Tinlab Machine Learning Groepsverslag

Thomas Alakopsa
0911723
Alex de Ridder
0937558

October 30, 2019

1 Samenvatting

Voor Tinlab Machine Learning moest een controller worden gemaakt voor het computer programma Torcs. Om dit te realiseren is onderzoek gedaan in Machine Learning principes. Na dit onderzoek is er een neuraal netwerk gemaakt in Java met de behulp van de library Encog. Dit netwerk kan een auto in Torcs op elk parcours van start tot finish brengen zonder dat deze crashed. Zelfs met de "Noicy sensoren" is dit nog niet voorgekomen.

2 Inleiding

Voor Tinlab Machine Learning wordt de verworven kennis over Machine Learning toegepast door een intelligente controller te maken voor race simulatie Torcs. In plaats van zelf aan de knoppen te zitten en de auto te besturen, zal er een programma geschreven worden die aan de hand van getrainde modellen en binnenkomende data zelfstandig de auto bestuurd.

3 Projectopzet

Er wordt tijdens dit project gebruik gemaakt van een versie van Agile. Het werk wordt verdeeld in kleine opdrachten die afgemaakt moeten worden in een periode van twee weken. Voordat de kleine opdrachten worden gemaakt, is er een algemene planning gemaakt. De eerste draft is in section 3.1 te vinden. De uiteindelijke gerealiseerde planning staat in bijlage 8.4.

Om de code en verslagen op te slaan van dit project gebruiken wij Github, in bijlage 8.7 is de log te vinden van de commits. Alle verslagen worden lokaal geschreven in Latex en als de taak klaar is gepusht op Github.

3.1 Algemene planning

Taak	**Planning(week)**
Requirements	1
Testplan	2
Persoonlijk verslag	6
Onderzoek controller	4
Eerste versie controller	6
Testen en verbeteren	7
Groepsverslag controller	7
Reflectieverslag	8

Product backlog

Prioriteit	User story
Must	Als projectlid, neem ik deel aan de boeken-club gehost bij klasgenoten, zodat ikmeer in-formatie over Machine Learning verkrijg
Must	Als projectlid, host ik een boekenclub voor klasgenoten over een thema binnen de AI, zodat ik meer in-formatie over Machine Learning leer en uitdeel
Must	Als project lid, maak ik een persoonlijk verslag, over de wekelijkse geleerde stof en zelfopgedane kennis, zo-dat ik meer informatie over Machine Learning verkijg.
Must	Als projectlid, schrijf ik een ethische verantwoord-ing, zodat er een ethisch product wordt geleverd.
Must	Als projectlid, wil ik een controller trainen, die een auto rijdt in het computerprogramma torcs, zodat het vak behaald kan worden
Should have	Als project lid, train ik een netwerk dat geen schade rijdt, omdat een controller die niet crashed goed getrained is.
Should have	Als project lid, wil ik veel verschillende soorten netwerken trainen, zodat we veel netwerken met elkaar kunnen vergelijken en een conclusie kunnen trekken over welke eigenschappen een postieve of negatieve invloed hebben.

Eisen controller

- Het neurale netwerk moet de auto snel de baan laten afleggen (50 km/u gaan over heel de baan is dus niet goed.)
- Getraind worden door een van de AI technieken
- Neurale network is getrained met de gegeven train data
- Voorafgesteld doel halen (Start - Finish)
- Er moet zo weinig mogelijke schade aan de auto zitten

4 Vooronderzoek

4.1 Neuraal netwerk

Neurale netwerken zijn een reeks algoritmen die losjes gemodelleerd zijn van het menselijke brein. Een kunstmatig brein dat gemaakt is uit een hele grote reeks kunstmatige neuronen.

4.1.1 Perceptrons

Een van de meest fundamenteele kunstmatig neuron types is een perceptron. [3] Perceptronen zijn een belangrijk onderdeel van een neuraal netwerk en kennis hierover is nodig om een neuraal netwerk te begrijpen. Een perceptron pakt verschillende binary inputs:$x_1, x_2, ...x_n$ en produceerd een enkele binaire output. Je kan het zien als een functie die beslissingen voor je neemt, door verschillende factoren tegen elkaar te wegen en uiteindelijk met ja of nee te antwoorden.

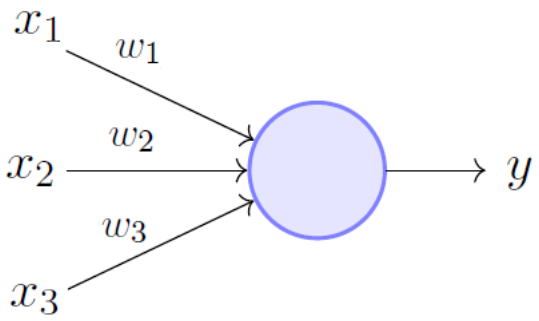

Figure 1: een enkelen perceptron

In afbeelding 1 is een perceptron te zien die 3 variabelen als input neemt: x_1, x_2 en x_3. Bij all deze waardes word een gewicht(weight) toegekend(w_n). Deze waarde geeft aan hoe belangrijk de input is voor deze neuron. De output van de neuron is de som van alle resultaten bij elkaar. $\sum_j w_j x_j$ en deze waarde vergelijken met een gekozen randwaarde(threshold) om de output the berekenen. In een meer wiskundige term (w = weight, x = input en b = bias):

$$\text{output perceptron} = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j + b > \text{threshold} \end{cases}$$

Je kan de output van een neuron beinvloeden door te spelen met de weights en thresholds. Door een input zijn weight te vergroten of de threshold te verlagen kan er hele andere resultaten uit het model komen [4].

Het is duidelijk dat de perceptron niet een compleet model is over hoe mensen hun beslissingen nemen. Maar het voorbeeld illustreert hoe een perceptron verschillende soorten bewijs kan afwegen om beslissingen te nemen. Daarom is het aannemelijk dat een complex netwerk van perceptrons vrij subtiele beslissingen zou moeten kunnen nemen [4].

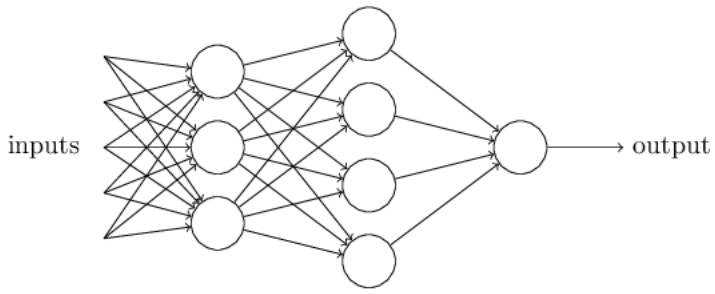

Figure 2: een neural network van meerderen perceptrons

In afbeelding 2 is een netwerk te zien, waar de eerste laag van perceptrons in het netwerk, drie simpele beslissingen neemt door de functie in vergelijking **??** uit te voeren. Naast de eerste laag zit er nu ook een tweede die de outputs van de eerste laag als input neemt. Op deze manier kan een perceptron in de tweede laag een beslissing nemen op een complexer en abstracter niveau dan perceptrons in de eerste laag. Deze complexiteit en abstractheid word verhoogd per extra laag dat je toevoegd. Op deze manier kan een meerlaags netwerk van perceptrons, zeer geavanceerde beslissingen nemen [4] [5].

De volgende stap is om ons netwerk zelf lerend te maken. Om dit te doen moet je kleine aanpassingen kunnen maken aand de weights en de biases. Deze kleine aanpassingen moet daarna ook een klein effect hebben op de output van het neurale netwerk. Echter dat is niet wat er gebeurt met perceptronen want deze heeft maar 2 outputs, een 1 en een 0. Een kleine aanpassing zal daarom niks doen of de hele uitkomst van de perceptron omdraaien. Je kan niet probleem omzeilen door een ander types neurons te gebruiken ,zoals de Sigmoid en tanh neurons. [5]

4.1.2 Activatie functies

De resultaten van een neuron worden berekend door een activatie functioes. Voor dit onderzoek worden eerst alleen de "Sigmoid" en "Tanh" activatie functies gebruikt, mochten deze niet goed werken worden andere activatie functies verder onderzocht [5].

De sigmoid en tanh neurons lijken erg op perceptrons, alleen de manier hoe de ouput berekend wordt is compleet anders. Een sigmoid function neemt alle mogelijke nummers als een input en berekent het naar een getal tussen de 0 en 1.

$$i_\theta(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Een tanh functie neemt alle mogelijke nummers als input en brekend het naar en getal tussen de -1 en de 0.

$$j_\theta(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2}$$

De twee berekeningen lijken erg op elkaar, omdat tanh een geschaalde vorm is van de sigmoid functions.

Omdat de data die gebruikt word voor het sturen van de torcs auto tussen de -1 en 1 zit is het verstandig om gebruik te maken van de tanh functie. Omdat tanh alle reel getal tussen de -1 en 1 kan hebben, dus waarden zoals -0.875 en 0.132. Dit zorgt ervoor dat de uitkomst van een neuron niet een simple een ja of nee is maar veel complexer getal dat van alles en nog wat kan betekenen. Nu als er een kleine verandering plaatsvind in de weights of biases van het neurale netwerk zullen deze kleine verandering ook te vinden worden in de output, wat uiteindelijk zorgt dat het neurale netwerk zelf kan leren.

4.1.3 Zelf lerend netwerk

Er is data nodig om een neuraal netwerk te trainen. Nadat deze data verzameld is kan er gebruik gemaakt worden van een methode die het neurale netwerk traint. Voorbeeld van zulke methodes zijn forward propagation, back propagation en resilent propagation. Het doel van deze algoritmes is om alle neuronen een weight en een bias te geven. Hoe dat in praktijk gaat hangt af van de dataset en welk algoritmn er gebruikt gaat worden, kort samengevat gaat het op deze manier [5]:

- De start waarden voor de weights en biases zijn vaak random.
- De dataset wordt meegegeven aan het network en daar wordt data/resultaten uit gegenereerd.

- Deze resultaten worden vergeleken met de verwachting en het foutpercentage/error wordt berekend.
- Deze informatie wordt gebruikt om het het netwerk te tunen, zodat de error verminderd wordt.
- De stappen worden herhaald tot het netwerk aan de eisen voldoet.

4.1.4 Overfitting

Neurale netwerken zijn erg kwetsbaar voor overfitting. Dit is als een neural netwerk alleen kan omgaan met de data uit een specifieke dataset en niet goed reageert op toekomstige waarnemingen. De informatie in deze sectie komt uit het boek "Algoritms to live by" [1]

Overfitting ontstaat door het opnemen van een overvloed aan factoren in het neurale netwerk. Dit kan door heel veel specifieke data te geven aan het netwerk of door heel veel neuronen te gebruiken, die connecties gaan maken met de gegeven data. Wanneer dit gebeurt, zal het netwerk goed passend zijn voor de bestaande gegevens/input, maar een betere pasvorm voor de trainiingsset, hoeft niet te resulteren in een betere prestatie.

Voor een netwerk met te weinig train data en/of neurons(Underfitting), als het neurale netwerk te eenvoudig wordt gemaakt, kan deze het essentiële patroon in de data niet vastleggen.

Het is voor het trainen van een neuraal netwerk moeilijk te verspellen of er overfitting zal plaatsvinden, alleen door goed na te denken over welke informatie je het netwerk zal laten geven, hoelang je het zal trainen en hoeveel neurons en layers het netwerk zal krijgen kan je zorgen dat je geen last krijgt van overfitting. De netwerken moet altijd getest worden of ze in de buurt komen van de resultaten die verwacht worden, of dat ze teveel zijn getraind op de trainingsset.

5 Methode

Aan het begin van het project zijn meerdere datasets gemaakt door de input en output te loggen van een goed rijdend systeem. Deze datasets zullen gebruikt worden om het neurale netwerk te trainen en/of testen.

Voor communicatie met de "Torcs server" kan er gebruik gemaakt worden van een client in Java of C++. De Java client heeft de voorkeur, met als voornaamste reden dat er op github [2] een opzet te vinden is om een Neural Netwerk te schrijven in de taal Java. Deze versie maakt gebruik van een algoritme, dat niet door Machine Learning is gemaakt.

Het eindproduct zal een neuraal netwerk zijn dat getraind is door middel van een dataset en verifiërd door andere datasets. Aanpassingen in de instellingen van het neurale netwerk zullen uiteindelijk leiden tot de beste controller. Het neurale netwerk zal tijdens het trainen na een X aantal iteraties opgeslagen worden en bij vroegtijdig stopzetten kan de training ook weer hervat worden, vanaf de laatst opgeslagen iteratie.

Het getrainde neurale netwerk kan geverifiërd worden op twee manieren, namelijk met de testdatasets of door het neurale netwerk te gebruiken in Torcs. Resulaten van de test met de datasets is binnen enkele secondes uitgerekend en kan het eerste onderscheid geven. Bij het gebruiken van Torcs is duidelijk te zien hoe het neurale netwerk reageert op de omgeving en of en hoe het vastloopt. In de resultaten zal dan het verband uitgelegd worden tussen de instellingen van het neurale netwerk en resultaten. Meer informatie hierover is te vinden in het testplan, wat wij ook zullen uitvoeren, zie bijlage 8.2.

Java is gekozen als taal om het neurale netwerk te trainen. De beste manier voor een neuraal netwerk te programeren is door gebruik te maken van bestaande libaries. Na wat googlen kwamen we op twee opties, de java library Encog gebruiken [**?**] of een andere taal gebruiken en deze koppelen aan de client van Elvira door een UDP verbinding. De laatste optie zal het meeste werk kosten, dus er is gekozen om eerst te kijken wat Encog voor Neuraal Netwerk opleverd.

Trainen en creeëren van het netwerk moet gemakkelijk kunnen op een server, zodat beide projectgenoten bij de training kunnen komen. Dit kan door middel van een jar file, deze jar file en de datasets zijn als enige nodig om een neuraal netwerk te genereren en trainen.

6 Resultaten

Tijdens het uitvoeren van de methodes van het testplan 8.2 werd al snel duidelijk dat één methode niet het verwachte resultaat leverde. De error, die wordt berekend met de datasets, heeft geen verband met de prestatie in Torcs. Het is belangrijk dat het neurale netwerk in Torcs goed functioneerd. Daarom is besloten om deze methode niet verder te gebruiken, deze optie is echter nog in de source code te vinden.

In de testresulaten 8.3 is goed het verschil te zien tussen de netwerken. Tijdens het testen hebben we een stuckmethode en remmethode toegevoegd. In bepaalde situaties nemen deze methodes/algorithme het neurale netwerk over. Deze algoritmes zijn nodig om de auto van de baan te krijgen na een crash of sneller te laten opstarten. Bij elk neurale netwerk en race remde het autonome netwerk bij het begin van de race, dit is natuurlijk niet nodig. In bijlage 8.6 en 8.6 is de uiteindelijke implementatie te vinden van deze methodes.

De meeste netwerken trainen niet verder binnen de range van 3000 en 5000 trainiteraties. Voor dit aantal iteraties heeft het netwerk niet goed genoeg geleerd, wat resulteert in slechte resultaten. Na de 5000 iretaties gaat de presatie weer omlaag, waarschijnlijk door overfitting.

Er is alleen gebruik gemaakt van de activatiemethode Tanh, omdat de resulaten goed genoeg zijn. Verbetering kan eerder plaatsvinden in de stuck- en remmethodes of aanpassing van de datasets. Dit is ook de rede, waarom er ook alleen gekozen is voor een Encog implementatie. Een kleine verbetering, wat door het gebruik van andere libraries of talen gerealiseerd kan worden, weegt niet op tegen de tijd en werk die erin gestoken moet worden. Daarom hebben we gekozen om een netwerk te kiezen en de stuck en -remmethode verder te ontwikkelen. Uit de resultaten bleek dat een aantal netwerken uitstoken boven de rest:

- 2 laags netwerk: eerste laag van 100 neurons, tweede laag van 40 neurons
- 3 laags netwerk: eerste laag van 100 neurons, tweede laag van 60 en een derde laag van 40 neurons.
- 2 laags netwerk: eerste laag van 200 neurons, tweede laag van 40 neurons.

Deze netwerken zijn verder getest op alle andere banen in Torcs, in de Gui mode, zodat duidelijk werd wat het verschil was tussen de netwerken (details). Het netwerk met 3 lagen, (100, 60, 40), was het snelst en meest betrouwbaar. Hierna is de stuck en rem methode perfect ingesteld voor dit netwerk.

7 Conclusie

Een controller voor Torcs is gemaakt, na onderzoek is gekozen om een neuraal netwerk te trainen in Java met de library Encog. Het neurale netwerk is getraind met de combined_dataset.csv aangeboden door de vakdocent. Na het trainen en testen van verschillende neurale netwerken is uiteindelijk gekozen om een 3 laags neuraal netwerk te gebruiken dat is opgemaakt uit lagen van 100, 60 en 40 neuronen. Voor alle lagen wordt gebruik gemaakt van de activatiefunctie Tanh. Wanneer het neurale netwerk vastloopt, zal een algoritme het neurale netwerk tijdelijk overnemen.

8 Bijlages

8.1 Architectuurontwerpen

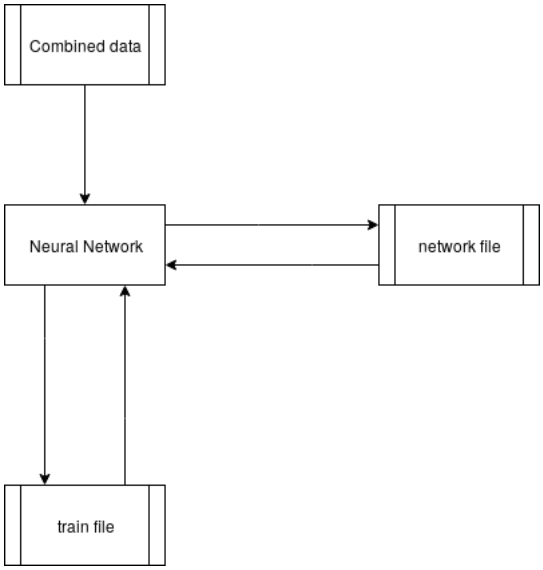

Figure 3: Training uitvoeren, resulteert in network en train file

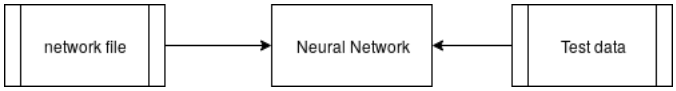

Figure 4: Testen van getrainde netwerk, resulteert in error waardes

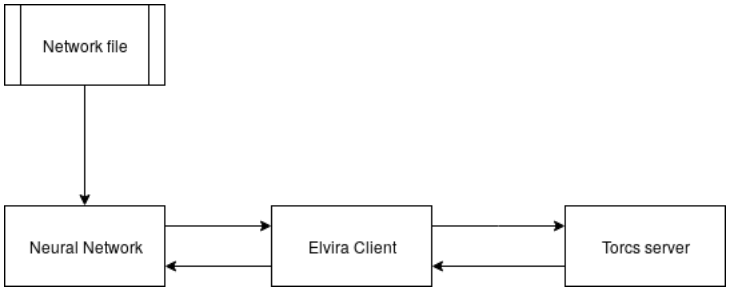

Figure 5: Testen van getrainde netwerk, resulteert in Torcs resultaten

8.2 Testplan

Er zijn heel verschillende mogelijkheden voor het maken en testen van neurale netwerk. Tijdens het testen is het is moeilijk om te zien wat een positieve of negatieve invloed heeft. Daarom wordt elk getrainde neurale netwerk vergeleken, om te kijken welke eigenschappen samen het beste resultaat geeft. Om te komen tot het beste resultaat hebben we de volgende manieren om te testen: kijken naar de optimale errorwaarde, netwerk moet zo min mogelijk crashen, gemakkelijk herstellen na een crash en een parcours als snelste voltooien. Deze opties kunnen veder uitgewerkt worden tijdens het testen.

Voor elk parcours is een data beschikbaar van een goed presterende controller in Torcs. De errorwaarde voor het getrainde netwerk en de testdata van een parcours kan binnen millisecondes berekent worden. Met de errorwaardes kan je gemakkelijk zien of het neurale netwerk goed getraind is en met deze data is het eerste verschil tussen het neurale netwerk te zien.

Het neurale netwerk kan ook getest worden in torcs, torcs heeft twee verschillende manieren om te runnen: text mode of gui mode. De text mode geeft beste laptijd, hoogste snelheid, schade en de totale race. In de textmode zijn alleen resultaten te zien, het voordeel is dat hiervoor weinig tijd nodig is van de tester en deze data veel meer zegt dat errorwaardes. Deze test wordt uitgevoerd waar het neurale netwerk wordt beoordeeld op de volgende criteria:

- De tijd waarop de auto 1 ronde rijdt op de geselecteerde baan
- Het aantal schaden dat de auto heeft aan het einde van 1 ronde
- De maximum snelheid die een auto behaald tijdens 1 ronde

De gui mode wordt gebruikt om te zien hoe het neurale netwerk de auto crasht en hoe die de auto terugzet op de weg. Door de schaden uit de vorige test kan er al worden geconstateert dat een netwerk gecrashed is. Elke crashsituatie is uniek , dus het is niet zeker of het neurale netwerk zichzelf terug kan plaatsen op de weg. Het beste neurale netwerk zal nooit crashen, maar om zeker te weten of het zichzelf uit een crash kan halen, moet er een crashsituatie gecreëerd worden. Er kan voor gekozen worden om de simulatie gelijk te laten crashen en kijken hoe het neurale netwerk dit oppakt. Of om te racen op een dirt baan, hier zal de auto snel tegen een muur rijden, echter is het niet zeker of de unstuck methode hetzelfde werkt op de dirt baan dan op een asfalt baan.

De datasets zijn gemaakt op asfalt, daarom is het belangrijk dat er ook getest wordt op asfalt. Ook zijn bochten, rechte stukken en hellingen van belang in het testparcours zodat je test op zoveel mogelijk variatie in situaties. Er gaat getest worden op de volgende parcous(tracks).

- Aalborg
- Alpine 1

- Alpine 2

Gedurende het testen wordt er gekeken welke testmethodes het beste werken en daar zullen we dan ook uiteindelijk mee gaan testen. In het kopje resultaten zal dit verder worden uitgelegd.

8.3 Testresultaten

Samenvatting

Network	Aalborg	Alpine 1	Alpine 2
80_65_50_35 V1	DNF	1	1
100_60_40 V2	1	3	6
100_60_40 V1	2	4	4
20_18_16	4/5	7	8
100_40_V1	DNF	9	5
100_40_V2	DNF	-	3
100_40_V3	9	13	9
200_40	8	6	17
100_60_40 V3	6	10	7
200_100_40 V1	DNF	5	13
200_100_40 V2	10	8	17
200_100_40 V3	11	10	
44_33_33_22 V1	DNF	14	12
44_33_33_22 V2	7	11	11
44_33_33_22 V3	3	12	17
80_65_50_35 V2	DNF	15	2
80_65_50_35 V3	DNF	2	17
30_24_18	DNF	-	-

Data per race

Aalborg

Place	Network	Time	Damage	Topspeed	Stuck method
1	100_60_40 V2	1:38:80	0	174	Version 2
2	100_60_40 V1	1:47:41	0	176	Version 2
3	44_33_33_22 V3	1:48:55	5	171	Version 2
4	20_18_16	1:48:94	441	172	Version 1
5	20_18_16	1:48:94	441	172	Version 2
6	100_60_40 V3	1:54:93	197	175	Version 2
7	44_33_33_22 V2	1:56:06	935	173	Version 2
8	200_40	1:57:61	569	167	Version 2
9	100_40_V3	2:01:62	142	175	Version 2
10	200_100_40 V2	2:21:22	1207	176	Version 2
11	200_100_40 V3	4:03:88	1605	179	version 2
18	200_100_40 V1	DNF	-	-	Version 2
18	44_33_33_22 V1	DNF	-	-	Version 2
18	80_65_50_35 V1	DNF	-	-	Version 2
18	80_65_50_35 V2	DNF	-	-	Version 2
18	30_24_18	DNF	-	-	Version 2
18	100_40_V1	DNF	-	-	Version 2
18	100_40_V2	DNF	-	-	Version 2

Alpine 1

Place	Network	Time	Damage	Topspeed	Stuck method
1	80_65_50_35 V1	2:57:88	0	211	Version 2
2	80_65_50_35 V3	2:58:38	0	211	Version 2
3	100_60_40 V2 1	3:00:20	0	209	Version 2
4	100_60_40 V1	3:00:23	0	211	Version 2
5	200_100_40 V1	3:07:46	43	201	Version 2
6	200_40	3:08:63	551	207	Version 2
7	20_18_16	3:08:78	1461	204	Version 2
8	200_100_40 V2	3:13:24	178	204	Version 2
9	100_40_V1	3:14:56	540	191	Version 2
10	100_60_40 V3	3:17:41	657	209	Version 2
11	44_33_33_22 V2	3:23:78	253	211	Version 2
12	44_33_33_22 V3	3:30:99	979	211	Version 2
13	100_40_V2	3:33:50	1952	195	Version 2
14	44_33_33_22 V1	3:35:77	1202	208	Version 2
15	80_65_50_35 V2	3:38:81	1608	207	Version 2
16	200_100_40 V3	3:57:84	225	206	Version 2
17	100_40_V3	6:09:33	7456	197	Version 2

Alpine 2

Place	Network	Time	Damage	Topspeed	Stuck method
1	80_65_50_35 V1	2:06:37	355	188	Version 2
2	80_65_50_35 V2	2:10:66	307	187	Version 2
3	100_40_V2	2:11:41	1837	192	Version 2
4	100_60_40 V1	2:12:63	37	191	Version 2
5	100_40_V1	2:14:44	238	188	Version 2
6	100_60_40 V2	2:15:54	223	190	Version 2
7	100_60_40 V3	2:16:60	63	190	Version 2
8	20_18_16	2:20:88	601	192	Version 2
9	100_40_V3	2:25:62	417	190	Version 2
10	200_100_40 V3	2:27:14	0	186	Version 2
11	44_33_33_22 V2	2:28:82	425	190	Version 2
12	44_33_33_22 V1	2:43:74	544	191	Version 2
13	200_100_40 V1	2:35:29	2158	181	Version 2
17	200_40	DNF	-	-	Version 2
17	200_100_40 V2	DNF	-	-	Version 2
17	44_33_33_22 V3	DNF	-	-	Version 2
17	80_65_50_35 V2	DNF	-	-	Version 2

Rauwe resulaten

Network	Map	Time	Damage	Topspeed	Stuck method
20_18_16	Aalborg	1:48:94	441	172	Version 1
20_18_16	Alpine 1	3:08:78	1461	204	Version 2
20_18_16	Alpine 2	2:20:88	601	192	Version 2
20_18_16	Aalborg	1:48:94	441	172	Version 2
30_24_18	Aalborg	DNF	-	-	Version 2
100_40_V1	Aalborg	DNF	-	-	Version 2
100_40_V1	Alpine 1	3:14:56	540	191	Version 2
100_40_V1	Alpine 2	2:14:44	238	188	Version 2
100_40_V2	Aalborg	DNF	-	-	Version 2
100_40_V2	Alpine 1	3:33:50	1952	195	Version 2
100_40_V2	Alpine 2	2:11:41	1837	192	Version 2
100_40_V3	Aalborg	2:01:62	142	175	Version 2
100_40_V3	Alpine 1	6:09:33	7456	197	Version 2
100_40_V3	Alpine 2	2:25:62	417	190	Version 2
200_40	Aalborg	1:57:61	569	167	Version 2
200_40	Alpine 1	3:08:63	551	207	Version 2
200_40	Alpine 2	DNF	-	-	Version 2
100_60_40 V1	Aalborg	1:47:41	0	176	Version 2
100_60_40 V1	Alpine 1	3:00:23	0	211	Version 2
100_60_40 V1	Alpine 2	02:12:63	37	191	Version 2
100_60_40 V2	Aalborg	1:38:80	0	174	Version 2
100_60_40 V2	Alpine 1	3:00:20	0	209	Version 2
100_60_40 V2	Alpine 2	2:15:54	223	190	Version 2
100_60_40 V3	Aalborg	1:54:93	197	175	Version 2
100_60_40 V3	Alpine 1	3:17:41	657	209	Version 2
100_60_40 V3	Alpine 2	2:16:60	63	190	Version 2
200_100_40 V1	Aalborg	DNF	-	-	Version 2
200_100_40 V1	Alpine 1	3:07:46	43	201	Version 2
200_100_40 V1	Alpine 2	2:35:29	2158	181	Version 2
200_100_40 V2	Aalborg	2:21:22	1207	176	Version 2
200_100_40 V2	Alpine 1	3:13:24	178	204	Version 2
200_100_40 V2	Alpine 2	DNF	-	-	Version 2
200_100_40 V3	Aalborg	4:03:88	1605	179	version 2
200_100_40 V3	Alpine 1	3:57:84	225	206	Version 2
200_100_40 V3	Alpine 2	2:27:14	0	186	Version 2
44_33_33_22 V1	Aalborg	DNF	-	-	Version 2
44_33_33_22 V1	Alpine 1	3:35:77	1202	208	Version 2
44_33_33_22 V1	Alpine 2	2:43:74	544	191	Version 2

44_33_33_22 V2	Aalborg	1:56:06	935	173	Version 2
44_33_33_22 V2	Alpine 1	3:23:78	253	211	Version 2
44_33_33_22 V2	Alpine 2	2:28:82	425	190	Version 2
44_33_33_22 V3	Aalborg	1:48:55	5	171	Version 2
44_33_33_22 V3	Alpine 1	3:30:99	979	211	Version 2
44_33_33_22 V3	Alpine 2	DNF	-	-	Version 2
80_65_50_35 V1	Aalborg	DNF	-	-	Version 2
80_65_50_35 V1	Alpine 1	2:57:88	0	211	Version 2
80_65_50_35 V1	Alpine 2	2:06:37	355	188	Version 2
80_65_50_35 V2	Aalborg	DNF	-	-	Version 2
80_65_50_35 V2	Alpine 1	3:38:81	1608	207	Version 2
80_65_50_35 V2	Alpine 2	DNF	-	-	Version 2
80_65_50_35 V3	Aalborg	DNF	-	-	Version 2
80_65_50_35 V3	Alpine 1	2:58:38	0	211	Version 2
80_65_50_35 V3	Alpine 2	2:10:66	307	187	Version 2

8.3.1 Stuck versies

Version	**U_T_L**	**M_U_A**	**M_U_S**	**M_U_D**	**M_U_D**
1.0	2.0	30/(180*pi)	5.0	0.9	0.2
2.0	2.0	30/(180*pi)	5.0	0.9	0.3

- U_T_L = UNSTUCK_TIME_LIMIT
- M_U_A= MAX_UNSTUCK_ANGLE
- M_U_S = MAX_UNSTUCK_SPEED
- M_U_D = MIN_UNSTUCK_DIST
- M_U_D = MAX_UNSTUCK_DIST

8.4 Planning

Taak	**Planning**	**Opgeverd**
Requirements	1	5
Testplan	2	6
Persoonlijk verslag	6	8
Onderzoek controller	4	5 (*np*)
Eerste versie controller	6	8
Testen en verbeteren	7	7 (np)
Tweede versie controller	6	7 (*np*)
Groepsverslag controller	7	7 (*np*)
Reflectieverslag	8	8 (*np*)

Table 1: Uiteindelijke gerealiseerde planning in weken. *np = nieuwe periode*

8.5 Handleiding gebruiken jar file

Standaard benodigdheden

- jar file
- 'train' en 'network' map
- 'torcs.properties' file

Argumenten

-createTraining

Hiervoor is de folder train nodig waarin 'combined_data.csv' of andere trainingset staat. Er worden dan een network file en eg file gegenereerd na 1000 iteraties.

-resumeTraining

Hiervoor is de folder train nodig waarin 'combined_data.csv' of andere trainingset staat. Daarnaast zijn deeen network file en eg file nodig om de training weer te hervatten.

-controlNetwork

Hiervoor is de folder train_data nodig waar alle testdatasets in staan, deze worden allemaal getest op de network file in de 'network' folder.

-run

Gebruken van neurale netwerk in Torcs, hiervoor moet de 'torcs.properties' file ingevuld worden en een network file staan in de 'network' map. Na het uitvoeren van de jar en command kan de naam van het network worden ingevuld.

8.6 Code

Stuck methode

```
@Override
    public void process(Action action, SensorModel sensors) {

        if (isStuck(sensors)) {
            System.out.println("Ik ben stuck");
            action.steering = -sensors.getAngleToTrackAxis() /(0.366519 *
                2);
            action.gear = -1; // reverse gear
            action.accelerate = 0.5D; // 50% accelerator pedal
            action.brake = 0.0D; // no brakes
        }
        return;
    }

    public boolean isStuck(SensorModel sensors) {

        if (Math.abs(sensors.getAngleToTrackAxis()) > MAX_UNSTUCK_ANGLE
            &&
                sensors.getSpeed() < MAX_UNSTUCK_SPEED &&
                Math.abs(sensors.getTrackPosition()) > MAX_UNSTUCK_DIST) {
            if (stuck > MAX_UNSTUCK_COUNT && sensors.getTrackPosition() *
                sensors.getAngleToTrackAxis() < 0.0) {
                return true;
            } else {

                stuck++;
                return false;
            }
        } else {
            stuck = 0;
            return false;
        }
    }
```

Remmen begin race

```
//Probleem dat Neurale Netwerk moeilijk start vanaf het begin van de race
       if (sensors.getSpeed() <= 50){
           stuck++;
           //Controle of de auto stil staat
           if (stuck <= 100){
               //Auto heeft moeite met wegrijden, override de brake van
                   het neurale netwerk
               action.brake = 0;
               return;
           }
       }
       if (stuck > 100){
           action.brake = 0;
           //Auto rijdt weer, brake van neurale netwerk gebruiken
           if(sensors.getSpeed() > 60){
               stuck = 0;
           }

       }
```

8.7 Commitlog

```
* 88810a0 finished perspectron in vooronderzoek
* a560bed worked on perceptron explanation
* e1d4ef3 Add AdjustBreaks method, add validation sets
* 18492f9 Change break algoritmn
* 23d92b0 Add filtering scorebord
* e72aeff Add option to give file name
* 4a8bab8 finished documenting all the training results
* f98924e added all the new networks
* ad5b9cf Methode eerste versie
* 40a5a30 Add verify method
* aae00ab Break can't be less then 0
* cdacea0 Fix resume method
* 87a6cf2 Groepsverslag opzet opgezet
* f4f1786 Add 100_40
* 820c658 added fucntion that asks for name of the file or press enter to use default
* 54dd446 added 80_65_50_35 version 1, 2 and 3 and 44_33_33_22 version 1,2 and 3. all with c
* 2d21c56 addes the result of network 200_100_40 and 100_60_40
* 07ad208 Add new score
* 61f81b0 updated version of 200_100_40 and 100_60_40
*   54e3c40 Merge branch 'master' of https://github.com/ThomasAlakopsa/machine_learning
|\
| * ae46afe Generate scoreboard pdf latex
| * 321361d Run new tests
* | c850536 added my networks
|/
* 9247e8d Nieuwe trainde networks
* 7468141 add scoreboard and remove print
* 6fba598 add scoreboard and remove print
* e84e80c Change stuck method & add new training
* feb9b08 added the best network we have for now
* 2f0c7ed Add extra layer & stuck method
* f23e4e0 Add right train_data
* 567d067 Test
* 6117f60 Add neuralnetworks
* 5fb6117 Add runnable
* 8bb5459 Added Neural Network with Encog
* b6b5ec9 removed not needed files
* a7054b0 added .bbl .gz and .blg files in the git ignore
* 1bf3bd6 updated .gitignore and added folders for personal reports and a template for the f
* 991ddb0 setting up work station / code is running
* b741a3f added the code from Elvira
* 0a4409a first commit
```

References

[1] Brian Christian and Tom Griffiths. *Algoritms to live by*. Willem Collins, 2016.

[2] ElviravdVen. Opzet client. `https://github.com/ElviravdVen/Torcs_CI`. [Online; gedownload op 18-juni-2019].

[3] Jean-Christophe B .Loiseau. Rosenblatts perceptron, the first modern neural network, maart 2011.

[4] Michael Nielsen. Using neural nets to recognize handwritten digits, Jun 2019.

[5] Jordi TORRES.ai. learning process of a neural network, 2018.