Als er meer equivarianties worden verwerkt in neurale netwerken die met *reinforcement learning* worden getraind, worden ze dan beter in het spelen van Snake?

Auteur: Jesse Maas
Begeleider: Thijs van Ommen
Tweede beoordelaar: Gerard Vreeswijk
Scriptie Bachelor Kunstmatige Intelligentie, UU, 7.5 EC

14 juli 2021

Inhoudsopgave

1 Samenvatting

Er is onderzocht wat de invloed van het verwerken van equivariantie in neurale netwerken is op de prestaties in het spelen van het spel Snake. De netwerken zijn getraind met een variant van het *reinforcement learning* algoritme Monte Carlo Exploring Starts.
Het verwerken van equivarianties verbetert in sommige situaties de prestaties van de netwerken. De netwerken die equivariant zijn onder translatie presteren beter dan degene die dat niet is. Het netwerk dat daarnaast equivariant is onder roteren en spiegelen, presteert bij veel beschikbare ervaring beter dan het netwerk dat alleen gebruikmaakt van equivariantie onder translatie, en evenveel trainbare parameters heeft. Het presteert echter slechter als het andere netwerk

evenveel rekencapaciteit wordt gegeven, en dus meer trainbare parameters. Dit laatste netwerk heeft echter juist meer slechte uitschieters als er minder data beschikbaar zijn. De gemiddelde prestaties verschillen echter niet significant.

2 Introductie

Reinforcement learning is een vorm van leren waarbij een agent acties onderneemt in een omgeving en leert van de beloning die hij krijgt, met het doel om de beloning te maximaliseren. Als computers deze vorm van leren effectief uit kunnen voeren, kunnen die veel verschillende problemen oplossen zonder expliciet geprogrammeerd te worden voor elke taak. *Reinforcement learning* wordt vaak gecombineerd met *deep learning*. Hiermee zijn indrukwekkende resultaten behaald, zoals Agent57, die 57 verschillende Atari games heeft leren spelen (Badia et al., 2020).
Een nadeel van *deep learning* is dat er veel data nodig is om goede prestaties te behalen (Gavrishchaka, Yang, Miao & Senyukova, 2018; Diligenti, Roychowdhury & Gori, 2017). Met name wanneer er geen simulatie beschikbaar is, en de agent in de echte wereld zijn ervaring moet opdoen, kan het duur zijn om veel data te verzamelen. Een manier om de benodigde data te verminderen is door voorkennis te verwerken in de architectuur van het neurale netwerk (Diligenti et al., 2017). Daarmee wordt de techniek minder algemeen, maar er zijn ook technieken die breed toepasbaar zijn.
Een voorbeeld hiervan zijn convolutionele neurale netwerken (CNN's). Deze maken gebruik van equivariantie onder translatie (Cohen & Welling, 2016). Ze gaan er ook van uit dat data die samen verwerkt moeten worden vaak dicht bij elkaar liggen in de ruimte. Een functie f is equivariant onder de transformaties in groep G (die bij CNN's bestaat uit alle mogeljke translaties) als $t \in G$ eerst toepassen op de de invoer en daarna f op de getransformeerde invoer tot hetzelfde resultaat leidt als direct f toepassen op de invoer en daarna t op de uitvoer van f.
Er zijn ook problemen die equivariant zijn onder rotatie en spiegelen. Zoals de meeste problemen die zich in een 2D ruimte, *top-down* perspectief, afspelen. Cohen en Welling (Cohen & Welling, 2016) hebben een architectuur voorgesteld die met equivarianties om kan gaan, in heb bijzonder met spiegelen en roteren. Zij hebben deze destijds geëvalueerd op basis van prestatie in *supervised learning* taken.
In het huidge onderzoek zal worden onderzocht wat de invloed is van het verwerken van verschillende equivarianties in de netwerkarchitectuur, in de context van *deep reinforcement learning*. De taak die hiervoor gekozen is, is het spelen van de game Snake op een klein bord van 7 bij 7 tegels. Deze taak is gekozen omdat die equivariant is onder rotatie, spiegeling, en tot op zekere hoogte onder translatie – of zelfs volledig wanneer de randen van het bord mee worden getransleerd. Snake is ook eenvoudig te programmeren, maar niet triviaal te spelen zodra het bord voller raakt. De architecturen die vergeleken worden zijn een standaard volledig verbonden netwerk, een CNN, en de groep-equivariante

CNN zoals voorgesteld door Cohen en Welling, met de groep P4M (spiegelen en rotaties die een veelvoud zijn van 90 graden). Zowel prestaties onder beperkte ervaring als onder een grotere hoeveelheid ervaring worden onderzocht. Dit leidt tot de volgende onderzoeksvraag:

Wat is de invloed van het verwerken van equivarianties in een neuraal netwerk op zijn prestaties in de game Snake (d.w.z. het gemiddeld aantal gegeten appel-tjes), bij een ruime en bij een beperkte hoeveelheid ervaring?

De netwerken worden getraind volgens een algoritme dat lijkt op Monte Carlo Exploring Starts (Sutton & Barto, 2018), waar de actiekwaliteitfunctie is vervangen door een neuraal netwerk.

3 Voorkennis

3.1 Regels van Snake

Hoewel Snake een redelijk bekend spel is, zullen de regels hier nog een keer herhaald worden. Sommige details verschillen per implementatie. Hier zal het spel worden beschreven zoals het geïmplementeerd is voor dit onderzoek.
Het spel speelt zich af op een raster van 7 bij 7. De speler bestuurt een slang die appels wilt eten. Elke tijdstap kan de speler één van de 4 windrichtingen kiezen waarheen het hoofd van de slang moet bewegen. Het kenmerkende systeem van dit spel is dat staart van de slang achter het hoofd aankomt door exact hetzelfde pad te volgen als het hoofd. Als het hoofd van de slang op een tegel komt waar zijn eigen staart is, gaat hij dood en is het game over. De slang sterft ook als die naar een locatie buiten het speelbord probeert te bewegen.
Als het hoofd van de slang het vakje betreedt waar de appel is, krijgt de speler een punt. De staart wordt dan 1 tegel langer, en het spel dus moeilijker. Ook wordt er een appel geplaatst op een nieuwe, willekeurige locatie die daarvoor leeg was. Er is altijd maar 1 appel tegelijk in het spel.
Het spel wordt geïnitialiseerd met alleen het hoofd van de slang in het midden en de appel op een willekeurige plek. In beurt 0 heeft de slang nog geen staart, maar die groeit over de eerste 2 beurten tot lengte 2 (of bijvoorbeeld tot lengte 3 over de eerste 3 beurten als hij in de eerste paar beurten meteen al een appel eet). De slang is dan in totaal 3 vakjes groot: 1 vakje met het hoofd, en 2 met de staart. Deze lengte is gekozen omdat het de grootste lengte is waarbij de slang altijd een actie beschikbaar heeft waaraan hij niet meteen sterft. Hij moet dus minimaal 1 keer de appel eten en dan groeien, om dood te kunnen gaan. De AI (Artificiële Intelligentie) is namelijk zo geprogrammerd dat hij nooit een actie onderneemt waaraan de slang meteen dood gaat, als dat niet hoeft. Zo krijgt hij minimaal een beloning van 1, ervan uitgaande dat het spel niet gestopt wordt omdat het te lang duurt.
Er is voor een aantal redenen gekozen voor een relatief klein veld van 7 bij 7. Het is minder interessant om de prestaties van verschillende netwerkarchitectu-

ren te bestuderen in situaties waarin het erg gemakkelijk is om bij de appel te komen. Het spel wordt moeilijk zodra de slang een groot deel van het veld in beslag neemt. Daarnaast kost het minder rekenkracht om een klein speelveld te verwerken met een neuraal netwerk, vergeleken met een groot speelveld. Dit, in combinatie met dat het spel korter zal duren, zorgt ervoor dat de experimenten vaker kunnen worden uitgevoerd.
Er is ook de regel toegevoegd dat de slang dood gaat, en het spel dus eindigt, als het hele spelbord gevuld is met de appel of de slang zelf. Hierdoor is de maximale score 45. Er zijn immers 7x7=49 tegels, de slang neemt in het begin 3 tegels in beslag, en het voedsel altijd 1. En 49-3-1=45. Dit was nodig omdat het programma anders crashte nadat de laatste appel was gepakt, omdat er geen ruimte meer was om een nieuwe appel te plaatsen.
Ook was het nodig om het spel te stoppen na een eindig aantal stappen. Soms ontstond er namelijk een situatie waarin de AI steeds dezelfde reeks acties koos en in een lus kwam. Het spel kan niet langer dan 5000 stappen duren. (Een stap bestaat uit dat de speler een richting kiest om heen te bewegen en het uitvoeren van die beweging.)
Voor mensen is er een extra uitdaging doordat ze beperkte tijd hebben om een richting te kiezen, anders herhaalt de slang de laatste actie. Meestal heeft de speler minder tijd naarmate het spel vordert. Dit is niet verwerkt in de regels, omdat de tijd die nodig is om een keuze te maken afhankelijk is van de hardware, en computers over het algemeen een uitstekend reactievermogen hebben. Op de computers waarop de experimenten zijn uitgevoerd waren de netwerken in staat vele duizenden keuzes per seconde te maken.

Het spel is volledig geïmplementeerd in termen van PyTorch operaties. Dit is gedaan omdat het neurale netwerk op de GPU wordt uitgevoerd. Op deze manier kunnen de data op de GPU blijven. De verwachting is dat dit het hele trainingsproces sneller maakt, maar dit is niet formeel gemeten.

3.2 Tensors

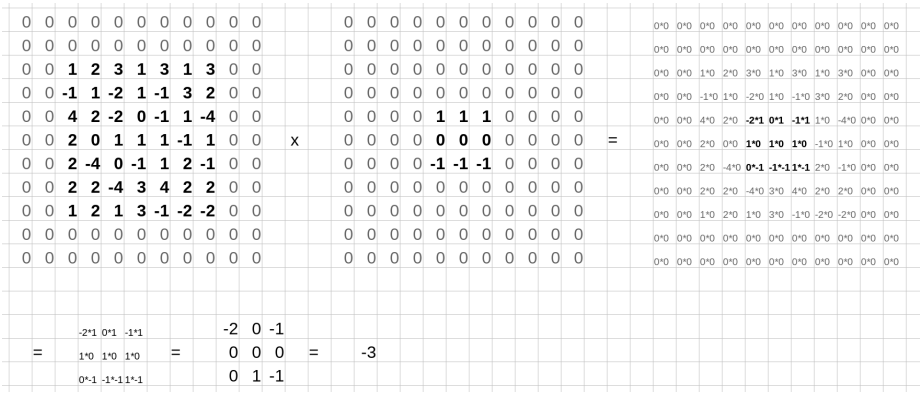
In dit artikel worden *tensors* gebruikt voor multidimensionale *arrays* van getallen, bijvoorbeeld om de toestanden en gewichten van neurale netwerken te noteren. In deze sectie wordt de gebruikte notatie kort toegelicht.
Om de formules simpel te houden kunnen dimensies een eindige of oneindige lengte hebben, hoewel die in de implementatie natuurlijk eindig moet zijn. Dimensies van oneindige lengte worden alleen gebruikt als het triviaal is om het equivalent te implementeren met eindige lengte.
Indices worden tussen vierkante haken achter de naam van de *tensor* geschreven. Als een *tensor* met d dimensie wordt geïndiceerd door i indices, resulteert dat in een *tensor* met $d - i$ dimensie. 0-dimensionale *tensors* zijn scalaire waarden, oftewel één getal. Bijvoorbeeld: als T een 4-dimensionale *tensor* is, dan is $T[1, 3]$ eeen 2-dimensionale *tensor*, en $T[1, 3, 2, 7]$ een getal.

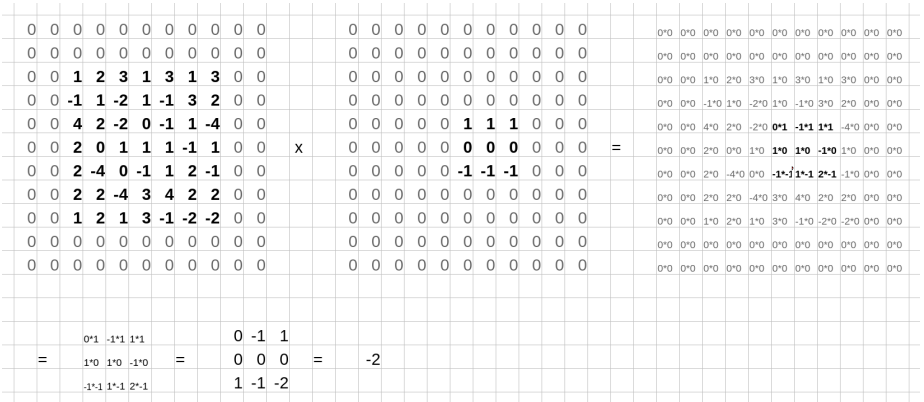
3.3 Convolutionele neurale netwerken

Eén van de architecturen die vergeleken wordt, is een convolutioneel neuraal netwerk (CNN). Convolutionele operaties werken meestal op 2-dimensionale, discrete ruimtes, zoals afbeeldingen die uit pixels bestaan of het speelbord van Snake dat uit vakjes bestaat. Er bestaan ook convolutionele operaties voor 1 of 3 dimensies, maar die worden hier niet behandeld.
Meestal zijn er meerdere getallen per punt in de ruimte. Een afbeelding heeft bijvoorbeeld meestal 3 waardes per pixel: de hoeveelheden dat de 3 primaire kleuren rood, groen en blauw aanwezig zijn. In deze uitleg zal eerst het geval behandeld worden van een ruimte met 1 getal per punt. Vervolgens zal het geval met meerdere kanalen behandeld worden (dat wil zeggen: meerdere getallen per punt). Uiteindelijk wordt het samengevat in een formule.
In de praktijk hebben de ruimtes altijd eindige afmetingen, bijvoorbeeld 1920 breed en 1080 hoog voor een HD afbeelding. Het is echter behulpzaam om over de ruimte te denken alsof deze oneindig uitstrekt, gevuld met de waarde 0 in de punten die buiten deze eindige hoogte en breedte vallen.
Het resultaat van de convolutionele operatie vormt de waardes in de volgende laag van het neurale netwerk, noem deze ruimte U. Om deze te berekenen wordt een serie vermenigvuldigingen en optellingen gedaan tussen de waardes in de vorige laag, noem deze ruimte I, en de ruimte met gewichten, noem deze G. Deze ruimte G bevat weer oneindig veel 0'en, en in een eindig gebied daadwerkelijk "nuttige" waardes. Dit eindige gebied in G noemen we de filter, en is in de praktijk meestal kleiner dan het eindige gebied in I en U. De waardes in de filter kunnen geleerd worden met een algoritme zoals backpropagation.
Het is het makkelijkst om eerst uit te leggen hoe de waarde van het punt (0, 0) in U wordt uitgerekend. Dit gaat als volgt:

1. Leg de 2 ruimtes I en G "op elkaar" om zo een nieuwe ruimte te krijgen met 2 getallen op elk punt.
2. In elk punt, vermenigvuldig de 2 getallen met elkaar
3. Sommeer de resultaten van de vermenigvuldigingen

Om de waarde in het punt (0, 1) in U uit te rekenen, wordt ongeveer dezelfde berekeningen gedaan. De enige uitzondering is dat alle gewichten in G eerst 1 naar rechts verplaatst worden. Hieronder zijn de 2 berekeningen voor (0, 0) en (0, 1) met een getallenvoorbeeld uitgewerkt.





Deze berekening wordt herhaald voor elk punt (x, y) in de laag U, waarbij de gewichten steeds x naar rechts en y naar beneden worden opgeschoven.
In de praktijk bestaat een punt meestal niet uit 1 getal, maar bevat het meerdere "kanalen", met 1 getal per kanaal. Om de waardes in kanaal k_U van de uitvoer te berekenen, wordt voor kanaal k_I van de invoer de hierboven beschreven berekening uitgevoerd, en worden de resultaten bij elkaar opgeteld. Voor elke combinatie van k_U en k_I is er een aparte verzameling gewichten. Dit betekent dat er dus $a_G = a_I * a_U$ verzamelingen gewichten zijn, waar a_I het aantal kanalen in de invoer is en a_U in de uitvoer.

Dit kan formeel worden samengevat in de volgende formule:

$$U[x_{uit}, y_{uit}, k_{uit}] = \sum_{(x_{in},y_{in})\in\mathbb{Z}^2} (\sum_{k_{in}=1}^{a_I} (I[x_{in}, y_{in}, k] * schuif(W[k_{uit}, k_{in}], x_{uit}, y_{uit})[x_{in}, y_{in}])),$$

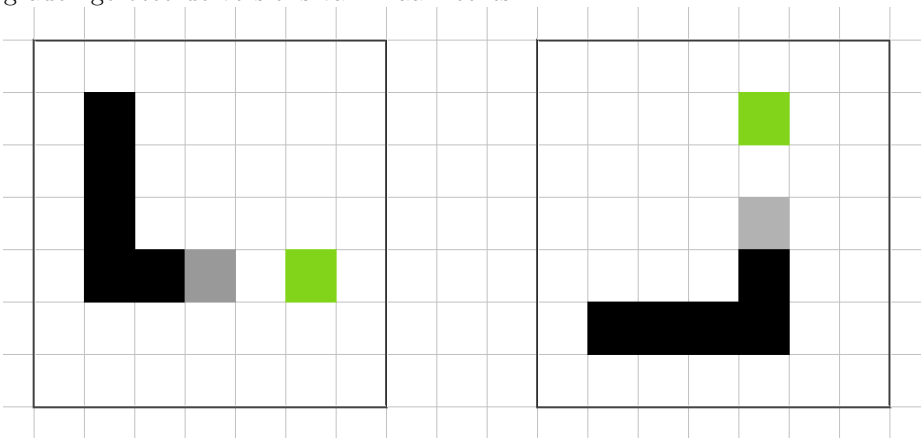
waarin geldt dat:
$schuif(a, dx, dy)[x, y] = a[x - dx, y - dy]$
W is een 4-dimensionale *tensor*. In deze formule worden eerst 2 indices meegegeven, wat resulteert in een 2-dimensionale *tensor*. Deze tensor wordt vervolgens

met x_{uit} naar rechts en y_{uit} naar beneden geschoven. Uiteindelijk worden nog 2 indices gegeven, om een scalaire waarde te krijgen.

De convolutionele operatie heeft een aantal voordelen over de standaard matrix vermenigvuldiging die in volledig verbonden lagen wordt gebruikt. Onder andere heeft die de eigenschap dat als de invoer getransleerd wordt, de uitvoer hetzelfde blijft, behalve dat het op dezelfde manier getransleerd is. Dit noemen we "equivariant onder translatie" (Cohen & Welling, 2016). Dit is gunstig in Snake, omdat de beste actie vaak hetzelfde zal zijn als zowel het voedsel als de slang bijvoorbeeld 1 vakje naar links worden verschoven. De verwachting is dan ook dat de CNN architectuur die uit 3 convolutionele operaties en 1 volledig verbonden operatie bestaat, sneller leert dan die alleen uit volledig verbonden operaties bestaat.
Een andere eigenschap is dat er wordt aangenomen dat gegevens die samen verwerkt moeten worden, vaak dicht bij elkaar liggen. Dit komt doordat de filter vaak redelijk klein is. Doordat er minder gewichten zijn, overfit het model ook minder snel. Hoe relevant deze aanname is voor Snake, is niet bekend.

3.4 Groep-equivariante CNN's

Naast translatie, is Snake equivariant onder rotatie en spiegelen. Hiermee wordt bedoeld dat als het spel een veelvoud van 90 graden wordt geroteerd, of wordt gespiegeld in de x of y-as, dezelfde transformatie kan worden toegepast op de beste actie op de nieuwe beste actie te krijgen. In het voorbeeld hieronder is het rechter speelbord gelijk aan het linker speelbord 90 graden tegen de klok in gedraaid. In de linker situatie is de beste actie waarschijnlijk "naar rechts". In de rechter situatie is de beste actie waarschijnlijk "naar boven", wat de 90 graden geroteerde versie is van "naar rechts".



Om gebruik te maken van deze eigenschap, kunnen de aangepaste convolutionele operaties gebruikt worden die Taco Cohen en Max Welling hebben voorgesteld (Cohen & Welling, 2016). Deze werken al volgt:
In het geval van Snake zijn er, naast translatie, 8 equivariante transformaties.

Dit zijn alle combinaties van rotaties met spiegelen. Er zijn 4 rotaties: geen rotatie, 90 graden, 180 graden en 270 graden, en 2 opties voor spiegelen: wel en niet gespiegeld. Alleen verticaal óf horizontaal spiegelen hoeft expliciet mogelijk zijn, want de één kan gemaakt worden uit de andere gecombineerd met rotaties.

Het berekenen de groep-equivariante convolutie lijkt erg om het berekenen van een normale convolutie, behalve dat er in de verborgen lagen van het netwerk steeds 8 aparte ruimtes zijn: 1 voor elk van de equivariante operaties. De eerste laag (die in het geval van Snake het speelbord representeert) is nog steeds 2-dimensionaal. Daarom gaat het berekenen van de eerste convolutie net iets anders dan de latere. We behandelen eerst de eerste convolutie.
Neem bijvoorbeeld het punt in de eerste verborgen laag op coördinaat (5, 3) in de ruimte die de operatie "roteer 90 graden" representeert. De waardes in dit punt worden op dezelfde manier uitgerekend als bij een normale convolutie, behalve dat de ruimte met gewichten niet alleen 5 naar rechts en 3 naar beneden worden verplaatst, maar vervolgens ook 90 graden wordt gedraaid.

De operatie tussen 2 verborgen lagen gaat ongeveer hetzelfde, behalve dat het iets ingewikkelder wordt omdat de invoer laag nu ook meerdere ruimtes heeft die equivariante operaties representeren.
Neem bijvoorbeeld het punt (5, 3) in de ruimte die "roteer 180 graden en spiegel" representeert van de 2 verborgen laag. De bijdrage van de ruimte "roteer 90 graden" in de eerste verborgen laag wordt berekend door eerst uit te rekenen welke operatie nodig is om van "roteer 90 graden" te gaan naar "roteer 180 graden en spiegel". In dit geval is dat "roteer (nogmaals) 90 graden en spiegel". Vervolgens wordt opgezocht welke ruimte gewichten bij deze operatie hoort, en wordt deze 5 naar rechts en 3 naar beneden geschoven, 90 graden gedraaid en dan gespiegeld. Uiteindelijke wordt er gesommeerd over alle bijdrages van de ruimtes in de vorige laag, om zo de waarde in punt (5, 3) te krijgen.

Deze 2 operaties kunnen formeel worden gemaakt met de volgende formules:

$$U[x_{uit}, y_{uit}, k_{uit}, index(t_{uit})] = \sum_{(x_{in}, y_{in}) \in \mathbb{Z}^2} (\sum_{k_{in}=1}^{a_I} ($$

$$I[x_{in}, y_{in}, k] * t_{uit}(schuif(W[k_{uit}, k_{in}], x_{uit}, y_{uit}))[x_{in}, y_{in}]))$$

en

$$U[x_{uit}, y_{uit}, k_{uit}, index(t_{uit})] = \sum_{(x_{in}, y_{in}) \in \mathbb{Z}^2} (\sum_{k_{in}=1}^{a_I} (\sum_{t_{in} \in T} ($$

$$I[x_{in}, y_{in}, k_{in}, index(t_{in})]$$

$$*$$

$$(t_{uit} \circ t_{in}^{-1})(schuif(W[k_{uit}, k_{in}, index(t_{uit} \circ t_{in}^{-1})], x_{uit}, y_{uit}))[x_{in}, y_{in}]))),$$

waarin geldt dat:
T is de verzameling van de 8 combinaties tussen roteren en spiegelen. De elementen uit T (dat zijn in deze formules t_{in} en t_{uit}) zijn functies van een 2-dimensionale *tensor* naar een geroteerde en/of gespiegelde 2-dimensionale *tensor*.
$index(t)$ is een bijectie die elk element uit T associeert met een getal van 1 tot 8.
W en $schuif$ hebben dezelfde definitie als in de formule voor de standaard convolutionele operatie, behalve dat W in de 2^{de} formule een extra dimensie heeft die de groepselementen representeert.

Merk op dat de gebruikte gewichten afhankelijk zijn van $t_{uit} \circ t_{in}^{-1}$. Om van "90 graden draaien" naar "180 graden draaien" te gaan worden dezelfde gewichten gebruikt als van "180 graden draaien" naar "270 graden draaien", want het verschil is in beide gevallen "90 graden draaien". Om van "90 graden draaien" naar "90 graden draaien en spiegelen" te gaan, worden echter andere gewichten gebruikt, omdat hier het verschil "spiegelen" is. Dit betekent dat er $8 * aantal\_kanalen$ ruimtes met gewichten zijn. (Cohen & Welling, 2016)

3.5 Rectified Linear Unit (ReLU)

De activatiefunctie die in de netwerken wordt gebruikt is *Rectified Linear Unit* (ReLU). De functie is als volgt gedefinieerd (Qiu, Xu & Cai, 2018):

$$ReLU(x) = x, \text{ als } x \geq 0$$

$$ReLU(x) = 0, \text{ als } x \leq 0$$

3.6 Softmax

Een andere functie die vaak als activatiefunctie wordt gebruikt, en in dit onderzoek wordt gebruikt om op een goede manier willekeurigheid te introduceren in de AI, is softmax. Softmax is als volgt gedefinieerd (Martins & Astudillo, 2016):

$$softmax_i(\vec{v}) = \frac{e^{v_i}}{\sum_j (e^{v_j})}$$

4 Methode

4.1 Leerstrategie

Er zijn een aantal strategieën geprobeerd om het netwerk te trainen: het netwerk voor elk van de 4 acties laten inschatten of het het huidige voedsel gaat bereiken, Q-learning, en Monte-Carlo Exploring Starts (MCES) (Sutton & Barto, 2018) met en zonder verdisconteerde beloning. Bij Q-learning en MCES was de Q-functie (degene die de verwachte beloning voor elk toestand-actie paar representeert) vervangen door het neurale netwerk. Uit een aantal informele

experimenten is gebleken dat MCES met verdisconteerde beloning het beste werkt.
MCES houdt een functie (in dit geval het neurale netwerk) bij die een inschatting maakt van hoe goed elke actie is om te nemen in elke toestand van het spel. Deze wordt aan het einde van elk gespeelde spel aangepast door voor elk toestand-actie paar dat voorgekomen is in het spel uit te rekenen wat de (verdisconteerde) beloning is die is behaald vanaf het moment dat de actie is ondernomen. Het toestand-actie paar als invoer en de behaalde beloning als uitvoer vormen dan een datapunt waar de functie op getraind kan worden.
Bij het berekenen van de beloning die behaald wordt na een gegeven actie neemt de beloning toe met r^b voor elk appel dat de slang zal eten vanaf dan, waar b het aantal beurten is totdat het appel wordt gegeten. Als $r = 1$ is de beloning niet verdisconteerd, en bij $r < 1$ wel. Op basis van een aantal informele experimenten is $r = 0.98$ gekozen. Het disconteren van de beloning is ook gedaan omdat het netwerk dan niet beloningen ver in de toekomst hoeft te voorspellen. Het geeft ook druk om snel naar het voedsel te gaan, in plaats van te wachten tot de slang er toevallig langs komt.
De verdisconteerde beloning wordt alleen gebruikt voor het trainen van de netwerken. Later, wanneer netwerken met elkaar worden vergeleken, wordt naar het totaal aantal gegeten appels gekeken over het hele spel, met een beloning van constant 1 per appel.
Het gebruikte algoritme wijkt verder nog af van zoals het door Sutton en Barto (Sutton & Barto, 2018) beschreven is in het volgende opzicht: er worden 256 spellen in parallel gespeeld, om gebruik te maken van de *multithreading* en *SIMD/SIMT* capaciteiten van GPU's. Er wordt pas getraind als alle 256 spellen klaar zijn. Zoveel spellen tegelijk spelen bleek nauwelijks langzamer dan 1 spel op de gebruikte hardware, en zorgt wel voor meer ervaring waar het netwerk van kan leren.
Ook worden toestand-actie paren die eerder voor zijn gekomen in hetzelfde spel niet overgeslagen in de training, omdat dit lastig was efficiënt te implementeren met PyTorch.

Het netwerk voorspelt altijd 4 waardes: de verwachte beloningen voor elk van de 4 acties. Bij training wordt de *loss* berekend door de voorspelde beloning te selecteren van de actie die uiteindelijk is gekozen. Deze wordt dan vergeleken met de daadwerkelijk verdisconteerde beloning die behaald is vanaf het moment dat die actie is gekozen, door de gekwadrateerde fout (squared error) te berekenen.
Na episodes waarin het spel in een lus kwam (waarin de AI steeds dezelfde reeks acties koos), speelde het netwerk vaak in de episodes daarna ook slecht. De hypothese was dat de gewichten soms zo veel aangepast waren, dat het netwerk veel nuttige kennis verloor. Dit heeft te maken met dat het netwerk opeens erg veel ervaring krijgt waarbij de verdisconteerde beloning ongeveer 0 was. Die episodes zijn immers erg lang. Daarom wordt de *loss* geschaald met een factor van $1000/n\_stappen$, waar $n\_stappen$ het aantal stappen is dat het langste spel heeft geduurd. Op basis van informele experimenten lijkt dit enigszins te

helpen.

4.2 Een actie kiezen

Het netwerk maakt een inschatting van de verdisconteerde beloning. Bij MCES wordt normaal steeds de actie met de hoogste verwachte verdisconteerde beloning gekozen (Sutton & Barto, 2018). Om te voorkomen dat het spel in een lus komt, is er op 2 manieren willekeurigheid toegevoegd. Ten eerste is er een 3% kans de slang een willekeurige actie neemt die er niet toe leidt dat de slang meteen dood gaat. Daarnaast wordt er willekeurigheid bij de wegingen van de acties opgeteld. Dit gebeurt volgens het volgende proces:
Eerst wordt door handgeschreven code bepaalt welke acties ervoor zorgen dat de slang meteen doodgaat. De voorspelde waardes van de acties die er toe leiden dat de slang meteen dood gaat, worden gezet op -100. Daarna wordt softmax toegepast of de 4 waardes. Dit wordt gedaan om de waardes dichter bij elkaar te brengen. De onmogelijke acties zijn nu helaas ook weer redelijk dicht bij de anderen komen te liggen. Daarom worden deze weer ingesteld op -100. Bij de gewichten wordt vervolgens nog een willekeurige hoeveelheid opgeteld, volgens een normaal verdeling met standaard deviatie 0,03 en gemiddelde 0. Uiteindelijk wordt de actie met het hoogste gewicht gekozen.
De waardes voor de standaard deviatie en de kans op een willekeurige actie zijn vastgesteld door verschillende waardes te proberen en zo laag mogelijke waardes te kiezen waarbij het spel niet zo vak in een lus kwam dat dit het leerproces te veel hinderde.

Het was nodig om te verbieden een actie te nemen waarbij de slang meteen dood gaat, omdat de slang eerst meestal dood ging voordat hij voedsel kon eten, waardoor er zelden een beloningssignaal kwam, en het netwerk niet leerde.

4.3 Netwerkinvoer

De netwerken observeren het volledige spel. Voor elke tegel zijn er 4 kanalen:

1. Gelijk aan 0 als de slang (hoofd of staart) niet op deze tegel is. Anders gelijk aan het aantal beurten dat het duurt voordat het stuk van de slang niet meer op deze tegel is. Het aantal beurten is onder de aanname dat de slang geen voedsel eet.
2. 1 als het hoofd van de slang op deze tegel is, anders 0.
3. 1 als er voedsel op deze tegel is, anders 0.
4. 1 als de tegel buiten het speelveld is, anders 0.

Er is een opvulling (padding) van 3 breed aan elke kant van het speelbord, opgevuld met kanaal 4 op waarde 1 en de rest op 0. Dit is gedaan omdat met elk van de 3 lagen in de convolutionele netwerken de ruimte afneemt met 1 in elke richting. Ook maakt dit het makkelijker om te zien waar de randen van het speelveld zijn.

4.4 Leerparameters

Als optimizer is Adam gebruikt met standaard parameters die in het artikel worden voorgesteld waarin Adam wordt geïntroduceerd, behalve een learning rate van 10^{-6}. Deze lage learning rate was nodig omdat anders het netwerk vaak de volgende episode juist slechter speelde. Waarschijnlijk heeft dit te maken met dat het netwerk te veel werd aangepast en nuttige kennis verloren ging. Die parameters zijn (Kingma & Ba, 2014):
$\beta_1 = 0.9$
$\beta_2 = 0.999$
$\epsilon = 10^{-8}$

Het netwerk speelt steeds 256 spellen tegelijkertijd. Als alle 256 spellen afgelopen zijn, wordt het netwerk getraind. Een datapunt bestaat uit een toestand-actie paar als invoer en de verdisconteerde beloning als uitvoer. Er zijn dus veel datapunten per gespeelde spel. De datapunten worden ingedeeld in batches op basis van de beurt waarin die voorkwam. Alle datapunten in de eerste beurt van één van de 256 spellen vormen een batch, net als alle datapunten in de twee beurt, etc. Het aantal datapunten in een batch varieert dus tussen 256 en 1. De batches worden ook op volgorde van beurtnummer behandeld door het leeralgoritme. Deze indeling is gekozen omdat die het makkelijkste te implementeren was.

4.5 Hyperparameters voor netwerkarchitecturen kiezen

In de sectie hierna worden de volledige netwerkarchitecturen beschreven, inclusief hyperparameters zoals het aantal lagen en aantal kanalen of neuronen per laag. De hyperparamters zijn zo veel mogelijk hetzelfde gehouden. Het aantal lagen is bijvoorbeeld steeds hetzelfde, net als de filtergrootte bij de (G-)CNN's. De architecturen G-CNN, CNN en Dense zijn zo ontworpen dat ze ongeveer evenveel trainbare parameters hebben. Een gelijke hoeveelheid parameters bij Dense zorgt er echter voor dat het netwerk erg weinig neuronen zou hebben in de verborgen lagen. Daarom heeft deze twee keer zoveel gekregen.

Het was al vrij snel duidelijk dat bij veel ervaring CNN slechter presteerde dan G-CNN. Een mogelijke verklaring hiervoor is niet alleen dat G-CNN gebruik maakt van meer equivarianties, maar ook dat de trainbare parameters vaker worden gebruikt. Beide netwerken hebben ongeveer evenveel gewichten, maar in de groep-equivariante convolutionele operatie wordt elke ruimte met gewichten 8 keer gebruikt, vergeleken met 1 keer bij een standaard convolutionele operatie (dit volgt uit de formules gegeven in de voorkennis). Ook bestaat de toestand van één verborgen laag uit $8 * aantal\_kanalen * hoogte * breedte$ waardes bij G-CNN, maar slechts uit $aantal\_kanalen * hoogte * breedte$ waardes bij CNN. Dit zorgt er voor dat G-CNN waarschijnlijk meer rekenkracht en geheugen gebruikt dan CNN. Dit kan de oorzaak zijn dat G-CNN beter presteert dan CNN. Er is daarom een extra experiment uitgevoerd waarbij de normale

CNN en de G-CNN vergelijkbare beschikbare middelen hadden.
De gebruikte implementatie drukt de berekening van groep-equivariante convolutie uit in termen van een simpele transformatie van de filter en een gewone convolutie. Dit is mogelijk voor bepaalde groepen, zoals degene die uit rotatie en spiegelen bestaat (Cohen & Welling, 2016). De architecturen worden daarom vergeleken in een situatie waarbij deze convolutie steeds even groot is als de convolutie die gebruikt wordt in de CNN (in termen van hoogte, breedte, aantal kanalen en filtergrootte). Om efficiënt met de beschikbare rekentijd wordt hetzelfde netwerk, G-CNN, gebruikt. Alleen CNN wordt groter gemaakt. Dit grotere netwerk heet CNN-groot.

4.6 Volledige netwerkarchitecturen

In deze worden de volledige netwerkarchitecturen die met elkaar zullen worden vergeleken beschreven in termen van de operaties die worden uitgevoerd.

Netwerk naam: Dense
Aantal trainbare parameters: 12704
Operaties:

1. Filter de tegels buiten speelveld weg, en verwijder het kanaal dat 1 is als de tegel buiten het speelveld is. Vlak de overgebleven *tensor* van 7 bij 7 bij 3 af tot een *tensor* van 1 dimensie met afmeting $7 * 7 * 3 = 147$
2. 3 keer een volledig verbonden laag met output grootte 50 en activatiefunctie ReLU.
3. Volledig verbonden laag met output grootte 4 en activatiefunctie ReLU.

Netwerk naam: CNN
Aantal trainbare parameters: 5416
Operaties:

1. 3 keer een convolutie met een 3 bij 3 filter en activatiefunctie ReLU, en geen zero-padding. Het aantal kanalen in de uitvoer van elke operatie is 12.
2. Vlak de overgebleven *tensor* van 7 bij 7 bij 12 af tot een *tensor* van 1 dimensie met afmeting $7 * 7 * 12 = 588$.
3. Volledig verbonden laag met output grootte 4 en activatiefunctie ReLU.

Netwerk naam: G-CNN
Aantal trainbare parameters: 5713
Operaties:

1. 3 keer een groep-equivariante convolutie met een 3 bij 3 filter en activatiefunctie ReLU, en geen zero-padding. Het aantal kanalen in de uitvoer van elke operatie is 6.

2. Voor elke van de 4 richtingen:
   Zij r de rotatie die bij de richting hoort ("geen rotatie"bij naar boven, "90 graden roteren"bij naar links, "180 graden roteren"bij naar beneden, en "270 graden roteren"bij naar rechts) en m de operatie "spiegelen". Zij v een volledig verbonden operatie (met steeds dezelfde gewichten). Voer de volgende operatie uit voor $t = r^{-1}$ en $t = (r \circ m)^{-1}$ en laat de ingeschatte verwachte beloning bij deze richting gelijk zijn aan de som hiervan.

   (a) Selecteer uit de *tensor* de deelruimte die hoort bij $index(t)$ en pas t toe op deze deelruimte.

   (b) Vlak deze af van 7 bij 7 bij 6 tot een *tensor* van 1 dimensie met afmeting $7 * 7 * 6 = 294$.

   (c) Pas v toe.

   (d) Pas ReLU toe.

De operatie bij (2) is zo ontworpen dat het gehele netwerk equivariant is onder rotatie en spiegelen. Dit zal niet formeel worden bewezen, maar het is wel getest met een aantal invoeren.

Netwerk naam: CNN-groot
Aantal trainbare parameters: 5416
Operaties:

1. 3 keer een convolutie met een 3 bij 3 filter en activatiefunctie ReLU, en geen zero-padding. Het aantal kanalen in de uitvoer van elke operatie is 48.

2. Vlak de overgebleven *tensor* van 7 bij 7 bij 48 af tot een *tensor* van 1 dimensie met afmeting $7 * 7 * 48 = 2352$.

3. Volledig verbonden laag met output grootte 4 en activatiefunctie ReLU.

4.7 Broncode

De volledige broncode can gevonden worden op `https://github.com/aeduin/snake-ai-2`. Inclusief een README met instructies hoe de programma's uitgevoerd kunnen worden.

4.8 Prestaties meten

Om de prestaties van elke architectuur te meten, wordt voor elke architectuur 6 keer een netwerk geïnitialiseerd met willekeurige gewichten. Elk netwerk wordt voor 1000 episodes van 256 spellen getraind. Steeds wordt de gemiddelde score berekend over de 256 spellen. De gewichten die in het beste gemiddelde resulteren worden opgeslagen. Dit wordt gedaan voor de beste van de eerste 100 episodes en die van de volledige 1000 episodes.

Vanwege de willekeurigheid in het spel en de gekozen acties, wordt het gemiddelde over 256 spellen niet aangenomen als de daadwerkelijke kwaliteit van een netwerk. In de plaats daarvan wordt elk van de $6 * 4 * 2 = 48$ opgeslagen verzamelingen gewichten nogmaals in het bijbehorden netwerk geladen, en speelt die 2048 spellen. Het gemiddelde over die 2048 spellen is dan een datapunt. Voor elke architectuur zijn er dus 6 datapunten bij 100 episodes en 6 datapunten bij 1000 episodes.

5 Resultaten

De scores van CNN, G-CNN en Dense, berekend zoals beschreven onder Prestaties meten, staan in de tabel hier direct onder. De scores voor CNN-groot staan in een tabel verderop.

naam / episodes	100	1000
CNN	11,09	15,93
CNN	11,42	19,41
CNN	12,37	16,78
CNN	11,5	16,63
CNN	11,94	16,19
CNN	12,73	15,77
G-CNN	11,41	26,59
G-CNN	15,62	28,78
G-CNN	10,89	26,38
G-CNN	12,29	24,48
G-CNN	12,45	22,07
G-CNN	13,61	17,98
Dense	6,15	15,69
Dense	10,61	14,43
Dense	5,47	16,68
Dense	5,97	7,39
Dense	5,85	14,15
Dense	9,22	17,68

One-way ANOVA is toegepast op de kolom van 100 episodes, waarbij between groups $df = 2$, within groups $df = 15$. Er is een significant verschil tussen de architecturen ($p < 0.001$)
CNN had een gemiddelde van 11,84 en standaard deviatie van 0,6221
G-CNN had een gemiddelde van 12,71 en standaard deviatie van 1,702
Dense had een gemiddelde van 7,213 en standaard deviatie van 2,150
Als post hoc test is Bonferroni correctie toegepast. Hier komt uit:
CNN en G-CNN presteren niet significant anders
CNN presteert significant beter dan Dense ($p = 0,001$)
G-CNN presteert significant beter dan Dense ($p < 0,001$)

One-way ANOVA is ook toegepast op de kolom van 1000 episodes, waarbij *df* hetzelfde is. Er is een significant verschil tussen de architecturen ($p < 0.001$)
CNN had een gemiddelde van 16,79 en standaard deviatie van 1,343
G-CNN had een gemiddelde van 24,38 en standaard deviatie van 3,860
Dense had een gemiddelde van 14,34 en standaard deviatie van 3,654
Als post hoc test is Bonferroni correctie toegepast. Hier komt uit:
CNN presteert significant slechter dan G-CNN ($p = 0,003$)
CNN en Dense presteren niet significant anders
G-CNN presteert significant beter dan Dense ($p < 0,001$)

naam / episode	100	1000
CNN-groot	7,27	30,58
CNN-groot	13,88	32,98
CNN-groot	7,86	27,81
CNN-groot	14,26	32,81
CNN-groot	14,21	31,40
CNN-groot	13,77	32,62

Er is een t-test uitgevoerd tussen CNN-groot en G-CNN voor 100 en voor 1000 episodes. Voor 100 episodes heeft CNN-groot een gemiddelde van 11,87 en standaard deviatie van 3,350. Voor 1000 episodes zijn dat 31,37 en 1,976.
Bij 100 episodes verschillen G-CNN en CNN-groot niet significant anders.
Bij 1000 episodes presteert G-CNN significant slechter dan CNN-groot (p = 0,0049).

6 Discussie

Uit de experimenten blijkt dat de G-CNN en CNN initieel vergelijkbaar presteren, maar wel beter dan het volledig verbonden netwerk. Daarna waren er nog grote verbeteringen in de prestaties van G-CNN en het volledig verbonden netwerk (ongeveer een verdubbeling), maar CNN verbeterde minder. Uiteindelijk presteren CNN en Dense vergelijkbaar, en G-CNN het beste van deze 3.
De hypothese dat de betere prestaties van G-CNN ten opzichte van CNN vooral komen door de toename in gebruikte rekenkracht en geheugen lijkt te kloppen. CNN-groot presteert namelijk significant (nog) beter dan G-CNN. Dat de equivariantie vastgeprogrammeerd zit in de architectuur zit de prestaties uiteindelijk zelfs in de weg. Dit kan verklaard worden met dat de andere netwerken uiteindelijk ook functies leren die ongeveer equivariant zijn, maar dat er efficiëntere manieren zijn om die te berekenen, dan wanneer elke stap in de berekening ook equivariant moet zijn.
Een belangrijke factor in het bepalen van de verdisconteerde beloning is bijvoorbeeld hoe vol het speelveld is, of equivalent, hoe lang de slang is. In Dense kan er één neuroon zijn die dit uitrekent. In de andere architecturen moet er echter een volledig kanaal aan besteed worden, wat natuurlijk minder efficiënt is. Bij

G-CNN is een heel kanaal relatief duurder, omdat er maar 6 van zijn.

Een ander interessant resultaat is dat G-CNN en CNN vergelijkbaar presteren bij een kleine hoevelheid ervaring, maar Dense slechter. In de inleiding was als motivate gegeven dat het verwerken van equivarianties de benodigde ervaring kan verminderen. Aangezien CNN beter presteert dan Dense, lijkt dit tot op zekere hoogte het geval te zijn. Maar nog meer equivarianties dan translatie equivariantie verwerken lijkt niet veel toe te voegen.
Wat wel opvalt is dat Dense en CNN-groot, de netwerken met de meeste trainbare parameters, ook meer uitschieters hebben naar beneden in de 100 episodes situatie. Blijkbaar zorgt een grotere bewegingsvrijheid er voor dat het netwerk soms slecht wordt getraind.

In de situatie met minder data, worden er nog steeds 100 keer 256 spellen gespeeld van 100'en beurten. Dit is nog steeds een relatief grote hoevelheid data. Er kunnen op basis van dit onderzoek geen conclusies worden getrokken over situaties waarin nog minder data beschikbaar zijn. Daarover meer onder Toekomstig werk.

7 Toekomstig werk

Per keer dat het leerproces werd uitgevoerd waren er grote verschillen tussen de resultaten vroeg in het leerproces. Daarom zijn die niet met elkaar vergeleken. Deze verschillen kunnen onder andere komen doordat er keuzes zijn gemaakt in het leeralgoritme en de netwerkarchitecturen die mogelijk suboptimaal zijn, of in ieder geval niet getest, omdat het doel niet was de beste AI te maken, maar om architecturen te vergelijken. Een toekomstig onderzoek zou het leerproces kunnen verbeteren en stabieler maken om te onderzoeken wat de invloed van equivarianties is in situaties met nog minder data.
De indeling van batches is bijvoorbeeld waarschijnlijk niet optimaal, maar was gekozen omdat die het makkelijkste te implementeren is. Als het doel is een beter AI te maken, met een stabieler leerproces, zou geprobeerd kunnen worden of een willekeurige indeling van datapunten in batches van gelijke grootte beter werkt.
Wat ook kan verbeteren is het gebruikmaken van *max-pooling* operaties. Dit maakt de prestaties van de netwerken waarschijnlijk beter, maar misschien niet het leerproces.

Verder hebben CNN-groot en G-CNN vergelijkbare beschikbare middelen. Het zou nog interessant zijn om te kijken hoe een variant van Dense presteert als die ook evenveel beschikbare middelen heeft. Dit is echter lastig te definiëren. De tijd die nodig is voor het uitvoeren van de netwerken is namelijk afhankelijk van de gebruikte implementatie en hardware. Het kan gebeuren dat met PyTorch op de CPU een netwerk sneller is dan een andere, terwijl die andere juist sneller is met TensorFlow op de GPU. Dit definiëren en de experimenten uitvoeren

wordt overgelaten aan een toekomstig onderzoek. Een mogelijke definitie zou zijn "dezelfde hoeveelheid *floating point* operaties".

Voordat er algemenere conclusies kunnen worden getrokken over *reinforcement learning* en het verwerken van equivariantie in het model, moeten er ook onderzoeken gedaan worden waarin er naar andere problemen wordt gekeken. Snake is een simpel probleem vergeleken met wat je kan tegenkomen in de echte wereld, zoals auto rijden, een huis schoonmaken of robots in een logistieke onderneming aansturen.
Ook zou er gekeken kunnen worden naar problemen die op een andere manier equivariant zijn. Sommige problemen zijn bijvoorbeeld niet alleen equivariant onder rotaties die veelvouden van 90 graden zijn, maar over het gehele continuë spectrum van rotaties. Mogelijk heeft dit meer invloed op de prestaties dan bij een groep van transformaties die uit slechts 8 elementen bestaat.

8 Conclusie

Tot op zekere hoogte help het verwerken van equivarianties. In de situatie met minder data (100 episodes) presteren de netwerken die gebruik maken van equivarianties beter dan het standaard volledig verbonden netwerk. Degene die gebruik maakte van rotatie en spiegelen bovenop translatie presteerde beter dan degene die alleen gebruik maakte van translatie in de situatie met veel ervaring en evenveel trainbare parameters. Hij had ook minder slechte uitschieters in de situatie met evenveel beschikbare rekenmiddelen en weinig ervaring. In de laatste situatie werd het netwerk zonder rotatie en spiegelen uiteindelijk wel het beste naarmate er meer ervaring beschikbaar was.
De conclusie is dat het verwerken van equivariantie in de netwerkarchitecturen kan helpen om sneller een beter netwerk te trainen, maar met meer ervaring kan dit juist de prestaties in de weg zitten.

Referenties

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D. & Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. In *International conference on machine learning* (pp. 507–517).

Cohen, T. & Welling, M. (2016). Group equivariant convolutional networks. In *International conference on machine learning* (pp. 2990–2999).

Diligenti, M., Roychowdhury, S. & Gori, M. (2017). Integrating prior knowledge into deep learning. In *2017 16th ieee international conference on machine learning and applications (icmla)* (p. 920-923). doi: 10.1109/ICMLA.2017.00-37

Gavrishchaka, V., Yang, Z., Miao, R. & Senyukova, O. (2018). Advantages of hybrid deep learning frameworks in applications with limited data. *International Journal of Machine Learning and Computing*, *8*(6), 549–558.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Martins, A. & Astudillo, R. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning* (pp. 1614–1623).

Qiu, S., Xu, X. & Cai, B. (2018). Frelu: flexible rectified linear units for improving convolutional neural networks. In *2018 24th international conference on pattern recognition (icpr)* (pp. 1223–1228).

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*.