Kandidatuppsats

Jämförelse mellan neurala nätverk baserad AI och state-of-the-art AI i racing spel

Simon Karlsson, Christopher Jensen

Sammanfattning

Denna rapport jämför prestandan mellan state-of-the-art AI-botar i racing spelet TORCS och en AI-bot som kör med hjälp av ett artificiellt neuralt nätverk (ANN-bot). ANN-boten, som implementerades som en del av arbetet, använder en feedforward arkitektur och backpropagation för inlärning. Ett separat program som användes för att träna det neurala nätverket med träningdata som spelats in från TORCS implementerades också. Som state-of-the-art AI-botar användes AI-botar som har använts i en tävling.

De fyra AI-botarna testades på åtta olika banor och data om hur lång tid varje varv tog och hur snabbt AI-botarna körde sparades och sammanställdes. Resultaten visar att på banorna som ANN-boten klarar av att köra runt så är ANN-boten snabbare än en den långsamaste state-of-the-art boten, men ANN-boten klara inte av majoriteten av banorna som den testades på. Anledning till detta var antagligen brist på varierande träningsdata.

Simon Karlsson
Minervavägen 20
s91k@hotmail.com

Christopher Jensen
Minervavägen 20
chrippe10@hotmail.com

Handledare: Johan Hagelbäck
ISSN-nummer:

1. Introduktion

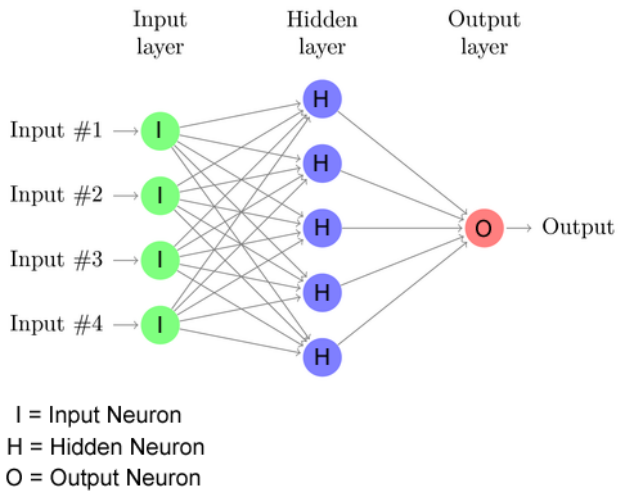
I den här rapporten kommer prestandan hos en AI-bot som använder artificiella neurala nätverk jämföras med prestandan hos state-of-the-art AI-botar som inte använder artificiell neurala nätverk i racing spelet TORCS. Vi kommer använda termen ANN-Bot för vår AI-bot som använder sig av artificiella neurala nätverk .

1.1 Bakgrund

1.1.1 Artificiella Neurala Nätverk

Ett artificiellt neuralt nätverk är en datastruktur som försöker efterlikna hur en hjärna är uppbyggd. Nätverket är uppbyggt av artificiella neuroner, som också kallas för perceptroner. En perceptron består av ett värde och en aktiveringsfunktion, som en del i ett nätverk så har den också minst en koppling som används för att skicka data till perceptroner i andra lager. En koppling består av ett värde och en vikt. När en perceptron aktiveras så summerar den värdena i varje koppling multiplicerat med kopplingens vikt. Summan skickas sedan till en aktiveringsfunktion och resultat blir perceptronens värde som används som värdet i en koppling till en annan perceptron eller som utdata [1].

Ett neuralt nätverk är uppbyggt av flera perceptroner som är organiserade i lager. Ett nätverk behöver minst två lager, ett för input och ett för output. Ett neuralt nätverk kan också ha dolda lager, vilket är lager mellan input och output lagren. Den vanligaste layouten är att ha tre lager och det behövs sällan mer än ett dolt lager [2], Figur 1 visar en överblick på hur ett artificiellt neuralt nätverk kan se ut.



Figur 1: Exempel på ett neuralt nätverk

När ett neuralt nätverk exekveras tar nätverket emot input data som sedan skickas till varje perceptron i input lagret. Därefter beräknas aktiveringsvärdet för varje perceptron och detta propageras vidare genom nätverket tills output värden fås [2].

Resultatet av detta är att varje utdata-värde egentligen är resultatet av en matematisk formel. För att formeln ska ge rätt resultat måste nätverket tränas, vilket i praktiken innebär att man modifierar vikterna på kopplingarna mellan perceptronerna. Detta kan göras manuellt i väldigt simpla nätverk och enkla problem men oftast använder man olika tekniker som bakåtpropagering.

1.1.2 Aktiveringsfunktioner

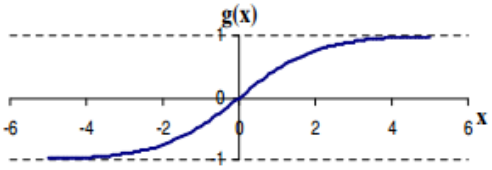
Det finns flera olika aktiveringsfunktioner som används i neurala nätverk. Aktiverings funktionens syfte är att beräkna om en perceptron är aktiv eller inte [3].

Bipolar Sigmoid

Bipolar sigmoid aktiverings funktionen utseende visas i Formel 1:

$$g(x)=\frac{1-e^{-x}}{1+e^{-x}} \qquad (1)$$

Och passar bra till applikationer som vill ha outputs mellan (-1, 1), Se Figur 2.



Figur 2: Bipolar sigmoid funktionens utseende.

1.1.3 Feedforward-nätverk

Det finns flera olika arkitekturer som beskriver hur ett neuralt nätverk kan vara uppbyggt. I ett feedforward-nätverk rör sig data bara i en riktning, dvs. perceptronerna i ett lager skickar bara data till perceptronerna i nästa lager, till skillnad från t.ex. ett återkopplat nätverk där utdata skickas tillbaka som indata. Se Figur 1 för ett exempel på ett feedforward-nätverk.

1.1.4 Inlärning och bakåtpropagering

Träning av neurala nätverk kan delas upp i två huvudkategorier: övervakad och oövervakad. Övervakad inlärning innebär att man har en mängd träningsdata som innehåller indata och känd utdata. Vid träning modifieras nätverket för att bli så bra på att matcha indatan med den kända

utdata som möjligt. I oövervakad träning har man ingen känd träningsdata och testar istället olika alternativ för att se hur bra de är [2].

Bakåtpropagering är en metod för övervakad träning. För att träna nätverk med bakåtpropagering så propageras data genom nätverket med indata från träningsdatan. När detta är gjort så jämförs utdatan med det korrekta resultatet från träningsdatan och ett felvärde för varje utdata beräknas. Detta felvärde bakåtpropageras sedan genom nätverket för att räkna ut felvärdet för varje vikt i nätverket. När det är gjort så modifieras vikterna för att ge ett resultat som är närmare det korrekta resultatet [2], pseudokod för bakåtpropagering visas i Algoritm 1.

```
for each example e in the training data
      execute the network using input from e
      compute error in output layer // Error=output från nätverket - output från e
      for each layer in network starting with output layer
            compute delta for all weights
      adjust the weights in the network // Modifiera vikterna baserat på delta
```

Algoritm 1

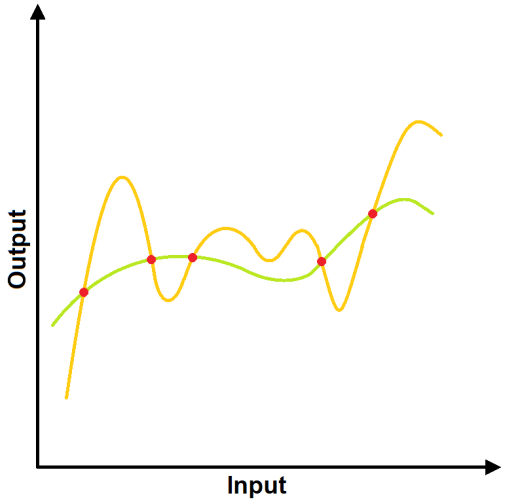
Träningen upprepas tills ett bestämt antal iterationer har körts eller någon annan gräns har nåtts, till exempel att summan av felvärdena från output lagret från flera tester är tillräckligt låg, exempel på beräkning av felvärde visas i Formel 2.

$$E = \sum_{i=1}^{n} (oa_i - oe_i) \qquad (2)$$

E = felvärde, n = antal output-värden, oa = output från nätverket, oe = output från träningsdatan.

1.1.5 Problem i Artificella Neurala Nätverk

Att se till att nätverket lär sig rätt saker är ett stort problem inom neurala nätverk. Ett nätverk kan till exempel hitta ett mönster i träningsdatan som inte är samma som det användaren ville att nätverket skulle hitta, men som ändå ger rätt svar vid träning. Risken för att detta uppstår kan minskas genom att använda ett större antal och mer varierande träningsdata. Ett exempel på ett nätverket som har lärt sig fel, men ändå får rätt vid träning visas i Figur 3.



Figur 3. Röda prickar är träningsdata, den gröna linjen är rätt funktion, orange linjen är resultatet från nätverket.

Det finns också ett problem som kallas för overfitting, vilket händer om nätverket tränar för mycket mot träningsdata och därför blir mindre generellt vilket leder till ett sämre resultat när det testas mot data som nätverket inte har tränats med. Risken för överträning kan minskas genom att avbryta träningen tidigare och låta nätverket ge större fel på testerna [2].

1.1.6 Artificiella Neurala Nätverk i spel

Generellt sett använder spel-AI mest statiska tekniker, som tillståndsmaskiner, istället för tekniker som anpassar sig efter motståndaren [1]. För att anpassa en spel-AI till en motståndare måste en statisk AI programmeras om. Motsatsen är adaptiva tekniker som kan lära sig själva och anpassa sig under spelets gång, t.ex. neurala nätverk som kan ges ny träningsdata.

Neurala nätverk användes till exempel för att styra bilar i Colin McRae Rally 2 eftersom det visade sig vara för svårt att skapa regler som kunde hantera att svänga i kurvor i lera och grus [4]. Colin McRae Rally 2 använde "offline learning" vilket innebär att nätverket tränades av utvecklaren innan det släpptes och sedan var statiskt i versionen som såldes.

Att låta nätverket tränas baserat på vad spelaren gör i spelet kan användas för att skapa intressanta spel, men det finns också risker. Till exempel använde spelet Black and White neurala nätverk för att styra beteendet hos ett monster. Nätverket användes för att styra högnivå beslut hos monster, t.ex. vad den äter, och spelaren kunde träna monstret att göra vissa saker. Dock kunde ett problem som kallas för "catastophic unlearning" uppstå, vilket innebär att ett neutralt nätverk lär sig något vilket leder till att all tidigare inlärning förstörs [2]. Problemet beror på att neurala nätverk kan anpassar sig för mycket efter ny information, vilket blir ett problem om nätverket tränas med ny data under körning. En lösning som kan reducera problemet är träna

nätverket med ny data och sedan träna nätverket med en mängd gammal data för att "påminna" nätverket vad det kunde [5].

1.1.7 Relaterade arbeten

Det finns flera arbeten där artificiella neurala nätverk används för att styra fordon. I "A human-like TORCS controller for the Simulated Car Racing Championship" [6] använder man också neurala nätverk för att köra bilar och precis som det här arbetet så använder de TORCS. Skillnaden är att dem försöker göra en AI-bot som beter på ett så mänskligt sätt som möjligt till skillnad från att bara implementera en AI-bot som gör fort. I deras implementation använder de också en indirekt kontroller, vilket i det här fallet innebar att de använde neurala nätverk för att räkna ut i vilken riktning AI-boten skulle köra i och sedan användes vanligare statiska tekniker för att faktiskt få bilen att köra i den riktningen.

Benoit Chaperot och Colin Fyfe testade att använda artificiella neurala nätverk för att köra en motocross i ett spel i " Motocross and artificial neural networks" [7]. De testade två metoder för att träna nätverk, ett som byggde på backpropagation och ett som byggde på evolutionära algoritmer. Deras tester visade att nätverket som tränats med backpropagation kunde klara testerna snabbast, men de noterade också nackdelarna med backpropagation: behovet av träningsdata och att nätverket som använde backpropagation inte var tränat för att hantera oförväntade händelser, till exempel att återhämta sig efter en krash.

Chaperot och Fyfe undersökte också om mer avancerade varianter av backpropagation och evolutionära algoritmer gav bättre resultat i "Improving Artificial Intelligence In a Motocross Game" [8]. Dock är de metoderna för komplicerade för att hinna implementera i det här arbetet och istället kommer backpropagation liknande den som användes i deras första arbete att användas.

1.1.8 TORCS

The Open Racing Car Simulator (TORCS) är en 3D bilspels simulator som stöds på alla Linux arkitekturer (32/64 bit), Mac OS X och Windows (32/64 bit). Utvecklingen av TORCS började 1997 av Christophe Guionneau och Eric Espié. TORCS är licensierad under GNU GPL och har öppen källkod som tillåter användaren att skapa sin egen AI-bot i c/c++, en bild på TORCS visas i Figur 4.

Figur 4: Bild av TORCS

Eftersom TORCS är modulerbar och har öppen källkod så har TORCS använts som bas för forskning inom en mängd olika områden [9]. TORCS använder sig av en sofistikerad fysik motor som tar hänsyn till många aspekter av hur en riktigt bil fungerar och hur den interagerar med omgivningen som bl.a. aerodynamik och bränsleförbrukning [10]. Tävlingar för AI-botar körs regelbundet på ett flertal olika banor utspridda under en längre period vilket ger de tävlande chans att modifiera sina AI-botar mellan banorna. Tävlingarna är öppna till de som vill tävla och uppfyller kraven som finns [11].

1.2 Syfte och mål

Målet med arbetet är att implementera en spel-AI som använder sig av neurala nätverk, träna nätverket med hjälp av träningsdata från olika typer av banor och sedan jämföra den mot AI-botar som har använts i tävlingar och presterat olika bra för att se om vår AI-bot kan mäta sig tidmässigt med de andra AI-botarna. Anledningen till att vi valde att implementera ett artificiellt neuralt nätverk i TORCS var för att vi ansåg att artificiella neurala nätverk var ett intressant område att lära sig mer om och TORCS har använts tidigare inom forskning och är lätt att implementera nya AI-botar.

1.3 Forskningsfråga

1. Hur snabb är en AI-bot baserad på neurala nätverk jämfört med state-of-the-art AI-botar för TORCS?

1.4 Metodik

För att svara på forskningsfrågan kommer en kvantitativ metod användas. Detta görs genom att implementera en AI-bot och testa den och tre olika state-of-the-art AI-botar på åtta olika banor. Resultaten från testerna kommer sedan sparas, sammanställas och diskuteras.

1.4.1 Implementation av AI

För att kunna göra tester så måste en AI-bot som använder neurala nätverk implementeras i TORCS. AI-boten bygger på en modifierad version av spelar-boten, vilket är en bot i TORCS som styrs av spelaren, där två neurala nätverk används för att köra bilen, ett nätverk hanterar styrning och ett annat hanterar gas och broms. Anledning till att en modifierad version av spelar-boten används är att det blir enkelt att testa ett nätverk separat, till exempel att genom att låta ett neuralt nätverk hantera styrning medan användaren sköter gas och broms.

Flera olika sorters indata kommer användas, de flesta använder funktioner som redan finns i TORCS:

1. Bilens hastighet.
2. Vinkel mot mittlinjen i det nuvarande segmentet.
3. Vinkel mot mittlinjen i ett segment framför bilen.
4. Avstånd till mittlinjen i det nuvarande segmentet.
5. Avstånden till vänster och höger kant i det nuvarande segmentet.

Avståndssensorerna används för att mäta avståndet mellan bilen och den närmaste kanten i den riktning som sensorn pekar.

De neurala nätverken kommer använda en feedforward arkitektur och kommer vara statiska under körning detta innebär att träningen av nätverket sker utanför AI-boten och nätverken i AI-boten kommer sedan läsa in en fil som beskriver hur nätverket är uppbyggt och värdena på dess vikter.

För att genomföra träningen kommer ett separat program implementeras som använder backpropagation för att modifiera vikterna i nätverket. Eftersom backpropagation är en övervakad träningsmetod så behövs en mängd träningsdata spelas in. Därför kommer ytterligare en modifierad version av spelar-boten implementeras som spelar in input och output värden flera gånger varje sekund som kommer användas av de neurala nätverken.

Målet med implementation är att implementera en AI-bot som kan köra runt banan den tränats att köra på utan att krocka med kanterna och på en rimlig tid.

1.4.2 Testning och utvärdering

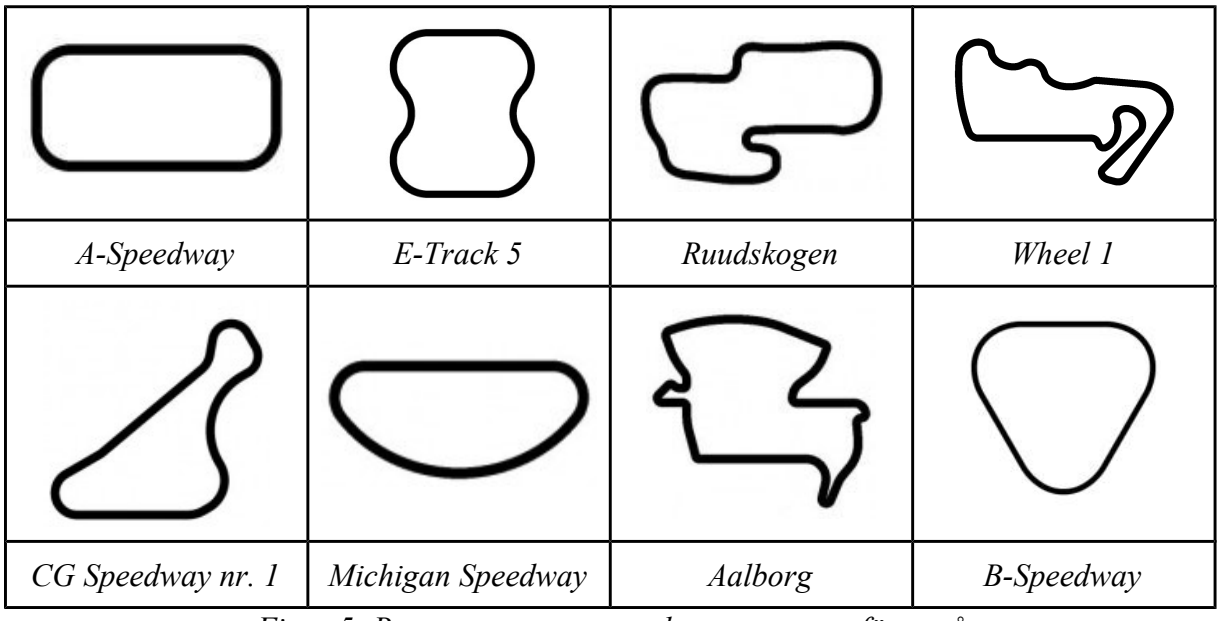
ANN-boten och en AI-botarna som har kört och vunnit i tävlingar kommer testas genom att låta dem köra på åtta olika banor. Under testerna kommer varje AI-bot köra ensam på banan, på grund av detta kommer inga tester på nätverkets förmåga att hantera kollisioner och omkörningar av andra bilar att göras. Tiden det tog för varje AI-bot att klara varje varv på banan och den högsta hastigheten bilen nådde kommer sparas. Om en AI-bot fastnar kommer den att få en

minut på sig att börja köra igen, om den inte har börjat köra efter en minut så kommer loppet att avbrytas och avbrottet samt hur långt AI-boten kom kommer att noteras.

Varje bana kommer testas fem gånger av varje AI-bot för att mäta eventuella skillnader i resultaten från varje lopp. När testerna är avklarade kommer resultatet att sammanställas och utvärderas för att jämföra vilka skillnaderna i prestanda och beteende mellan de olika AI-botarna är.

AI-botarna som inte använder neurala nätverk och som ska representera state-of-the-art AI-botar i TORCS är dandroid_2012 som vann TORCS Endurance World Championship 2012, berniw_2004 som kom näst sist och blacky_2012 som kom på en placering mellan de tidigare nämnda AI-botarna [11]. Den enda modifiering av AI-botarna som kommer göras är att bilen den använder kommer ändras till samma bil som AI-boten med neurala nätverk använder.

Testerna kommer genomföras på åtta olika banor. Två av dessa användes för att träna ANN-boten: A-Speedway och E-Track 5. Totalt så bestod tävlingen som state-of-the-art botarna deltog i av tio olika banor och av dem valde vi fem banorna för att använda i testet: CG Speedway number 1, Ruudskogen, Wheel 1, Michigan Speedway och Aalborg. Den sista banan, B-Speedway, har varken använts för träning eller i tävlingen. Banorna valdes för att försöka ge varierande situationer som AI-botarna måste hantera och för att testa AI-botarna på banor de har och inte har tränats eller anpassats för. Speedway banorna är breda och har mjuka kurvor, medan dem andra banorna är smalare och har skarpare kurvor. Figur 5 visar hur de olika banorna ser ut.



Figur 5: Banorna som testerna kommer genomföras på.

1.5 Uppsatsens struktur

1. Introduktion
Innehåller bakgrunden till arbetet, syftet med arbetet, forskningfrågan som ska besvaras och metoden som kommer användas.

2. Utförande
Beskriver hur metoden som presenterades i introduktionen användes, hur ANN-boten implementerades och tränades och hur testerna genomfördes.

3. Resultat
Resultaten från testerna presenteras.

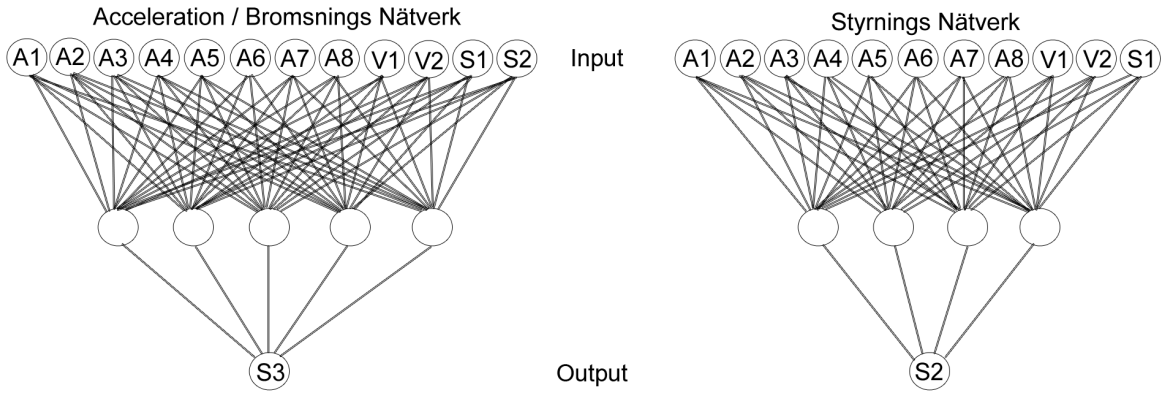
4. Diskussion
Innehåller en diskussion av resultaten från testerna och slutsatserna som drogs av dem. Innehåller också en reflektion om metoden som användes.

5. Framtida arbeten
Presenterar ett antal frågor som har kommit upp under arbetet och skulle kunna vara framtida arbeten.

2. Utförande

Vårt system består av två neurala nätverk för att köra bilen, det ena nätverket hanterade bilens acceleration och bromsning och det andra nätverket hanterade bilens styrning. Nätverkens utseende visas i Figur 6. Anledningen till att vi valde att använda fler än ett nätverk var för att dela upp problemet med att styra bilen i flera enklare subproblem som är lättare för nätverken att lära sig. Båda nätverken använder sig av bipolar sigmoid aktiveringsfunktionen som nämns i del 1.1.2.

Under träningen av nätverken så initieras vikterna med ett slumpat värde mellan -1 och 1.
För att få träningsdata så använde vi oss av en modifierad version av en spelar-bot som sparade ner information som nämns i Figur 6 och i listan nedanför tio gånger per sekund. Informationen används senare i en separat applikation som vi skapade för att utföra träningen av nätverket. Träningen fortsätter sedan tills vi når ett genomsnittligt felvärde som är lägre än ett gränsvärde vi bestämt i förhand, när träningen är klar sparas nätverkets vikter ner i en fil som sedan flyttas över till TORCS projektet som används av vårt neurala nätverk för att köra på banorna. Eftersom vi prövar oss fram med att lägga till fler perceptroner i det dolda lagret så sänks gränsvärdet tills felvärdet blir så lågt som möjligt. Sedan sätts gränsvärdet något högre än det för att förhindra överträning. Till exempel var gränsvärdet vid träningen av nätverket som hanterar acceleration/bromsning 0,17.



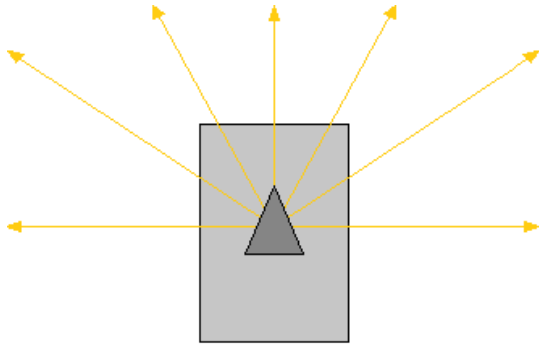
Figur 6: Bild på nätverken vi använde, det vänstra hanterar acceleration/bromsning och det högra hanterar styrning.

Nätverket som har hand om styrningen har 11 inputs.

1. Sju avståndssensorer, sensorernas placering visas i Figur 7.
2. Avstånd till mitten av segmentet på banan där bilen befinner sig i.
3. Vinkeln mot mittlinjen i det segmentet på banan som bilen befinner sig i.
4. Vinkeln mot mittlinjen i ett segment framför det som bilen befinner sig i.
5. Bilens nuvarande hastighet.

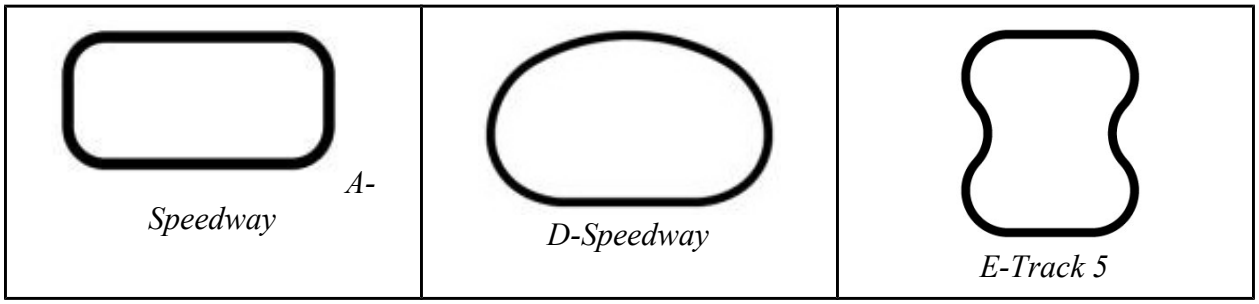
Nätverket som har hand om accelerationen och bromsningen har 12 inputs.

1. Sju avståndssensorer, sensorernas placering visas i Figur 7.
2. Avstånd till mitten av segmentet på banan där bilen befinner sig i.
3. Vinkeln mot mittlinjen i det segmentet på banan som bilen befinner sig i.
4. Vinkeln mot mittlinjen i ett segment framför det som bilen befinner sig i.
1. Bilens nuvarande hastighet.
2. Bilens styrning (resultatet från styrningsnätverket).



Figur 7: Avståndssensorernas riktning (orange linjer), den stora pilen visar vilket håll framåt är på bilen.

De neurala nätverken tränades med hjälp av träningsdata som spelats in när en spelare körde ett varv på banorna A-Speedway, D-Speedway och E-Track 5. Figur 8 visar hur de tre banorna ser ut.

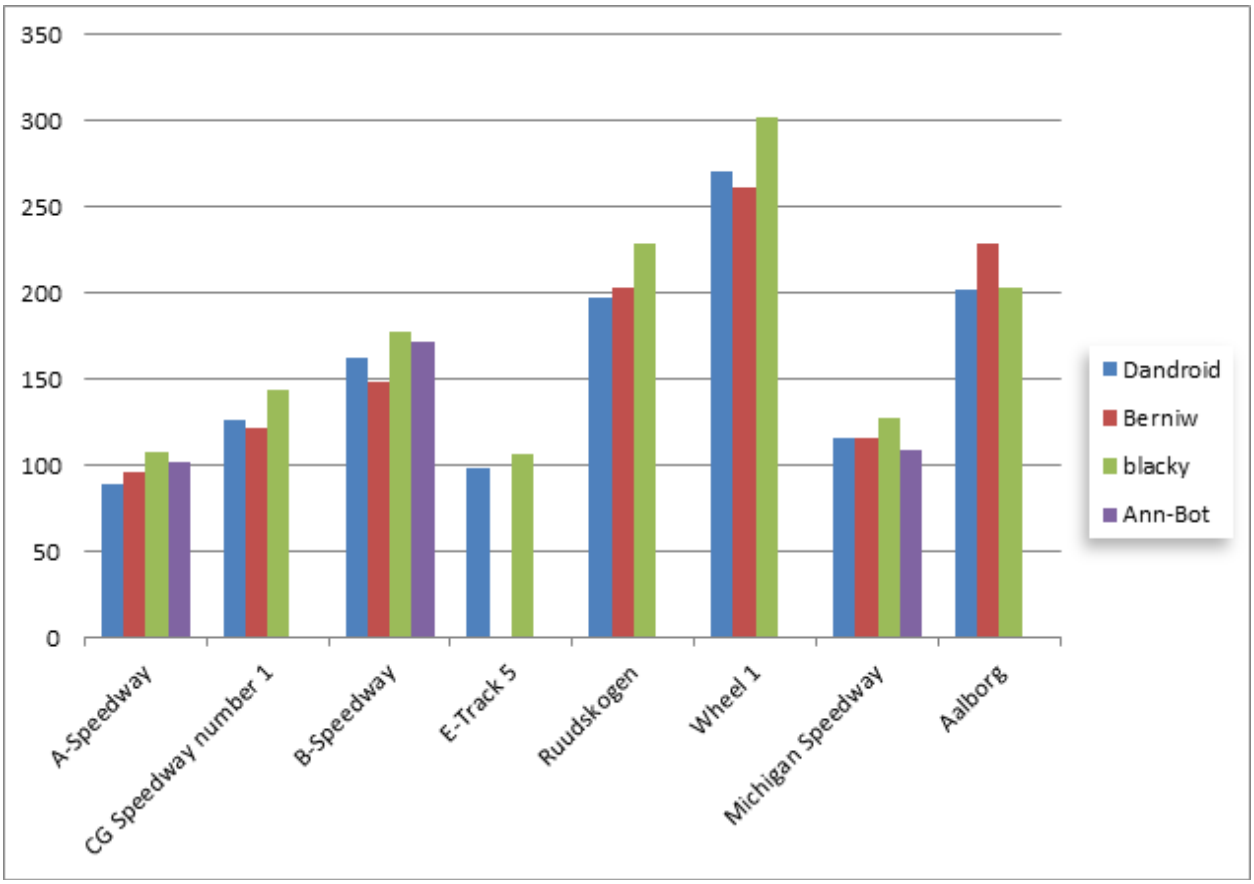


Figur 8: Banor som vi tränade nätverket på.

3. Resultat

De fyra AI-botarna testades på åtta olika banor i tre varv långa lopp. Av dem fyra banorna är ANN-boten tränad på att köra på två: A-Speedway och E-Track 5. AI-botarna testades fem gånger på varje bana, men det visade sig att resultatet var exakt samma på varje test. Därför presenteras inte medeltider och standardavvikelser.

Resultaten för AI-botarna presenteras i tabellerna 1 till 3. Bindestreck i tabellen visar att AI-boten inte klarade av banan, det vill säga att den fastnade i en minut, och K innebär att spelet kraschade.



Figur 9: Jämförelse av Total tid för varje AI-Bot per bana

Bana	**dandroid**	**berniw**	**blacky**	**ANN-bot**
A-Speedway	88.76s	96.67s	108.0s	101.68s
CG Speedway number 1	126.24s	121.44s	143.85s	-
B-Speedway	162.28s	148.91s	177.33s	172.3s
E-Track 5	98.12s	K	106.63s	-
Ruudskogen	197.30s	203.03s	228.48s	-
Wheel 1	270.80s	261.87s	302.36s	-
Michigan Speedway	115.86s	116.46s	127.36s	109.5s
Aalborg	202.38s	228.30s	203.63s	K

Tabell 1: Total tid för varje AI-bot

Bana	**dandroid**	**berniw**	**blacky**	**ANN-bot**
A-Speedway	28.07s	30.20s	34.49s	31.66s
CG Speedway number 1	41.05s	39.16s	47.03s	-
B-Speedway	52.02s	46.88s	57.27s	53.54s
E-Track 5	31.56s	K	34.28s	-
Ruudskogen	64.35s	66.22s	74.87s	-
Wheel 1	89.5s	85.63s	99.52s	-
Michigan Speedway	36.89s	36.73s	41.01s	33.43s
Aalborg	66.19s	74.83s	66.75s	K

Tabell 2: Tid på det bästa varvet

Bana	**dandroid**	**berniw**	**blacky**	**ANN-bot**
A-Speedway	262 km/h	262 km/h	248 km/h	249 km/h
CG Speedway number 1	243 km/h	247 km/h	231 km/h	-
B-Speedway	278 km/h	305 km/h	270 km/h	284 km/h
E-Track 5	223 km/h	K	203 km/h	-
Ruudskogen	259 km/h	242 km/h	232 km/h	-
Wheel 1	246 km/h	247 km/h	231 km/h	-
Michigan Speedway	270 km/h	284 km/h	252 km/h	284 km/h
Aalborg	231 km/h	225 km/h	230 km/h	K

Tabell 3: Högsta hastighet

4. Diskussion

Resultatet från testerna för ANN-boten är tydligt uppdelade i två kategorier. Vid testerna på A-Speedway, B-Speedway och Michigan Speedway var ANN-boten snabbare än den långsammaste state-of-the-art AI-boten på den banan: 7 sekunder snabbare på A-Speedway, 5 sekunder snabbare på B-Speedway och 18 sekunder snabbare på Michigan Speedway. På Michigan Speedway var ANN-boten den snabbaste AI-boten.

På andra banorna gick det betydligt sämre för ANN-boten, på första varvet på båda banorna fastande boten och kom inte loss, förutom på Aalborg där spelet kraschade. En av banorna som ANN-boten fastande på var en av banorna som hade använts som en källa för träningsdata: E-Track 5. Den stora skillnaden mellan banorna som ANN-boten klarade och inte klarade verkar vara att alla banorna som ANN-boten klarade var ovaler, medans dem som den inte klarade hade svängar åt både höger och vänster.

Det verkar som ANN-boten hade lärt sig att bara svänga åt vänster trots att den hade fått träningsdata från en bana som hade högersvängar, men eftersom vi tilllåter en del felmarginal från testerna så kan nätverket ha lärt sig fel vid högersvängar men kompenserat genom att bli övertränat på resten av banorna. Anledning att den inte lärde sig högersvängar skulle kunna bero på brist på input vilket gjorde att den bara kunde tränas att svänga åt ett håll eller så berodde det på att det behövdes en större mängd och mer varierande träningsdata.

Skillnaderna mellan ANN-boten och state-of-the-art AI-botarna berodde i det här testet väldigt mycket på banan som AI-botarna testades på. På banorna som ANN-boten klarade av var den snabbare än den långsamaste state-of-the-art-boten, men ANN-boten klarade inte heller av majoriteten av banorna som den testades på.

Testerna visar en del av problemen med att skapa ett generellt neuralt nätverk. Inom racingspel skulle detta problem kunna lösas genom att träna ett specifikt nätverk för varje bana eftersom själva träningen inte tar särskilt lång tid. Alternativt så skulle problemet kunna förenklas genom att använda en indirekt kontroller där det artificiella neurala nätverket hanterar ett högnivå problem, till exempel åt vilket håll bilen ska köra, medan vanlig regelbaserad AI hanterar hur bilen behöver gasa, bromsa eller svänga för att åka i den riktningen [6].

4.1 Reflektion

Det stora problemet med det här testet var att ANN-boten bara testades ordentligt på en bana för att kontrollera att det fungerade, istället borde det ha behövt klara alla träningsbanorna utan att krocka innan den fick gå vidare till testning vilket kunde ha resulterat i en ANN-bot som kunde klara fler banor.

4.2 Slutsats

Testerna visade att ANN-boten klarade att köra runt mindre än hälften av banorna som den testades på, men på banorna som den kunde klara såg körde den snabbare än den långsammaste state-of-the-art AI-boten och i ett fall var den snabbare än alla state-of-the-art AI-botarna. Anledning till att ANN-boten inte kunde klara av vissa banor berodde på att den inte kunde hantera högersvängar vilket i sin tur antagligen berodde på en brist på träningsdata.

5. Framtida arbete

ANN-boten som utvecklades i det här arbete hade problem med att köra på små och kurviga vägar, men var bra på att köra på breda vägar med lite kurvor. En idé hade kunnat vara att försöka skapa ett neuralt nätverk som kan hantera specifikt den typen av banor eller båda.

Det neurala nätverket i det här projektet var ett feedforward nätverk och tränades med hjälp av backpropagation, andra arkitekturer och inlärningsmetoder skulle kunna testas för att se om de ger något bättre resultat.

I testerna i det här arbetet var AI-botarna ensamma på banan, men i ett racing spel brukar det finnas flera bilar i ett lopp. Det hade varit intressant att utveckla ett neuralt nätverk som kan hantera att andra bilar som kör runt på banan. Detta hade krävt att det neurala nätverket kan hantera kollisioner och omkörningar, vilket kan vara komplicerat.

6. Referenser

[1] M. Buckland, *AI techniques for game programming*, Premier Press, 2002
[2] B. Schwab, *Game Engine Programming, second edition*, Charles River Media, 2009
[3] B. Karlik och A. Vehbi Olgac, Performance analysis of various activation functions in generalized MLP architectures of neural networks, *International Journal of Artificial Intelligence And Expert Systems*, 1 (4), s. 111-122, 2011
[4] Generation5.org, Intervju med Jeff Hannan, 2001, http://www.generation5.org/content/2001/hannan.asp (Hämtad 2013-05-23)
[5] M. Frean och A. Robins, Catastrophic forgetting in simple networks: an analysis of the pseudorehearsal solution, *Network: Computation in Neural Systems*, 10 (3), s. 227-236, 1999
[6] J. Mu˜noz, G. Gutierrez och A. Sanchis, A human-like TORCS controller for the Simulated Car Racing Championship, *CIG 2010. IEEE Symposium on Computational Intelligence and Games*, s. 473-480, 2010
[7] B. Chaperot och C. Fyfe, Motocross and artificial neural networks, *Game Design And Technology Workshop 2005*, 2005.
[8] B. Chaperot och C. Fyfe, Improving artificial intelligence in a motocross game, *CIG 2006. IEEE Symposium on Computational Intelligence and Games,* s. 181-186, 2006
[9] TORCS, http://torcs.sourceforge.net/ (Hämtad 2013-05-02)
[10] L. Cardamone, D. Loiacono och P. Luca Lanzi, Learning Drivers for TORCS through Imitation Using Supervised Methods, *CIG 2009. IEEE Symposium on Computational Intelligence and Games*, s. 148-155, 2009
[11] TORCS Racing Board, http://www.berniw.org/trb/, (Hämtad 2013-05-27)