få en tilstrekkelig trent modell burde det beregnes mellom 10 og 20 timer. Fordelen er at etter man har trent modellen, trenger man ikke å trene den igjen så lenge spillet fungerer på samme måte. Om måten spillerne spiller på eller måten spillet utformer seg på endrer seg mye, kan det være nødvendig å trene på nytt.

Modellene må trenes separat på forskjellige baner, da den lærer seg mønstre som er spesifikke for banen den trener på. Den håndterer også kun et gitt antall spillere, så hvis antall spillere varierer vil det være nødvendig å trene flere modeller til de forskjellige antallene.

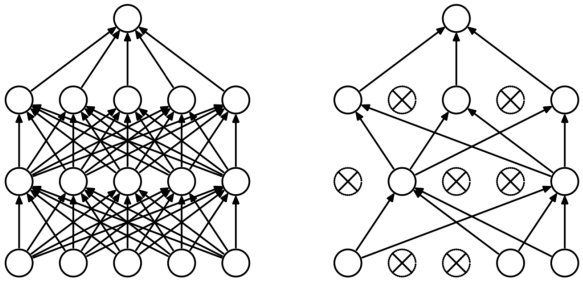
Modellene er generelle og håndterer forskjellige antall attributter og spillere når den trener, men etter den gitte modellen er trent må både antall attributter og antall spillere være det samme som den trente på. Modellene er ikke utviklet spesifikt for Setback, noe som betyr at de skal kunne brukes innenfor andre relaterte domener, hovedsakelig andre spill. Vi kan ikke si noe om hvor godt de vil prestere i andre situasjoner. Det eneste som må endres er algoritmene for å finne interessepunkter.

Siden systemet er skrevet i Python, skal det være enkelt å integrere dette med en server for å bruke det i matcher som spilles.

5.4 Alternative metoder

I denne seksjonen vil vi gå gjennom teknikker som er verdt å vurdere i et slikt prosjekt, som vi har valgt å ikke ta i bruk.

5.4.1 Dropout



Figur 5.1: Illustrasjon over et vanlig nevralt nettverk (venstre) og et nettverk med ignorerte noder (høyre) [24]

Dropout er en teknikk hvor enkelte tilfeldige noder i den nevrale nettverket blir midlertidig ignorert. Nodene, og deres tilhørende koblinger blir kun ignorert for en iterasjon før et sett med nye noder blir valgt. Dette gjøres for å unngå overfitting

Et problem som oppstår ved å bruke dropout er at det fører til en økning i treningstiden. Tiden det tar å trene nettverket økes med 2-3 ganger, dette fordi at ved hver iterasjon vil nettverket trene et "nytt" redusert nettverk [24].

For oss var ikke overfitting et problem da oppgaven tillot en modell som var spesifisert til en spesifikk bane. Dette førte til at dropout ble nedprioritert, og vi til slutt ikke hadde tid til å teste ut denne teknikken. Uansett så ville økningen av treningstiden ikke vært gunstig for oss grunnet begrensingene vi hadde i både tid og ressurser.

5.4.2 Transfer learning

Transfer learning er en teknikk hvor man først trener opp en modell, som så trenes videre på et liknende problem. Dette fungerer ved at den første modellen man trener, heretter kalt "den generelle modellen", blir trent opp.

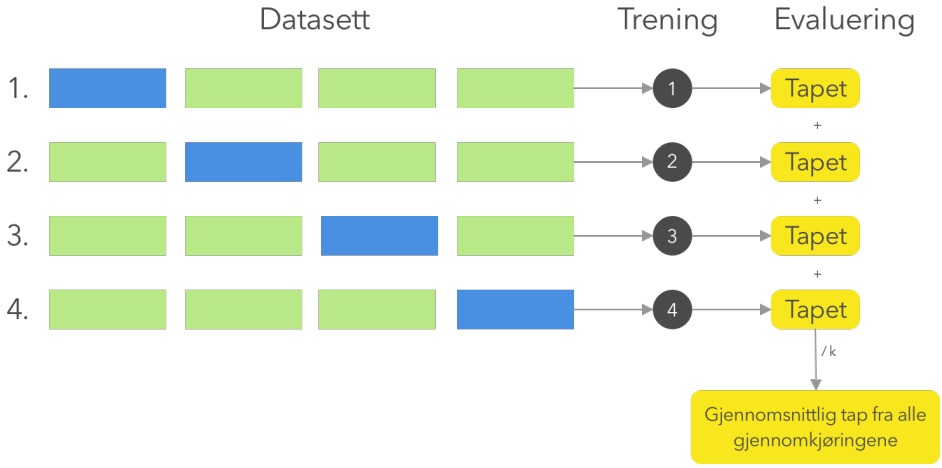
Her må man ha et omfattende datasett og trene den opp slik at den blir god innenfor sitt problem. Den generelle modellen kan da bli trent videre på et annet datasett som består av data relatert til den nye problemstillingen. Resultatet blir så en "spesifikk modell" [25].

Ofte så brukes transfer learning når man vil trene en modell, men ikke har ressurser til å trene modellen fra bunn av. Man kan da finne en modell som er trent på et lignende problem, og trene denne videre.

Vi valgte å ikke ta i bruk transfer learning da vi uansett måtte trene modellene våre fra bunnen av, og vi jobbet kun med en bane. I fremtiden kan det være gunstig å finne en måte å utnytte transfer learning slik at man kan lage en generell modell som videre kan spesifiseres til hver bane i spillet.

5.4.3 Cross-validation

Cross-validation er en måte å dele opp datasettet slik at man får utnyttet all dataen. Ved standard $70\% - 30\%$ oppdeling får man kun trent på 70% av dataen. Med cross-validation får man brukt all dataen gjennom å kjøre flere treningsgjennomganger med forskjellig oppdeling av datasettet.



Figur 5.2: Illustrerer hvordan datasettet blir delt opp forskjellig for hver treningsgjennomgang. Blå rute representerer testdatasett. Som man kan se testes modellen på forskjellig data for hver gjennomgang. I dette eksempelet er $k = 4$

Noe som blir klart med denne teknikken er at man klarer å utnytte all dataen, samtidig som at testingen ikke skjer med data som modellen har trent på. Dette gjør at man får utnyttet all den dataen man har tilgjengelig. Ulempen

er at man i stedet for å kjøre en treningsgjennomgang, må kjøre k antall gjennomganger. Dette kan øke tiden det tar å trene nettverket vesentlig [26]. Vi valgte derfor å ikke ta i bruk denne teknikken i prosjektperdioden.

5.4.4 Balansering av datasett

En trend som vi observerte var at prediksjonene ofte fortsatte i samme retning etter at den faktiske dataen hadde en retningsendring. Et eksempel kan man se på figur 4.11. Her kan man se at prediksjonene rundt $x = 0.6$ utfører retningsendringen for sent.

Dette kan være en indikator på et datasett som ikke er i balanse. Med balanse menes variasjon i dataen. Dersom det forekommer en overvekt av en type data/et scenario anser vi datasettet som ubalansert. En konsekvens er at entiteten som har en overvekt i forekomster vektes tyngre. Ut ifra figur 4.11 så tyder det på at datasettet vårt bestod av en overvekt av sekvensielle posisjonsendringer uten retningsendringer. Dette igjen vil føre til at modellen utformer en bias, og den vil anta at sannsynligheten for at spilleren beveger seg i rett retning er høyere enn at spilleren utfører retningsendring.

Balansering av sekvenssentitivt datasett er en utfordring, spesielt siden dataen vi jobber med representerer posisjoner. Fjerning eller generering av data vil direkte påvirke hvordan modellen tolker dataen. Dersom vi kutter bort deler av posisjonsprogressjonen så vil dataen representere en spiller som hopper fra et punkt til et annet, uten å være i setback. Og ved generering av data så vil datasettet ende opp med datapunkter som ikke samsvarer med en realistisk gjennomspilling.

Det finnes metoder for å balansere et sekvenssentitivt datasett, både fjerning eller generering av data. Man kan også endre på tapsfunksjonen. Balansering av datasett, spesielt tidsserier er komplisert [27]. Ved trening observerte vi at modell 4.11 viste forbedringer over modell 4.10, selv med mindre trening. Grunnet dette resultatet valgte vi å nedprioritere balanseringen og heller prioritere videreutvikling av modellen 4.11.

5.5 Begrensninger

5.5.1 Datasett

En mulig løsning for å få bedre treffsikkerhet på modellen er å trene på et større datasett. Med et større datasett så ville modellen fått trent på flere ulike variasjoner, som igjen ville hjulpet modellen med å ta gode avgjørelser når den møter scenarioer som den ikke har sett før.

Vi fikk tilgang til to datasett. Det første var et datasett som bestod av ti matcher med to spillere. Det var dette datasettet som vi jobbet mest på. Vi fikk også tilgang til et større datasett på 20 matcher med fire spillere. Dette datasettet hadde også flere attributter. Det andre datasettet ble ikke testet ordentlig da det kom veldig sent i prosjektet, og vi fikk problemer med maskinvare. Modellene fungerte med dette datasettet, men å se hvor godt de presterte ble ikke prioritert på grunn av tid.

Desverre så finnes det ingen gode retningslinjer for hvor stort et datasett bør være, og det finnes måter å utnytte små datasett [28]. Men oftest ser man at større datasett bidrar til bedre treffsikkerhet [29].

Datasett 1 bestod også av noen få attributter; posisjon og inSetback. Posisjonene og inSetback former et mønster hvor posisjonene vil følge tidligere progresjon i revers så lenge inSetback er true. Vi skulle gjerne hatt noen flere attributter som for eksempel inAir. Denne atributten kunne beskrevet en spiller som er i luften, og som dermed ikke kunne hatt retningsendring. Siden fysikken i spillet er forutsigbar kunne dette bidratt til at modellen kunne predikert bedre et slikt scenario.

En annen attributt er skjold. Dersom modellen hadde vist hvor mye skjold spilleren har kunne den antatt om spilleren kommer til å bli satt i setback før det faktisk inntreffer. Dette fordi at den kunne gjenkjent det som en interessant tilstand over en tilstand hvor spilleren har fullt skjold.

Både inAir og skjold var i datasett 2. Desverre så fikk vi ikke trent modellene vår på dette datasettet da vi fikk trøbbel med maskinvare og trening.

5.5.2 Setback

Et problem med denne oppgaven er at setbacks i dette spillet er i seg selv ganske uforutsigbart. En spiller kan havne i setback hvis han/hun blir skutt, men kan også gå inn i setback frivillig. Dette er et problem for algoritmen som

finner interessepunkter, da en som frivillig går inn i setback ikke nødvendigvis er så interessant. Det er også et problem for maskinlæringsmodellene, da det kan være vanskelig å predikere når noen velger å gå inn i setback. En løsning på dette kan være å legge til en ekstra attributt i datasettet som sier om spilleren gikk frivillig inn i setback eller ikke.

Når en spiller går inn i setback etter å ha bli skutt kan også være uforutsigbart. Det avhenger både av om en spiller velge å skyte i et gitt øyeblikk, og om skuddet treffer. Dette er mulig å predikere kort tid fram i tid, men når man skal predikere flere sekunder fram i tid kan dette bli vanskelig. Vi ser at modellene våre noen ganger evner å predikere fremtidige setbacks, og at de noen ganger ikke klarer dette.

5.5.3 Visualisering

Visualisering av resultatene våre viste seg å bli problematisk jo lengre ut i prosjektet man kom. Målet var å fremstille resultatene i en form som var oversiktlig og lettleselig, samtidig som det fanget de viktigste punktene vi ønsket å få frem. Grafene som visualiserer resultatene viser ikke utviklingen av tidsstegene, men følger kun samme tidsserie. En løsning på dette kunne være å skille tidssteg på størrelse/tykkelse på linjene, eller bruke en fargegenerator som markerer linjene med mørkere farge jo lengre ut i tidsserien man kommer.

5.6 Prosjektarbeid og etikk

Dette prosjektet har pågått under et hektisk semester med mye fokus på bacheloroppgaven, balansert med eksamener og prosessen med avsluttende studie. Vi har klart å balansere prioriteringene godt og kommet med et sluttresultat vi er godt fornøyde med. Gruppearbeidet har fungert bra under hele prosjektperioden, med en fast struktur på oppmøte som vi anser som viktig for arbeidsprosessen. Gruppen bestod av studenter med variert erfaring innen relevante tema, men det var ingen problem da eventuelle hindringer var kun et spørsmål unna. En kombinasjon av gode tilbakemeldinger fra veileder og oppgavestiller sammen med beslutninger fra oss gjorde at vi kom oss gjennom en ambisiøs og spennende bacheloroppgave.

Som utviklere har vi et ansvar for å følge etiske retningslinjer. Innen vår profesjonsetikk (computeretikk) handler dette om å sikre at programmer som utvikles ikke blir misbrukt eller skaper negative konsekvenser for brukeren.

I vårt tilfelle har vi fått data fra Riddlebit som stammer fra menneskelige spillere. Det er av personvernshensyn viktig å være forsiktig med slik data. Dataen vi fikk var anonymisert, og vi hadde ingen måte å finne ut hvilken spiller som var knyttet til gitte data.

Det er i dag mye diskusjon rundt maskinlæring og etikk. Det er stort fokus på at vi må være forsiktige med maskinlæring, da det er vanskelig å forutse de etiske konsekvensene. Vi anser ikke dette som et problem i vår oppgave, da den ligger innenfor et spesifikt område som er vanskelig å misbruke.

6 Konklusjon og videre arbeid

6.1 Konklusjon

Målet med denne rapporten var å utforske hvilke maskinlæringsteknikker som kunne brukes til å predikere fremtidige interessepunkter i dataspill, samt å utforske hvor langt fram i tid det var hensiktsmessig å predikere. Dette utførte vi ved å teste flere metoder på data samlet inn fra spillet Setback. Vi har testet ut GRU, CNN, og flere metoder med LSTM.

LSTM er en av de mest brukte teknikkene på tidsserier. Denne teknikken er spesielt god på tidsserier der hendelser avhenger av hva som har skjedd tidligere. Tidligere arbeid viser til gode resultater på lignende problemstillinger som den vi har jobbet med. Vi fant også i dette prosjektet at LSTM gjorde det best.

Vi testet ut flere metoder som brukte grid som input. Dette viste seg å fungere dårlig på denne dataen, da gridene ble for store, og hadde alt for mange nullverdier.

LSTM-modellene ble testet på å predikere ulike antall tidssteg fram i tid. Alle modellene predikerer relativt godt 1 tidssteg fram i tid, men vi ser store forskjeller på hvor godt de predikerer lengre fram i tid. Modell 3 presterer best for alle tidssteg, og blir bedre enn de andre desto lengre tidsgap som skal predikeres. Med denne modellen, med den treningen vi har fått utført, ser det ut til at det er hensiktsmessig å predikere 20 tidssteg fram i tid. Når dette er sagt så ser det ut til at også 40 tidssteg er innen rekkevidde. Da vi har fått noe problemer med å gjennomføre tilstrekkelig trening, kan det antas at denne modellen presterer enda bedre om den får trent tilstrekkelig.

6.2 Videre arbeid

I videre arbeid med dette prosjektet anbefales å jobbe videre med modell 3. Denne modellen fikk aldri trent så mye som ønsket, og hvis den får det kan man nok forvente enda bedre resultater enn det som er vist fram her. Det anbefales også å teste systemet i praksis. Disse modellene ble aldri testet opp mot tilskuersystemet i spillet, og det er dermed vanskelig å si noe konkret om hvor godt de fungerer.

Videre burde det utprøves om flere attributter øker presisjonen til modellen. I et slikt spill er det mange faktorer som avgjør hva som skjer, og vi har

jobbet med relativt få. Attributter som skudd og retning på spiller kan være med å avgjøre hva som kommer til å skje.

Man burde også prøve å trene modellen på vesentlig større datasett enn det som er gjort her. Modellene i dette prosjektet er trent på veldig begrenset data, og dette kan ha stor innvirkning på hvor godt de presterer.

Det kan også være interessant å prøve ut noen av teknikkene vi gikk gjennom i kapittelet om alternative metoder (se kapittel 5.4). Dette er teknikker som kan gjøre treningen til modellene bedre eller raskere, som vi av ulike grunner ikke har brukt i dette prosjektet.

Referanser

[1] PwC Global Entertainment & Media Outlook, "Video games and e-sports." https://www.pwc.com/gx/en/industries/tmt/media/outlook/segment-findings.html, 2018. [Sist besøkt 14 April 2019].

[2] X. Su, "Business analytics techniques (leksjon: Data analytics techniques iini3012 big data." . [Sist besøkt 18 Mai 2019].

[3] skymind.ai, "A beginner's guide to deep reinforcement learning." https://skymind.ai/wiki/deep-reinforcement-learning. [Sist besøkt 18 Mai 2019].

[4] University of Tampere, "Neurocomputing4." http://www.uta.fi/sis/tie/neuro/index/Neurocomputing4.pdf, Ukjent. [Sist besøkt 6 Mai 2019].

[5] University of Tampere, "Neurocomputing2." http://www.uta.fi/sis/tie/neuro/index/Neurocomputing2.pdf, Ukjent. [Sist besøkt 6 Mai 2019].

[6] A. Sharma, "Understanding activation functions in neural networks." https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0, 2017. [Sist besøkt 10 April 2019].

[7] A. Botchkarev, "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology." https://arxiv.org/pdf/1809.03006.pdf, Ukjent. [Sist besøkt 6 Mai 2019].

[8] D. Sabinasz, "Gradient descent and backpropagation." https://www.codingame.com/playgrounds/9487/deep-learning-from-scratch---theory-and-implementation/gradient-descent-and-backpropagation. [Sist besøkt 30 April 2019].

[9] C. Olah, "Understanding lstm networks." https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. [Sist besøkt 29 April 2019].

[10] S. Rathor, "Simple rnn vs gru vs lstm :- difference lies in more flexible control." https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57, 2018. [Sist besøkt 10 Mai 2019].

[11] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks." https://arxiv.org/pdf/1701.05923.pdf, 2017. [Sist besøkt 30 April 2019].

[12] E. Reppel, "Visualizing parts of convolutional neural networks using keras and cats." https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59, 2017. [Sist besøkt 13 Mai 2019].

[13] A. Karpathy, "Convolutional neural networks (cnns / convnets)." http://cs231n.github.io/convolutional-networks/. [Sist besøkt 13 Mai 2019].

[14] S. Hladky and V. Bultiko, "An evaluation of models for predicting opponent positions in first-person shooter video games." http://www.csse.uwa.edu.au/cig08/Proceedings/papers/8071.pdf, 2008. [Sist besøkt 28 April 2019].

[15] B. Geisler, "An empirical study of machine learning algorithms applied to modeling player behavior in a "first person shooter" video game." https://pdfs.semanticscholar.org/86a9/d4762c5cb49d2fd1f594fae3409e746d5155.pdf, 2002. [Sist besøkt 30 April 2019].

[16] C. Ballinger, S. Liu, and S. Louis, "Identifying players and predicting actions from rts game replays." https://www.cse.unr.edu/~simingl/papers/publish/Identifying_Players_and_Predicting_Actions_from_RTS_Game_Replays.pdf, 2014. [Sist besøkt 15 April 2019].

[17] OpenAI, "Openai five." https://openai.com/blog/openai-five/, OpenAI Blog, 2018. [Sist besøkt 15 April 2019].

[18] M. B. Hagen, "Ruteprediksjon av maritim trafikk med nevrale nettverk." NTNU, IDI, "https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2564787/19941_FULLTEXT.pdf?sequence=1&isAllowed=y", 2018. [Sist besøkt 30 April 2019].

[19] L. Drevland and P. Finseth, "Evaluating machine learning methods forcity bike demand prediction in oslo." NTNU, IDI, "https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2564786/17987_FULLTEXT.pdf?sequence=1&isAllowed=y", 2018. [Sist besøkt 28 April 2019].

[20] R. Fu, Z. Zhang, and L. Li, "Using lstm and gru neural network methods for traffic flow prediction." https://www.researchgate.net/

profile/Li_Li240/publication/312402649_Using_LSTM_and_GRU_neural_network_methods_for_traffic_flow_prediction/links/5c20d38d299bf12be3971696/Using-LSTM-and-GRU-neural-network-methods-for-traffic-flow-prediction.pdf, 2016. [Sist besøkt 6 Mai 2019].

[21] S. Lee, A. Isaksen, C. Holmgård, and J. Togelius, "Predicting resource locations in game maps using deep convolutional neural networks." http://julian.togelius.com/Lee2016Predicting.pdf, 2016. [Sist besøkt 03 Mai 2019].

[22] S. K. Sashank J. Reddi, Satyen Kale, "On the convergence of adam and beyond." https://openreview.net/forum?id=ryQu7f-RZ. [Sist besøkt 6 Mai 2019].

[23] J. F. Puget, "And the most popular language for machine learning is...." https://dzone.com/articles/and-the-most-popular-language-for-machine-learning, 2017. [Sist besøkt 14 April 2019].

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting." http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf, 2013. [Sist besøkt 14 Mai 2019].

[25] S. Martin, "What is transfer learning?." https://blogs.nvidia.com/blog/2019/02/07/what-is-transfer-learning/, 2019. [Sist besøkt 14 Mai 2019].

[26] J. Schneider, "Cross validation." http://www.cs.cmu.edu/~schneide/tut5/node42.html. [Sist besøkt 19 Mai 2019].

[27] G. Lemaître, F. Nogueira, and C. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning." http://www.jmlr.org/papers/volume18/16-365/16-365.pdf, 2017. [Sist besøkt 15 Mai 2019].

[28] A. Beam, "You can probably use deep learning even if your data isn't that big." https://beamandrew.github.io/deeplearning/2017/06/04/deep_learning_works.html, 2017. [Sist besøkt 15 Mai 2019].

[29] M. Johnson and D. Nguyen, "How much data is enough? predicting how accuracy varies with training data size." http://web.science.mq.edu.au/~mjohnson/papers/Johnson17Power-talk.pdf, 2017. [Sist besøkt 15 Mai 2019].

A Ordliste

backpropagation Traversering av det neurale nettverket og justering av vekter underveis for å øke treffsikkerheten. 8, 11, 23

bias En konstant som brukes for å "forskyve" en funksjon. $f(x) = x + 1$ hvor 1 er bias. 7

cluster analysis Teknikk for statistisk data analyse. 4

epoch En hel gjennomkjøring av nettverket med forward propagation og backpropagation. 9

feature engineering Behandling av atributter i et datasett for å forbedre datasettet. 5

features En feature er det samme som en attributt, et datasett kan bestå av en eller flere features. 5

first-person shooter Førstepersonsskytespill er en sjanger innen dataspill der man ser ut fra øynene til karakteren man spiller. 1

forward propagation Traversering av det neurale nettverket for å få et resultat basert på input data. 6, 7, 11

gradient descent En optimaliseringsalgoritme som brukes for å oppdatere vektene under trening av maskinlæringsmodeller. 10, 11, 23

grid Rutenett. 25, 50

mean squared error En metode for å regne ut tap. 9

non-player character Ikke-spillerstyrt spiller. 1

overfitting En maskinlæringsmodell som blir for tilpasset dataen den er trent på, og dermed håndterer data som blir for ulik treningsdataen dårlig. 5, 44

reinforcement learning Forsterkende læring, en type maskinlæringsalgoritme som definere en løsning for så å bruke belønninger dersom algoritmen kommer nærmere løsningen. 4

setback En funksjon i spillet Setback der karakteren havner i en tidligere posisjon den var i. 2, 20, 47

supervised learning Veiledet læring, en type maskinlæringsalgoritme som finner en funksjon basert på inndata og utdata. 4

unsupervised learning Ikke-veiledet læring, en type maskinlæringsalgoritme som baserer seg på inndata uten et tilhørende utdata. 4

B Akronymer

ANN artifical neural network. 16

CNN convolutional neural network. 14, 26, 50

CSV comma separated values. 22, 28

FPS first-person shooter. 1, 16

GRU gated recurrent units. 14, 30, 50

IDE integrated development environment. 29

LSTM long short-term memory. 12, 14, 23, 30, 50

MSE mean squared error. 9, 23, 39

NPC non-player character. 1, 16

ReLU rectified linear unit. 6

RNN recurrent neural network. 12, 14

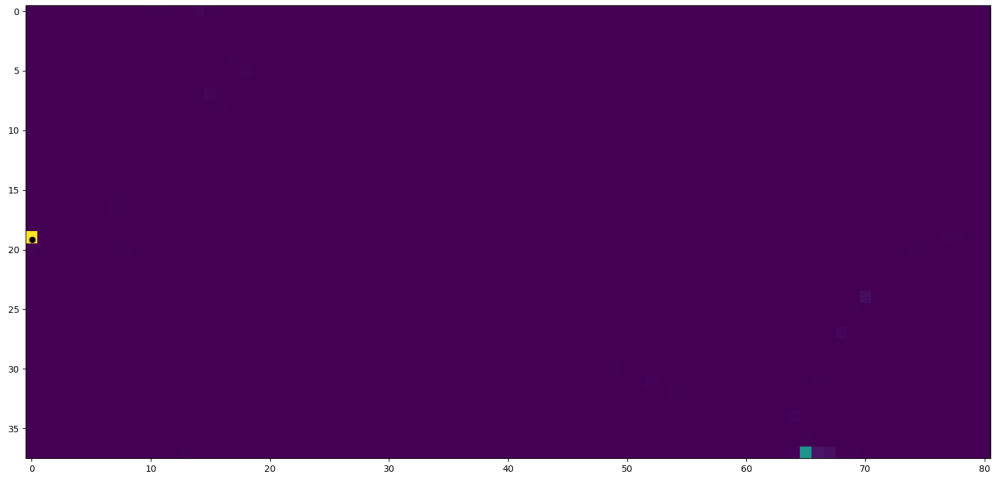
C Original oppgavetekst

Arbeidstittel	DeepSpec
Hensikten med oppgaven: Hensikten med oppgaven er å lage en forbedret "spectating"-opplevelse i spill samt å utforske hvordan vi kan anvende maskinlæring på spesielle typer aktivitetsdata. Prosjektet skal utvikles med generalisering i fokus, noe som gjør at sluttproduktet vil kunne fungere som et noe spesialisert prediksjons-system.	
Kort beskrivelse av oppgaveforslag: Posisjonering av virtuelle kameraer basert på nylig aktivitet på en bane i et 3D flerspiller konkurranse-spill. I tillegg til klassisk programvareutvikling kan prosjektet innebære representasjon og behandling av aktivitetsdata, maskinlæring og statistisk analyse, samhandling mellom flere kameraenheter.	
Oppgaven passer for:	Bacheloroppgave
Kan oppgavestiller stille arbeidsplass med nødvendig utstyr og programvare:	All programvare er gratis, kan kanskje stille med deltids-plass.
Begrensninger i tilgjenglighet av opplysninger o.l:	Oppgavetakere vil signere en NDA, men det burde ikke påvirke prosjektet eller rapporten
Opplysninger om oppgavestiller	
Navn på bedrift eller organisasjon:	Riddlebit AS
Addresse:	Munkegata 58
Postnummer:	7011
Poststed:	Trondheim
Navn på kontaktperson/veileder:	Jonathan Jørgensen
Telefon:	48206901
Epost:	jonathan@riddlebit.net
Navn på kontaktperson 2/veileder 2:	Ole Christian Eidheim

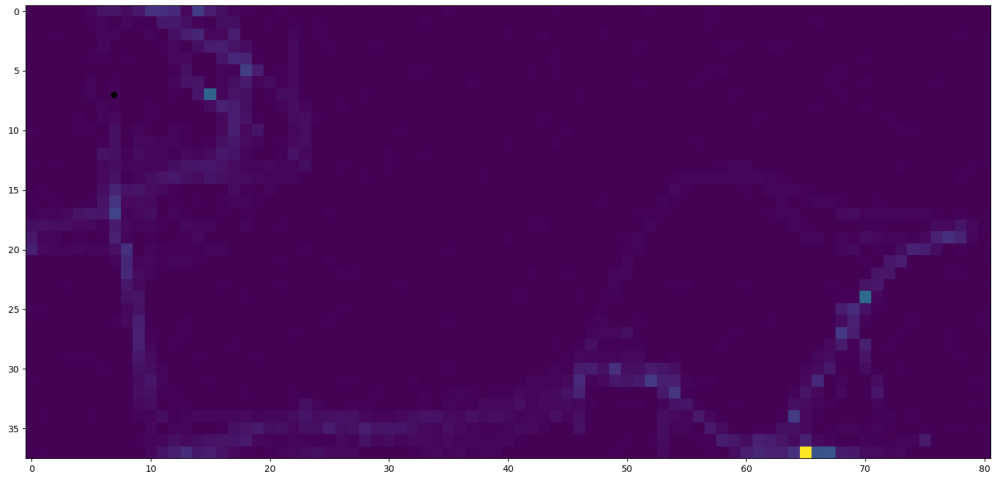
D Større visualiseringer av resultater

Grid

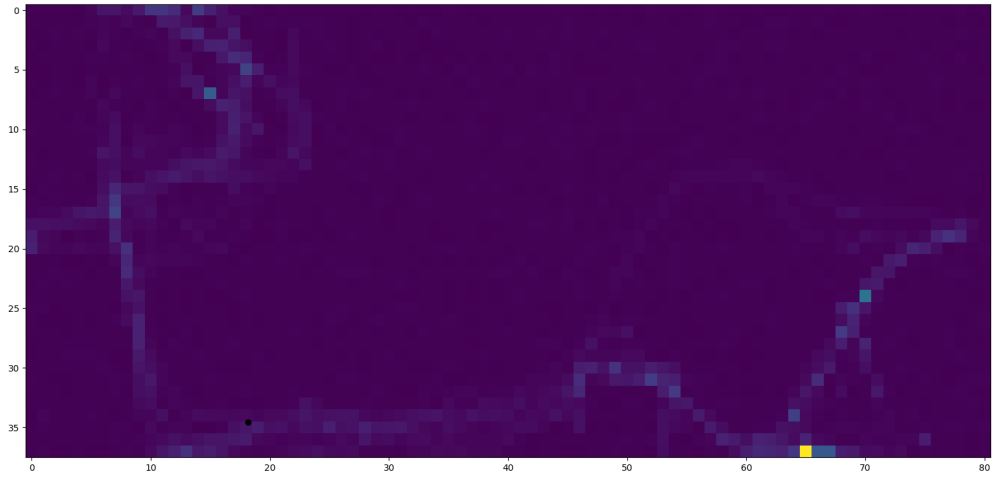
Figur 4.2



Figur 4.3

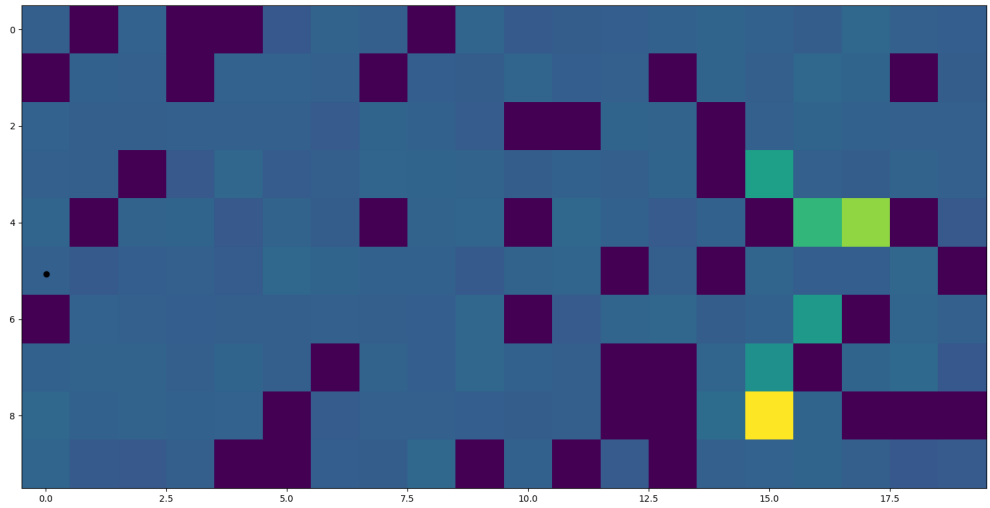


Figur 4.4

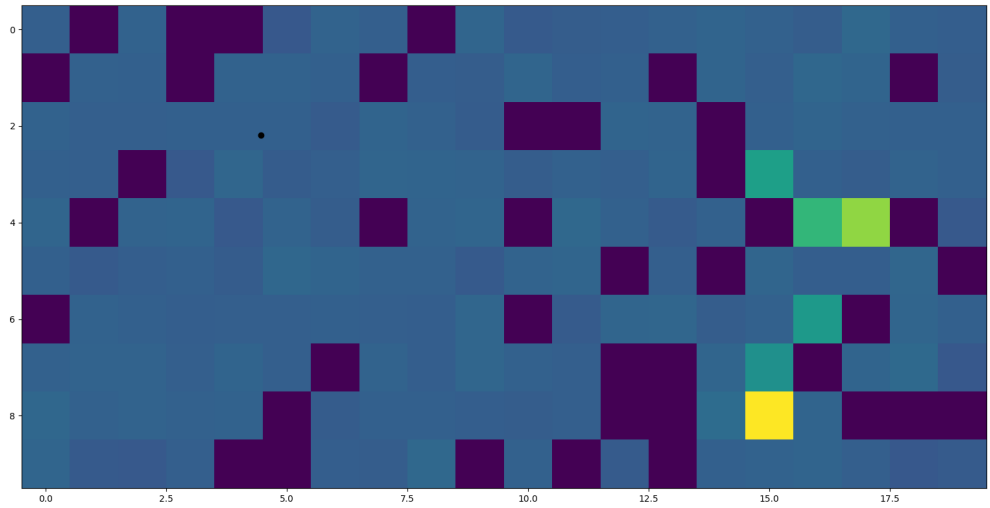


CNN

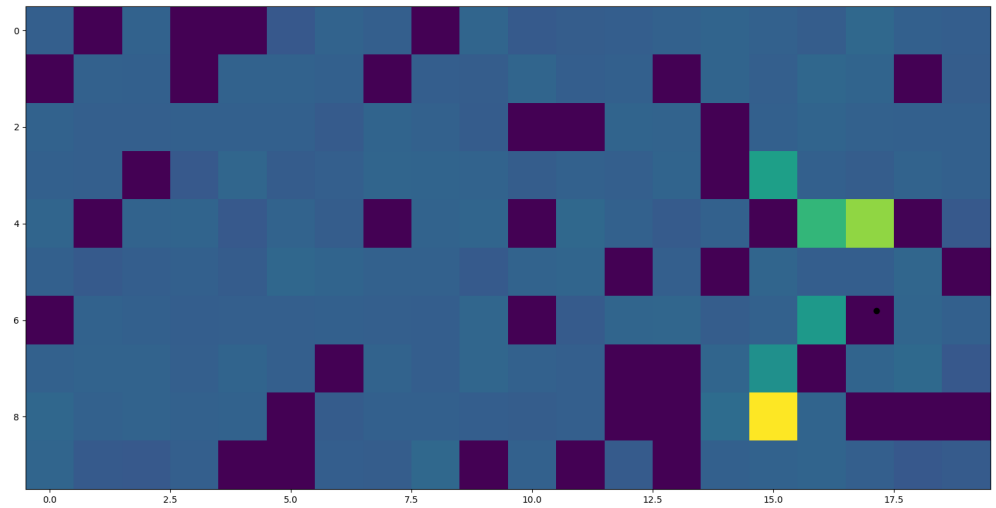
Figur 4.5



Figur 4.6

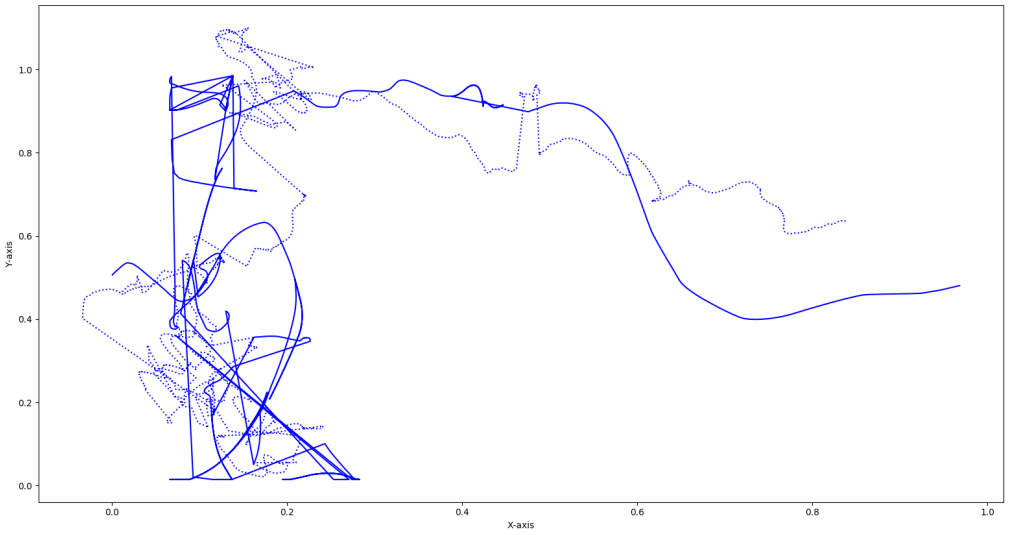


Figur 4.7



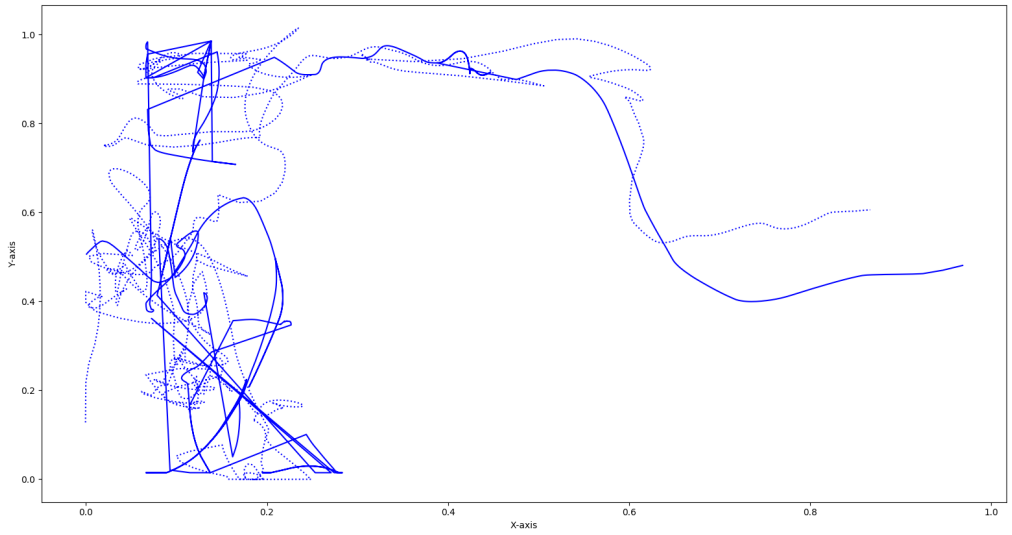
GRU

Figur 4.8

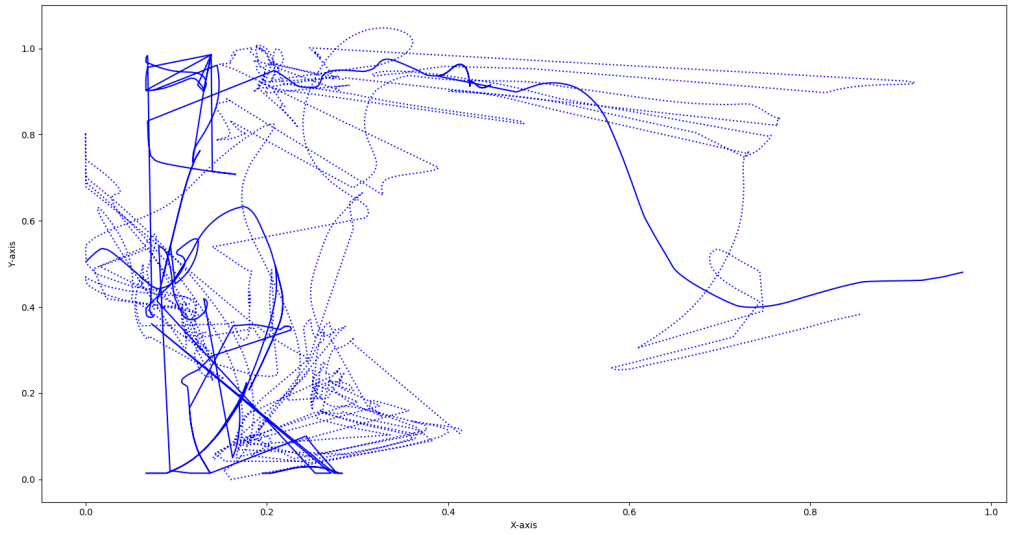


LSTM

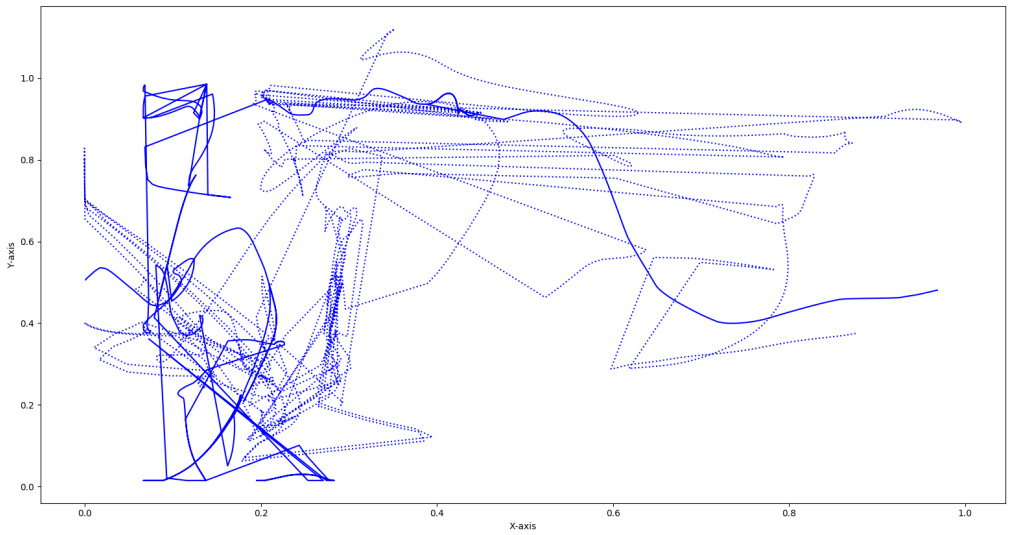
Figur 4.9a



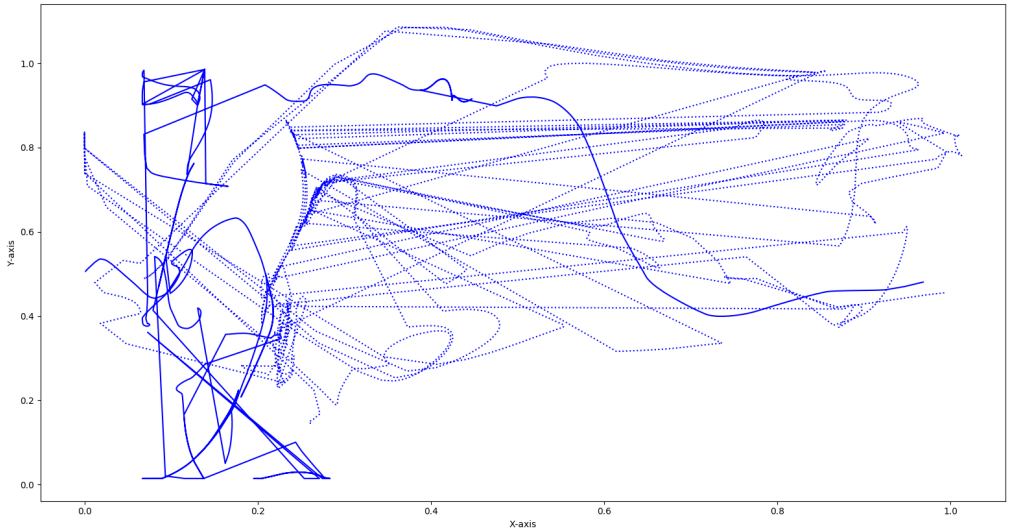
Figur 4.9b



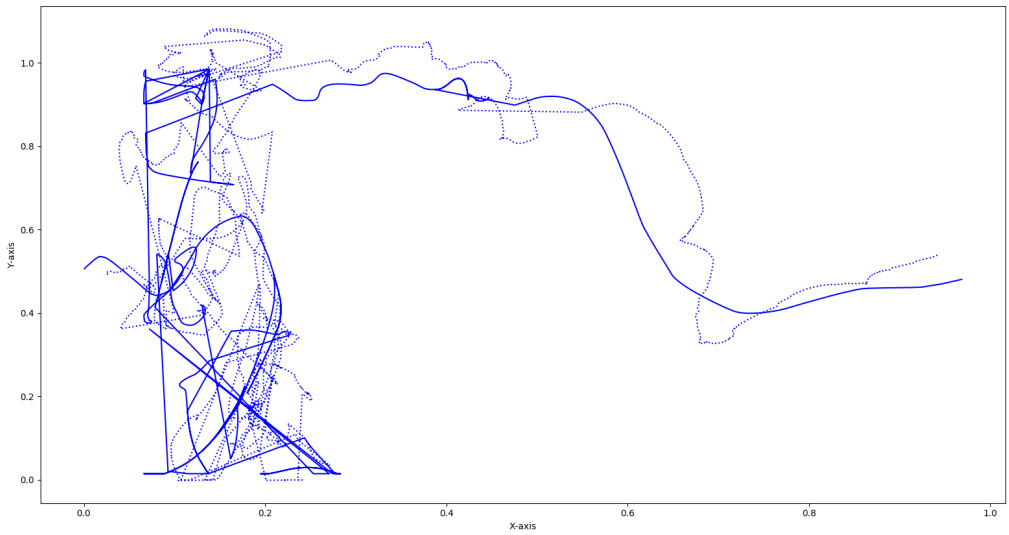
Figur 4.9c



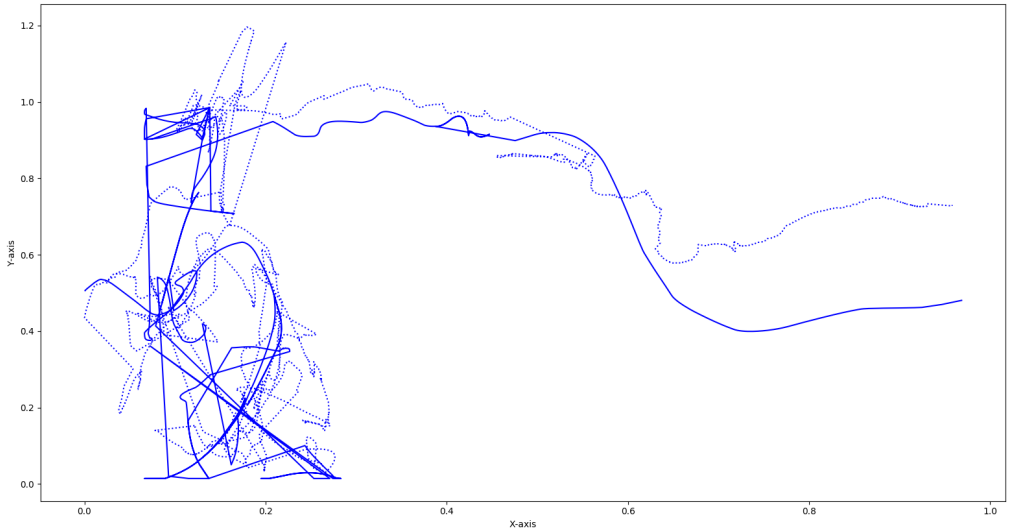
Figur 4.9d



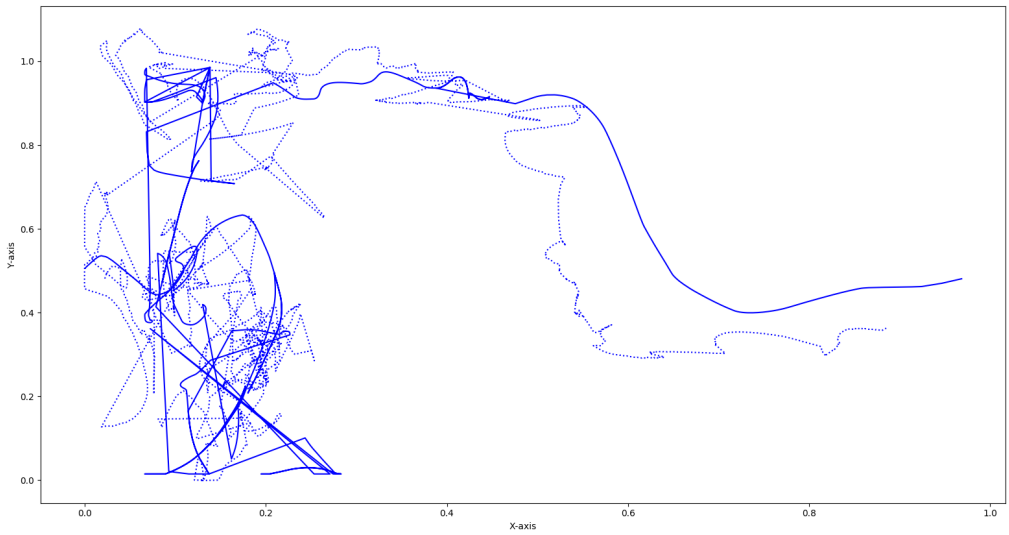
Figur 4.10a



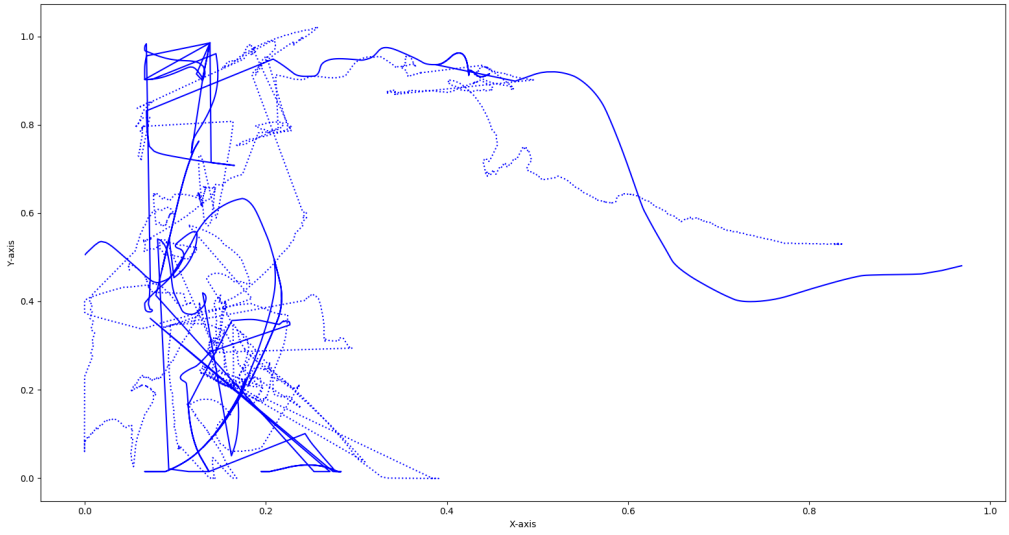
Figur 4.10b



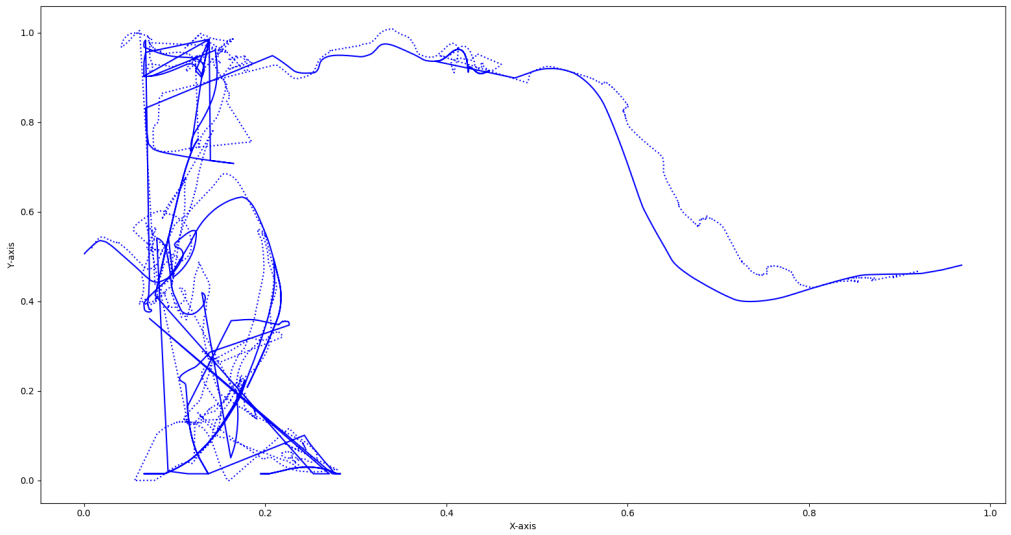
Figur 4.10c



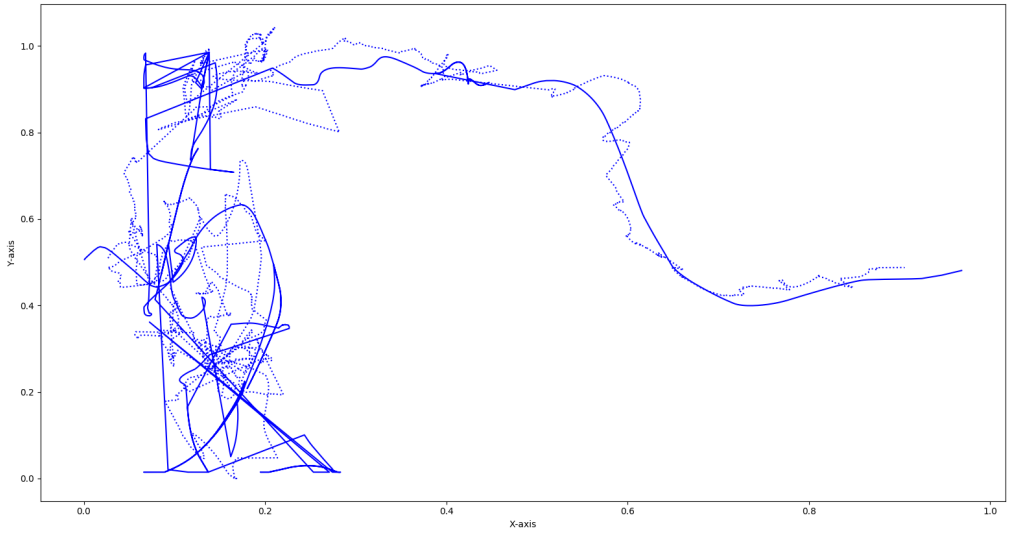
Figur 4.10d



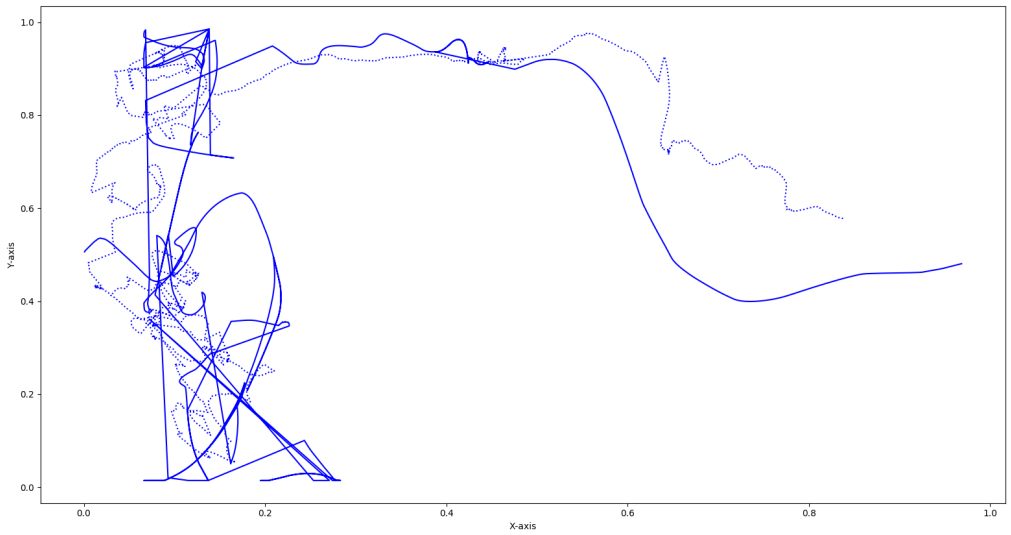
Figur 4.11a



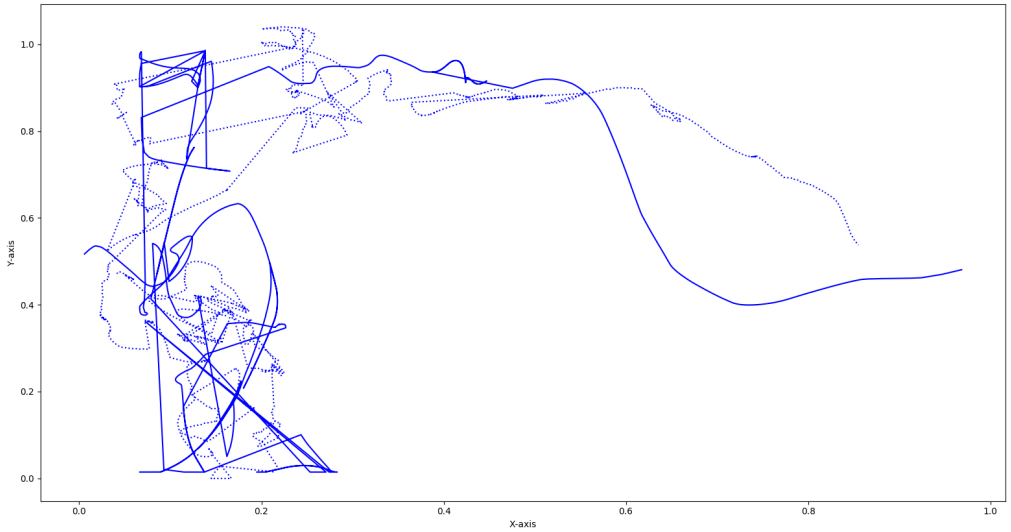
Figur 4.11b



Figur 4.11c



Figur 4.11a



NTNU
Kunnskap for en bedre verden