

Evolutiestrategieën voor het ondertitelen van afbeeldingen met neurale netwerken

Ruben Cartuyvels

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen, hoofdoptie
Artificiële intelligentie

Promotor:
Prof. dr. M.-F. Moens

Assessoren:
Prof. dr. ir. D. Roose
Ir. T. Leeuwenberg

Begeleider:
Ir. G. Spinks

© Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail info@cs.kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Ik zou graag iedereen bedanken die mij geholpen heeft tijdens het schrijven van deze thesis. In het bijzonder wil ik mijn begeleider, Graham Spinks, bedanken voor zijn raad en begeleiding. Ook mijn promotor, prof. M.-F. Moens, dank ik voor haar feedback. Vervolgens ben ik mijn vriendin, Margot, en mijn familie uiterst dankbaar voor hun steun.

Ruben Cartuyvels

Inhoudsopgave

Voorwoord	i
Inhoudsopgave	ii
Samenvatting	iii
Lijst van figuren, tabellen en algoritmes	iv
Lijst van symbolen en afkortingen	vii
1 Inleiding	1
2 Literatuurstudie	5
2.1 Machine learning en deep learning	5
2.2 Natural Language Processing	12
2.3 Neuro-evolutie	25
3 Evolutiestrategieën voor het ondertitelen van afbeeldingen met neurale netwerken	39
3.1 NIC-ES: een evolutiestrategie voor IC met neurale netwerken	40
3.2 NIC-NES: een natuurlijke evolutiestrategie voor IC met neurale netwerken	50
4 Experimenten & resultaten	53
4.1 Voorbereiding en praktische overwegingen	53
4.2 Experimenten	54
4.3 Verdere discussie	65
4.4 Suggesties voor verder onderzoek	68
5 Conclusie	71
Referenties	73
A Overzicht van de hyperparameters	81
B Voorbeelden van gegenereerde ondertitels	83
C Plots van NIC-ES en NIC-NES tijdens het trainen	85

Samenvatting

Deze thesis stelt voor om neurale netwerken voor tekstgeneratie te trainen met neuro-evolutie. Twee algoritmes worden voorgesteld: NIC-ES en NIC-NES, resp. een evolutiestrategie en een natuurlijke evolutiestrategie.

Neurale netwerken worden tegenwoordig getraind voor tekstgeneratie door ze met cross-entropy te trainen om woord per woord, gegeven het vorige ground-truth-woord, het volgende ground-truth-woord te voorspellen. Tijdens inferentie moeten de netwerken echter hele zinnen ineens genereren, met de zelf voorspelde vorige woorden als invoer. Bovendien worden ze beoordeeld met andere evaluatiefuncties dan de cross-entropy, waar ze voor getraind zijn. Recent werden policy gradients met succes gebruikt om deze discrepanties te verhelpen. Zowel de klassieke training als training met policy gradients wordt echter met gradient descent en backpropagation doorgaans op dure vector processing units zoals GPU's uitgevoerd.

NIC-ES en NIC-NES optimaliseren netwerken, net zoals policy gradients, voor het genereren van hele zinnen en niet van individuele woorden, en voor de relevante evaluatiefunctie. Beide algoritmes zijn zeer paralleliseerbaar, hebben geen GPU's nodig en kunnen op gedistribueerde systemen uitgevoerd worden. Daarom zouden ze kunnen profiteren van de opkomst van parallelisatie en cloud computing.

In de experimenten worden NIC-ES en NIC-NES gebruikt voor het trainen van RNN's om ondertitels te genereren voor afbeeldingen uit de MSCOCO Captions dataset. Daarbij verbeteren ze de CIDEr-score op de testset t.o.v. training met cross-entropy met resp. 2.3% en 8.2%. De voortraining met cross-entropy blijft essentieel. NIC-ES en NIC-NES worden uitgevoerd op 96 CPU's in parallel en convergeren in vergelijkbare tijd met een policy-gradient-methode op een GPU.

Drie varianten op mutaties en drie evaluatiefuncties worden voorgesteld en vergeleken in de experimenten, alsook twee selectiemechanismen in NIC-ES en twee manieren om minibatches te gebruiken in NIC-NES.

Lijst van figuren, tabellen en algoritmes

Lijst van figuren

2.1 Reinforcement learning	6
2.2 Een feedforward neuraal netwerk	8
2.3 Recurrent Neural Network	9
2.4 Teacher forcing en backpropagation through time	16
2.5 Tekstgeneratie met een RNN tijdens inferentie	17
2.6 Ondertitelen van afbeeldingen met CNN als encoder en RNN als decoder	19
2.7 Neural Image Caption Generator	20
2.8 Exposure bias	24
2.9 Evolutionair algoritme	26
3.1 De CNN-LSTM-opstelling en de gezochte parameters van NIC-ES	44
3.2 Evaluatieprocedure van NIC-ES	48
4.1 SCST, NIC-ES en NIC-NES: CIDEr in functie van de tijd	57
4.2 NIC-NES en NIC-ES: mutatievarianten	59
4.3 NIC-ES: toernooiselectie versus uniforme willekeurige selectie	64
4.4 NIC-NES: met en zonder gelijke minibatches	65
B.1 Vergelijking ondertitels	84
C.1 SCST, NIC-ES en NIC-NES: CIDEr in functie van het aantal epochs	85
C.2 SCST, NIC-ES en NIC-NES: bij starten van θ'_{xent}	86
C.3 NIC-NES: trainingrun	87
C.4 NIC-ES: trainingrun	88

Lijst van tabellen

4.1 Waarden voor hyperparameters van NIC-ES en NIC-NES	56
4.2 Scores van XENT, SCST, NIC-ES en NIC-NES op de validatie- en testset	57

4.3 Validatiescores van NIC-ES en NIC-NES met mutatievarianten	60
--	----

Lijst van algoritmes

2.1 Evolutiestrategie van Such et al.	35
3.1 NIC-ES: evolutiestrategie voor IC	41
3.2 NIC-ES: parallelle versie	42
3.3 NIC-NES: natuurlijke evolutiestrategie voor IC	51

Lijst van symbolen en afkortingen

x	Vector
W	Matrix
$W_{m,2}$	Element in W uit rij m , kolom 2
$W_{m,1:n}$	Vector van elementen in W uit rij m en kolommen 1 tot n
\hat{x}	Schatter voor stochastische variabele of sample van kansverdeling
$\mathcal{D}, \mathcal{D}_{val}, \mathcal{D}_{test}$	Dataset: trainingset, validatieset, testset
D	Aantal datapunten in \mathcal{D}
\mathcal{B}	Minibatch uit dataset
B	Aantal datapunten in \mathcal{B}
$x^{(i)}$	Datapunt met index i uit \mathcal{D}
θ	Parameters van een neuraal netwerk
$f_\theta, f(\mathbf{x}; \theta)$	Functie die neuraal netwerk met parameters θ implementeert
h_t	Verborgen toestandsvector van RNN in tijdstap t
$f^{(i)}$	Functie die laag i in neuraal netwerk implementeert
p_{model}	Categorische kansverdeling bepaald door kansen in softmax van uitvoer van model
$J(\theta)$	Loss functie gebruikt in deep learning
\mathcal{V}	Woordenschat
V	Grootte van woordenschat
W_e	Word embeddinglaag in neuraal netwerk: matrix van word embeddings
w_e	Word embedding van een woord
u_i	Woord: index in \mathcal{V}
y	Ground-truth-zin uit (x, y) -paar uit \mathcal{D}
\hat{y}	Zin bestaande uit $\hat{u}_t \sim p_{model}$
y^*	Zin bestaande uit $u_t^* = \arg \max_k p_{model}(u_t = k)$
y^\diamond	Meest waarschijnlijke zin: $\arg \max_y p_{model}(y)$

<i>I</i>	Afbeelding
<i>q</i>	Tussentijdse representatie berekend door encoder in encoder-decoder-framework
γ	Genotype van een evolutionair algoritme
λ	Aantal nakomelingen in een EA
μ	Populatiegrootte in een EA
$F(\gamma)$	Fitness- of evaluatiefunctie gebruikt in EA
σ	Standaardafwijking van mutaties of mutatiesterkte
δ	Mutatievector
s	Vector met sensitiviteitswaarde per element in γ
S	Toernooigrootte bij toernooiselectie in een EA
C	Aantal elitekandidaten bij elitisme in een EA
E	Aantal elites in een EA
P	Aantal iteraties bij patience
H	Aantal iteraties bij schedule
BPTT	Backpropagation through time
CNN	Convolutional Neural Network
EA	Evolutionair algoritme
ES	Evolutiestrategie
IC	Image captioning
LSTM	Long Short-Term Memory
M-STD	Standaardmutaties
ML	Machine learning
NES	Natuurlijke evolutiestrategie
NIC-ES	Neural Image Captioning - Evolution Strategy
NIC-NES	Neural Image Captioning - Natural Evolution Strategy
NN	Neuraal netwerk
PG	Policy gradients
RL	Reinforcement learning
RNN	Recurrent Neural Network
SCST	Self-critical sequence training: een policy-gradient-methode
SGD	Stochastic Gradient Descent
SL	Supervised learning
VM-G	Veilige mutaties door uitvoergradiënten
VM-P	Proportionele mutaties
XENT	Cross-entropy

Hoofdstuk 1

Inleiding

Het genereren van coherente en zinvolle tekst in natuurlijke taal is een kernprobleem in machine learning en artificiële intelligentie. Het begrijpen van afbeeldingen is een andere grote uitdaging. In het ondertitelen van afbeeldingen, of *image captioning*, komen beide taken samen: om een goede ondertitel te kunnen genereren moet een algoritme zowel de essentie uit de afbeelding kunnen halen, als deze essentie kunnen omzetten in een mooie zin.

De potentiële toepassingen van algoritmes die afbeeldingen kunnen ondertitelen zijn talrijk. Recent werd zo een algoritme bv. gebruikt om mensen met een zwak gezichtsvermogen te helpen om afbeeldingen op een website te begrijpen [52], maar dat zou ook uitgebreid kunnen worden naar assistentie bij het begrijpen van heel hun reële omgeving. De potentiële toepassingen voor algoritmes die tekst kunnen genereren, en vooral voor algoritmes die tekst kunnen genereren terwijl ze geconditioneerd zijn op invoer, zijn eindeloos. Ontwikkelingen in tekstgeneratie kunnen zorgen voor vooruitgang in toepassingen zoals image captioning, vertaalsystemen, dialoogsystemen en vele andere toepassingen [93, 85, 94]. Die invoer zouden afbeeldingen kunnen zijn, antwoorden van een gesprekspartner, zinnen in een andere talen of nog veel meer.

Tekstgeneratie, beeldanalyse en ook image captioning kennen door de recente opkomst van deep learning een enorme progressie [40, 26, 84, 94]. Neurale netwerken, getraind met *gradient descent* en *backpropagation*, vormen nu de standaardkeuze als machine-learning-modellen voor zowel NLP-taken als CV-taken. Ondanks de sprong voorwaarts die neurale netwerken hebben veroorzaakt zijn tekstgeneratie en ook image captioning verre van opgelost, en blijven het actieve onderzoeksgebieden met nog veel ruimte voor verbetering.

Enkele tekortkomingen werden recent opgemerkt aan de manier om taalmodellen voor tekstgeneratie te trainen, die na de superieure resultaten met neurale netwerken de norm was geworden. Deze modellen worden typisch met *teacher forcing* getraind. Dat is een techniek waarbij het model leert om telkens het juiste volgende woord te voorspellen, gegeven het vorige *ground-truth*-woord. De *cross-entropy loss* tussen het voorspelde volgende woord en het ground-truth volgende woord wordt gebruikt als lossfunctie, en de gradiënt voor gradient descent wordt met backpropagation door

1. INLEIDING

het netwerk gepropageerd. Onderdeel 2.2.1 beschrijft deze procedure in meer detail.

Tijdens inferentie echter zijn de ground-truth-woorden niet vorhanden en voor-spelt het model bijgevolg telkens het volgende woord op basis van het vorige woord dat het model zelf voorspelde. Deze discrepantie werd *exposure bias* genoemd [8, 65]. Daarenboven worden modellen voor tekstgeneratie vaak geëvalueerd op het voorkomen van groepen van woorden in gegenereerde zinnen. De gebruikte evaluatie-functies zijn niet differentieerbaar: woorden komen een geheel aantal keer voor of niet, dus de functies zijn discreet. Daarom kunnen ze niet zonder meer als lossfunctie voor gradient descent en backpropagation gebruikt worden, en daarom wordt de cross-entropy gebruikt. Dat creeert een tweede discrepantie: trainen voor een andere lossfunctie dan de uiteindelijk gebruikte evaluatiefunctie.

Ranzato et al. stelden als remedie voor om tekstgeneratie te formuleren als taak zoals in *reinforcement learning* [65]. Zo kunnen de modellen getraind worden om hele zinnen te genereren en kunnen, met behulp van *policy-gradient*-methoden, er toch gradiënten voor gradient descent en backpropagation berekend worden uit waarden van de niet-differentieerbare evaluatiefuncties. Ze verbeteren resultaten op verschillende taken, wat bevestigt dat de twee discrepanties effectief nadelig zijn voor de performantie van algoritmes. Hun algoritme veroorzaakt een golf van gelijkaardige methoden toegepast op uiteenlopende taken met tekstgeneratie, en veel nieuwe state-of-the-art resultaten [70, 64, 50, 48, 99]. In sectie 2.2.3 in de literatuurstudie wordt dieper ingegaan op reinforcement learning voor tekstgeneratie.

Neuro-evolutie, of het gebruik van evolutionaire algoritmes (EA) in deep learning, is een idee dat al lang bestaat [55, 21, 101]. Ondanks interessante resultaten was neuro-evolutie, door beperkte rekenkracht en snel toenemende complexiteit bij het evolueren van grote en diepe neurale netwerken, lang gelimiteerd tot kleine netwerken, zeker naar hedendaagse normen [32, 82, 58, 59]. Evolutionaire algoritmes bezitten nochtans aantrekkelijke eigenschappen waardoor ze, mits mitigatie van de vernoemde beperking, nuttige optimalisatiealgoritmes voor deep learning zouden kunnen zijn.

De eerste eigenschap zet neuro-evolutie in een rechte lijn tegenover gradient descent en backpropagation, op vlak van executie en hardware. De laatste twee zijn inherent sequentiell en worden doorgaans uitgevoerd op zeer gespecialiseerde hardware: krachtige *vector processing units* zoals GPU's en TPU's. De voornaamste groei in rekenkracht in recente jaren heeft echter plaatsgevonden in parallelisatie. Door hun inherent sequentiële aard kunnen gradient descent en backpropagation minder voordeel halen uit parallelisatie. In neuro-evolutie daarentegen bestaat het meest rekenintensieve werk in elke iteratie uit het uitvoeren van een groot aantal, van elkaar onafhankelijke evaluaties. Deze algoritmes lenen zich dus van nature uit uitstekend voor parallelle executie.

De volgende gunstige eigenschap is dat evolutionaire algoritmes weinig technische voorwaarden opleggen aan de te optimaliseren doelfunctie. Het enige dat een EA nodig heeft zijn evaluaties van de doelfunctie in bepaalde punten in de zoekruimte. Reinforcement-learning-taken worden vaak gekenmerkt door ijle, stochastische of niet-voorspelbare beloningen waaruit agenten moeten leren. Zodoende werd (en wordt) neuro-evolutie vaak gebruikt voor reinforcement learning [32, 29, 82, 81, 47].

Bovendien bleek uit recent onderzoek dat, met tegenwoordig beschikbare rekenkracht, het toch mogelijk is om grotere neurale netwerken te trainen met EA's [75, 83]. Netwerken waarmee state-of-the-art resultaten in RL waren gehaald met algoritmes die gradient descent en backpropagation gebruiken werden getraind voor RL-taken met neuro-evolutie en veel resultaten werden op deze manier zelfs overtroffen. In sectie 2.3 wordt neuro-evolutie en ook het vermelde recente onderzoek in meer detail besproken.

Deze thesis zal onderzoeken of neuro-evolutie gebruikt kan worden om neurale netwerken te trainen voor tekstgeneratie, en of het daarbij een meerwaarde kan bieden tegenover bestaande methoden. Vervolgens wordt ook de vraag hoe algoritmes voor neuro-evolutie aangepast kunnen worden voor tekstgeneratie onderzocht.

De bijdrage van deze thesis is enerzijds geïnspireerd op het succes van het gebruik van reinforcement learning in tekstgeneratie, en anderzijds op het succes van neuro-evolutie voor reinforcement learning. Tekstgeneratie formuleren als RL-probleem kan de twee vermelde discrepanties overkomen. Vervolgens netwerken voor deze formulering trainen met neuro-evolutie in plaats van policy gradients kan een alternatieve, aanvullende methode zijn voor de huidige standaardprocedure; namelijk policy gradients, gradient descent en backpropagation. Recent onderzoek heeft aangetoond dat EA's wel degelijk in staat zijn om netwerken van hedendaagse grootte te trainen, wat de praktische haalbaarheid van de algoritmes in deze thesis valideert.

Meer specifiek stelt deze thesis in hoofdstuk 3 twee algoritmes voor om de parameters van LSTM-netwerken te zoeken die afbeeldingen ondertitelen [31]. Beide algoritmes lossen tekstgeneratie op als reinforcement-learning-probleem. NIC-ES is een evolutiestrategie gebaseerd op het algoritme waar Such et al. goede resultaten mee haalden op RL-taken [83]. NIC-NES is een natuurlijke evolutiestrategie gebaseerd op het algoritme waar Salimans et al. goede resultaten mee haalden [75]. Beide algoritmes optimaliseren voor de CIDEr-score, een niet-differentieerbare score die kandidaatzinnen beoordeelt op basis van het voorkomen van woordgroepen [90]. Ook worden netwerken door beide algoritmes getraind om hele zinnen te genereren en niet individuele woorden. Daardoor lijden NIC-ES en NIC-NES, zoals het algoritme van Ranzato et al., niet onder de besproken discrepanties. NIC-ES en NIC-NES kunnen op CPU's uitgevoerd worden en zijn zeer paralleliseerbaar. In de experimenten werden tot 96 vCPU's gebruikt, maar nog veel meer CPU's zouden gebruikt kunnen worden.

Naast de algoritmes zelf worden ook enkele veiligere varianten op de klassieke mutatie bij EA's voorgesteld. Een van die mutaties werd gebruikt door Lehman et al. [46], en wordt in deze thesis aangepast voor taalmodellen. Ook verschillende evaluatielijnen, allen gebaseerd op de CIDEr-score, worden voorgesteld. NIC-ES kan gebruik maken van twee verschillende selectiemechanismen, en NIC-NES kan evaluaties uitvoeren op gelijke of verschillende minibatches.

Hoofdstuk 4 beschrijft de experimenten en de resultaten. NIC-ES en NIC-NES worden vergeleken met teacher forcing en met *self-critical sequence training*

1. INLEIDING

(SCST), een policy-gradient-methode [70]. Ook de verschillende evaluatiefuncties en mutatievarianten, alsook de andere vermelde instellingen, worden vergeleken. De resultaten worden ook geïnterpreteerd in hoofdstuk 4 en op basis van de bevindingen worden er suggesties gegeven voor verder onderzoek.

Deze thesis is een verkenning van het gebruik van evolutionaire algoritmen voor het trainen van neurale netwerken voor tekstgeneratie. Daarbij mag het woord “verkenning” benadrukt worden: er werd naast de experimenten van Miikkulainen et al. [52] geen recent onderzoek gevonden waarin neuro-evolutie op deze schaal werd toegepast op tekstgeneratie met neurale netwerken. Moest blijken dat NIC-ES en NIC-NES goed werken voor image captioning, kan dat bijdragen aan de heropleving die EA’s in deep learning op dit moment kennen [83, 75, 52, 67], en kan dat specifiek de deur openen voor EA’s in tekstgeneratie.

Het feit dat gradient descent via backpropagation op GPU’s als norm wordt gezien heeft natuurlijk een reden: namelijk dat het goed werkt. Dat blijkt ook uit de resultaten van de experimenten, waar SCST de grootste verbetering geeft. Neuro-evolutie zou echter wel meer in het licht kunnen komen als alternatief, dat heel andere hardware gebruikt en dus in andere situaties bruikbaar is. Dat kan het draagvlak voor deep learning vergroten: deep learning op parallelle CPU’s, terwijl het nu onafscheidelijk gekoppeld is aan training op GPU’s.

De tekst begint met een literatuurstudie. Eerst wordt een achtergrond in machine learning en deep learning gegeven. Dan worden NLP en tekstgeneratie ingeleid, samen met gebruikte modellen en teacher forcing. Het ondertitelen van afbeeldingen en het gebruik van policy gradients in tekstgeneratie worden toegelicht. Dan volgt een bespreking van neuro-evolutie: eerst evolutionaire algoritmes in het algemeen, dan EA’s in deep learning en vervolgens een bespreking van de algoritmes waar NIC-ES en NIC-NES op gebaseerd zijn.

In hoofdstuk 3 worden NIC-ES en NIC-NES uitgelegd, met ontwerpbeslissingen en hun motivering. Hoofdstuk 4 vervolgt met de experimenten: de voorbereiding en praktische overwegingen, een overzicht en vergelijking met teacher forcing en SCST, en experimenten die inzoomen op deelaspecten. Daaropvolgend worden suggesties voor verder onderzoek gegeven. Tot slot volgt in hoofdstuk 5 de conclusie van de thesis.

Hoofdstuk 2

Literatuurstudie

In dit hoofdstuk worden ander onderzoek, concepten en bevindingen besproken die relevant zijn voor deze thesis. Eerst beschrijft het hoofdstuk kort de algemene achtergrond van deep learning: probleemstelling, types modellen en het trainen. Basisideeën en begrippen worden uitgebreid ingeleid. Een lezer die reeds vertrouwd is met de nodige basisconcepten kan dus zonder probleem delen overslaan. Een lezer met een achtergrond in machine learning kan bijvoorbeeld beginnen bij sectie 2.2.

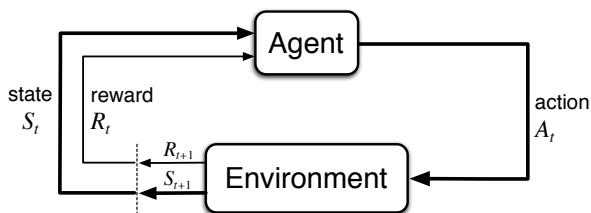
In sectie 2.2 beschrijft de literatuurstudie Natural Language Processing, met de representatie van data en de gebruikte modellen. Tekstgeneratie wordt gedefinieerd en afwijkingen tussen trainen en inferentie worden uitgelegd, samen met reeds voorgestelde oplossingen. Gezien deze thesis evolutionaire algoritmes voorstelt om een neuraal netwerk te trainen voor het ondertitelen van afbeeldingen, worden daarna deze toepassing (afbeeldingen ondertitelen) en reeds voorgestelde modellen besproken. Vervolgens worden enkele recente succesvolle methoden die tekstgeneratie als reinforcement-learning-probleem formuleren toegelicht. Tot slot worden evolutionaire algoritmes en toepassingen ervan in deep learning behandeld. Verder bouwend op het voorgaande over reinforcement learning legt de tekst uit waarom EA's interessant kunnen zijn voor tekstgeneratie.

2.1 Machine learning en deep learning

Als achtergrond worden in deze sectie beknopt de nodige concepten uit machine learning en deep learning herhaald.

Een “machine-learning-algoritme is een algoritme dat in staat is om te leren uit data” [24]. Hierbij is het de bedoeling dat de performantie van het algoritme, gemeten met een gekozen evaluatiemetriciek, stijgt met het aantal datapunten dat het algoritme verwerkt. Elk datapunt is een collectie van *features*, kwantitatief gemeten kenmerken van een datapunt, en kan typisch voorgesteld worden als vector of een matrix. Een verzameling van datapunten heet een dataset. De soort data en de wijze waarop die beschikbaar is voor het algoritme groeperen machine learning (verder ML)

2. LITERATUURSTUDIE



Figuur 2.1: Reinforcement learning

De interactie tussen agent en omgeving in een reinforcement-learning-probleem [86].

in paradigma's. De voor deze thesis relevante paradigma's zijn *supervised learning* (verder SL) en reinforcement learning (verder RL).

Supervised learning. Naast features \mathbf{x} hoort bij elk datapunt ook een label \mathbf{y} . Dat label kan eender welke vorm aannemen, zoals een categorie (classificatie) of numerieke waarden (regressie). Een supervised learning algoritme leert om uit de features het label te voorspellen, door de kansverdeling $p(\mathbf{y} \mid \mathbf{x})$ te benaderen [24]. De verhouding van correct en foutief voorspelde labels, of nauwkeurigheid, is een eenvoudige manier om zo'n algoritme kwantitatief te evalueren. Het doel van een supervised learning algoritme is het halen van een hoge nauwkeurigheid. Een algoritme met een goede *generalisatie* haalt een hoge nauwkeurigheid niet enkel bij invoer van datapunten \mathbf{x} die in de beschikbare dataset voorkomen, maar ook bij ongeziene datapunten.

Reinforcement learning. RL is begaan met het maximaliseren van beloningen die een agent in een omgeving waarneemt, door te zoeken naar een optimale *policy*-functie $\pi(a|s)$ die de acties a van de agent in toestand s stuurt. De agent interageert met de omgeving in de tijdstappen: hij voert acties uit die al dan niet een effect hebben op de omgeving en observeert (een deel van) de omgeving en eventuele beloningen [86]. Zie figuur 2.1 voor een overzicht van de interacties tussen agent en omgeving. Hier krijgt het algoritme dus geen volledig en onmiddellijk beschikbare dataset van datapunten met labels, maar krijgt het data in de vorm van ervaringen of interacties met de omgeving.

2.1.1 Parametrische modellen en neurale netwerken

Neem als invoer vector \mathbf{x} en als uitvoer vector \mathbf{y} . Een model beschrijft een set van functies met als signatuur $\mathbf{x} \mapsto \mathbf{y}$: de hypothese-ruimte. Bij een machine-learning-probleem zoals in deze thesis wordt voor een gegeven model (of type van model) zo een functie $f : \mathbf{x} \mapsto \mathbf{y}$, die voldoet aan bepaalde voorwaarden, gezocht of benaderd. De gezochte functie f kan onder andere een policyfunctie $\pi : s \mapsto a$ zijn die beloningen maximaliseert zoals in RL, of een functie $f : \mathbf{x} \mapsto \mathbf{y}$ die invoer afbeeldt op labels met een zo hoog mogelijke nauwkeurigheid zoals in SL.

Bij parametrische modellen is f afhankelijk van parameters θ : $\mathbf{y} = f(\mathbf{x}; \theta)$. Het model trainen komt neer op de beschikbare data gebruiken om de beste parameters θ te zoeken.

Neurale netwerken. De parametrische modellen die deze thesis beschouwt heten neurale netwerken (verder NNs). Een neuraal netwerk is in essentie een compositie van eenvoudigere functies, met als een mogelijke representatie een gerichte graaf die beschrijft hoe de functies samengesteld zijn. Een *feedforward neural network* definieert, zoals hierboven, een afbeelding $\mathbf{y} = f(\mathbf{x}; \theta)$, verder ook f_θ . De naam *feedforward* volgt uit het feit dat informatie van \mathbf{x} naar \mathbf{y} in één richting door het netwerk stroomt: de gerichte graaf is acyclisch. De uitvoer $\mathbf{y} = f(\mathbf{x}; \theta)$ eenmaal berekenen wordt ook een *forward pass* door het netwerk genoemd.

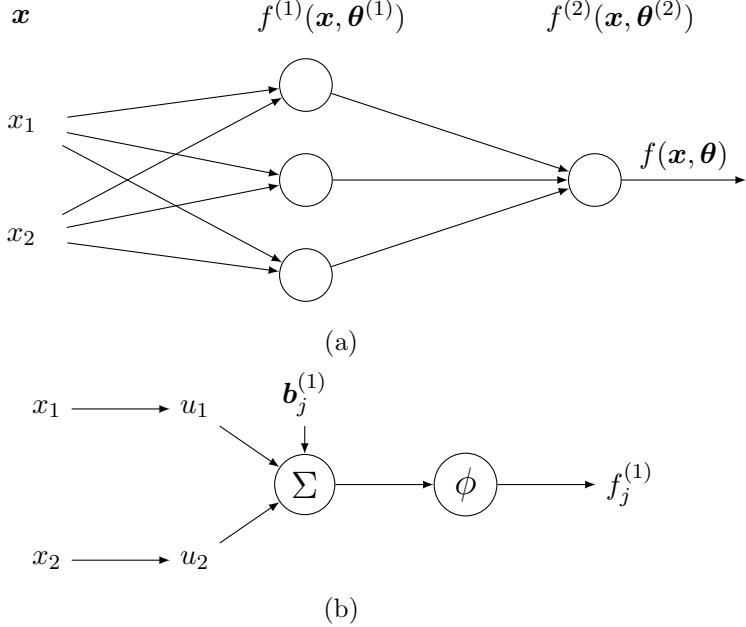
Stel dat $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$, dan worden $f^{(1)}$, $f^{(2)}$ en $f^{(3)}$ respectievelijk de eerste, tweede en derde laag van het netwerk genoemd. Het aantal lagen bepaalt de diepte van het netwerk. Lagen die geen invoer- of uitvoerlaag zijn worden verborgen of *hidden* lagen genoemd. De lagen van een NN hebben typisch vectoren als waarden, de dimensie van zo een vector bepaalt de breedte van een laag. In plaats van als functie $f^{(i)} : \mathbb{R}^n \mapsto \mathbb{R}^m$ kan een laag ook gezien worden als verzameling van m knopen of neuronen die in parallel elk een functie $f_j^{(i)} : \mathbb{R}^n \mapsto \mathbb{R}$ definiëren, met $1 < j < m$. Opdat de hypotheseruimte van de modellen ook niet-lineaire functies omvat moet ten minste één van de lagen een niet-lineaire functie implementeren [24]. Figuur 2.2 toont een neuraal netwerk, voorgesteld als gerichte graaf, dat de functie f implementeert:

$$\begin{aligned} f(\mathbf{x}; \theta) &= f^{(2)}(f^{(1)}(\mathbf{x}, \theta^{(1)}), \theta^{(2)}) \\ f^{(2)}(\mathbf{x}, \theta^{(2)}) &= (\mathbf{w}^{(2)})^T \mathbf{x} + b^{(2)} \\ f^{(1)}(\mathbf{x}, \theta^{(1)}) &= \phi(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}) \end{aligned} \tag{2.1}$$

Met $\theta = (\theta^{(1)}, \theta^{(2)}) = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{w}^{(2)}, b^{(2)})$ en $\phi(\mathbf{x})$ een niet-lineaire activatiefunctie. Wanneer opeenvolgende lagen bestaan uit neuronen die invoer krijgen van alle neuronen uit de laag ervoor worden de lagen volledig verbonden genoemd (of lineair wanneer ze geen ϕ bevatten). De in de praktijk gebruikte neurale netwerken zijn doorgaans veel dieper en hebben miljoenen parameters [30]. De term “deep learning” is een samentrekking van machine learning en diepe neurale netwerken, i.e. met veel lagen.

Recurrent Neural Networks. Recurrent neural networks [20] (verder RNN’s) zijn neurale netwerken die “gespecialiseerd zijn voor het verwerken van sequenties van waarden \mathbf{x}_t ” [24], waarbij $t = 1, \dots, T$ de index van vector $\mathbf{x}^{(t)}$ in de sequentie is. Het zijn netwerken waarvan de uitvoer in t niet enkel afhankelijk is van de invoer in t maar ook van een toestandsvector uit $t-1$. Het type RNN in deze thesis genereert uit een invoersequentie van vectoren $(\mathbf{x}_1, \dots, \mathbf{x}_T)$ zowel een uitvoersequentie $(\mathbf{y}_1, \dots, \mathbf{y}_T)$ als een *hidden-state*-sequentie $(\mathbf{h}_1, \dots, \mathbf{h}_T)$ door het herhaaldelijk toepassen van een functie $f : \mathbb{R}^k \times \mathbb{R}^n \mapsto \mathbb{R}^k \times \mathbb{R}^m$ met $k = \dim(\mathbf{h})$, $n = \dim(\mathbf{x})$, $m = \dim(\mathbf{y})$ [84, 26,

2. LITERATUURSTUDIE



Figuur 2.2: Een feedforward neuraal netwerk

Een volledig verbonden feedforward neuraal netwerk dat de functie f uit (2.1) implementeert. Figuur 2.2a toont een overzicht van een compositie van functies voorgesteld als gerichte graaf, met tweedimensionale invoer \mathbf{x} , laag $f^{(1)}$ met breedte drie en ééndimensionale uitvoerlaag $f^{(2)}$. Figuur 2.2b toont één van de drie knopen uit laag $f^{(1)}$, met $u_2 = \mathbf{W}_{j,2}^{(1)}$, $u_1 = \mathbf{W}_{j,1}^{(1)}$.

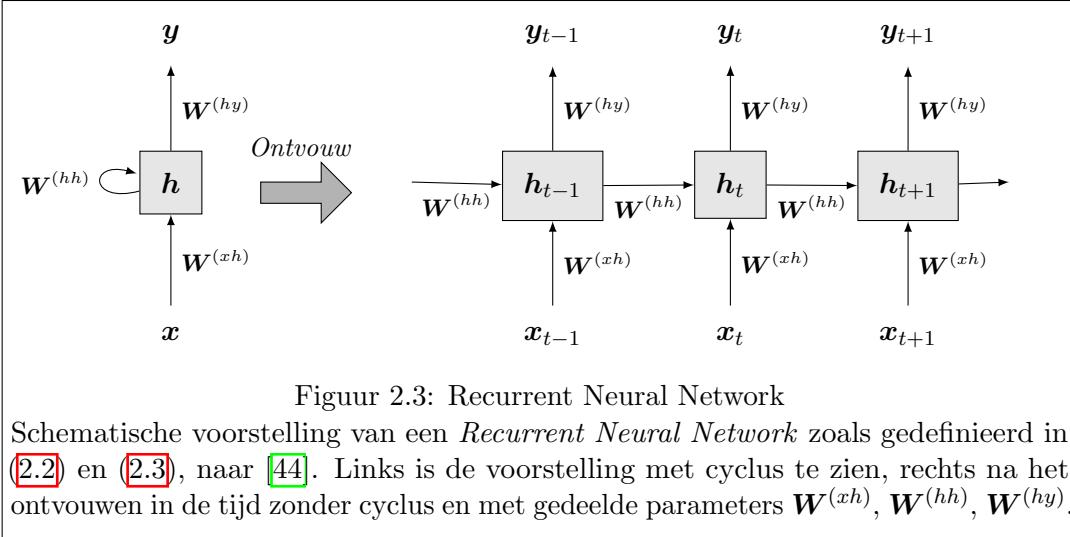
[27]:

$$\mathbf{h}_t, \mathbf{y}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta}), \quad (2.2)$$

Of als iets specifiekere veel voorkomende compositie:

$$\begin{aligned} \mathbf{h}_t &= f_h(\mathbf{W}^{(hh)} \mathbf{h}_{t-1} + \mathbf{W}^{(xh)} \mathbf{x}_t) \\ \mathbf{y}_t &= f_y(\mathbf{W}^{(hy)} \mathbf{h}_t). \end{aligned} \quad (2.3)$$

Als invoer in elke tijdstap krijgt het RNN niet enkel \mathbf{x}_t , maar ook de verborgen toestandsvector \mathbf{h}_{t-1} uit de vorige tijdstap. Dat laat het netwerk toe informatie uit het verleden te gebruiken bij het berekenen van uitvoer en dat maakt dit type model zo geschikt voor het verwerken van sequenties. Er zijn variaties op RNN's zoals hier gedefinieerd, bv. RNN's die enkel in de laatste tijdstap een uitvoer \mathbf{y}_t berekenen. Het recursief voorkomen van \mathbf{h} in definitie (2.2) leidt tot een cyclus in de computationele graaf van een netwerk dat een functie f implementeert [24]. De linkerkant van figuur 2.3 toont een voorstelling van een RNN met wederkerende connectie. Deze cyclus *ontvouwen* in de tijd (over heel de sequentie) levert een



Figuur 2.3: Recurrent Neural Network

Schematische voorstelling van een *Recurrent Neural Network* zoals gedefinieerd in (2.2) en (2.3), naar [44]. Links is de voorstelling met cyclus te zien, rechts na het ontvouwen in de tijd zonder cyclus en met gedeelde parameters $\mathbf{W}^{(xh)}$, $\mathbf{W}^{(hh)}$, $\mathbf{W}^{(hy)}$.

voorstelling van hetzelfde netwerk op zonder cyclussen. Dit komt overeen met in vergelijking (2.2) h_{t-1} vervangen door zijn definitie in functie van h_{t-2} , enz. tot het begin van de sequentie: vector h_1 die geïnitialiseerd wordt met een vaste waarde. Dit geeft uiteindelijk een expressie zonder recursie [24]. De rechterkant van figuur 2.3 toont deze voorstelling als acyclisch netwerk waarin parameters $\mathbf{W}^{(xh)}$, $\mathbf{W}^{(hh)}$, $\mathbf{W}^{(hy)}$ gedeeld worden over de tijdstappen.

Long Short-Term Memory netwerken (LSTM's) [31] zijn een type van RNN's die door het gebruik van een architectuur met *input*, *output*- en *forget-gates* de terugwaartse propagatie van gradiënten bij het trainen stabiliseren [27, 103, 10]. De concrete werking van LSTM's valt buiten het bestek van deze tekst.

2.1.2 Loss functies en gradient descent

Nu de modellen gedefinieerd zijn, kunnen hun parameters $\boldsymbol{\theta}$ met verschillende methoden gezocht worden. In deze thesis wordt voorgesteld om dat te doen met evolutionaire algoritmes. De huidige standaardmethode in deep learning is gradient descent. Daarom behandelt dit onderdeel gradient descent kort. Concepten zullen uitgelegd worden zoals ze van toepassing zijn in een supervised learning probleem met een beschikbare dataset \mathcal{D} van (\mathbf{x}, \mathbf{y}) -paren.

Loss functie. Zoals eerder vermeld komt het trainen van een parametrisch model neer op het zoeken van de optimale parameters $\boldsymbol{\theta}$, aan de hand van \mathcal{D} . Om de notie van goede parameters te formaliseren is de *lossfunctie* nodig: een gekozen functie $L : \mathbb{R}^m \times \mathbb{R}^m \mapsto \mathbb{R}$ die voor een datapunt (\mathbf{x}, \mathbf{y}) uit \mathcal{D} de uitvoer van het model $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$ en het doel-label \mathbf{y} afbeeldt op een getal dat aangeeft hoe gelijk \mathbf{y} en $\hat{\mathbf{y}}$

2. LITERATUURSTUDIE

zijn. Als $L(\hat{\mathbf{y}}, \mathbf{y})$ de loss is voor één datapunt uit \mathcal{D} , dan kan:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}_{(x,y) \in \mathcal{D}} [L(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y})] \\ &= \frac{1}{D} \sum_{i=1}^D L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)}) \end{aligned} \quad (2.4)$$

genomen worden als maat voor hoe goed de huidige $f(\mathbf{x}; \boldsymbol{\theta})$ is. Hierbij is D het aantal datapunten in \mathcal{D} . Zo wordt het trainen van een model geformuleerd als optimalisatieprobleem: de gezochte $\boldsymbol{\theta}^*$ zijn de parameters die de *training loss* $J(\boldsymbol{\theta})$ minimaliseren:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \quad (2.5)$$

Generalisatie. Het doel van een supervised learning algoritme is een verdeling p_{real} benaderen en daarmee

$$J_{real}(\boldsymbol{\theta}) = \mathbb{E}_{(x,y) \sim p_{real}} [L(\hat{\mathbf{y}}, \mathbf{y})] \quad (2.6)$$

minimaliseren, voor (\mathbf{x}, \mathbf{y}) uit de echte dataverdeling p_{real} . Omdat er typisch geen mogelijkheid is tot ongelimiteerd samplen uit p_{real} , wordt in de plaats p_{real} benaderd met p_{data} en daarmee de empirische lossfunctie

$$J_{empirical}(\boldsymbol{\theta}) = \mathbb{E}_{(x,y) \in \mathcal{D}} [L(\hat{\mathbf{y}}, \mathbf{y})] \quad (2.7)$$

geminimaliseerd. Daarbij zijn $(\mathbf{x}, \mathbf{y}) \sim p_{data}$, de kansverdeling gedefinieerd door de dataset, en niet uit p_{real} [15]. Een model dat goed generaliseert, heeft niet alleen een lage waarde voor $J_{empirical}$ maar ook voor J_{real} . Een voorwaarde hiervoor is natuurlijk dat p_{data} en p_{real} goed genoeg overeenkomen, met als ideale geval dat ze identiek zijn.

Maximum likelihood. Voor de keuze van een specifieke lossfunctie in SL kan worden gebruik gemaakt van het principe van *maximum likelihood estimation* (MLE). SL werd gedefinieerd als het benaderen van een conditionele kansverdeling $p_{data}(\mathbf{y} | \mathbf{x})$, d.w.z. de kansverdeling over labels \mathbf{y} gegeven \mathbf{x} uit dataset \mathcal{D} . Gezien het parametrische modellen betreft die p_{data} benaderen, is de kansverdeling die het model definieert ook afhankelijk van parameters: $p_{model}(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta})$. Een kanttekening hierbij is dat eender welke functie f_{θ} van een parametrisch model een kansverdeling p_{θ} kan definiëren, bijvoorbeeld door de softmax te nemen van de uitvoerlaag van f_{θ} . Neem \mathbf{X} en \mathbf{Y} alle datapunten en labels uit \mathcal{D} . De maximum-likelihood-schatter voor $\boldsymbol{\theta}$ is dan de waarde die p_{model} zo parametriseert dat de kans op exact \mathbf{Y} van p_{model} samplen gegeven \mathbf{X} maximaal is [24]:

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg \max_{\boldsymbol{\theta}} p_{model}(\mathbf{Y} | \mathbf{X}; \boldsymbol{\theta}) \\ &= \arg \max_{\boldsymbol{\theta}} \mathbb{E}_{(x,y) \in \mathcal{D}} [\log p_{model}(\mathbf{y}^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta})] \\ &= \arg \min_{\boldsymbol{\theta}} H(p_{data}, p_{model}) \end{aligned} \quad (2.8)$$

Met andere woorden: als $J(\boldsymbol{\theta}) = H(p_{data}, p_{model})$ uit vergelijking (2.5), met $H(p_{data}, p_{model})$ de cross-entropy (verder XENT) tussen kansverdelingen p_{data} en p_{model} , dan is $\boldsymbol{\theta}^*$ de maximum-likelihood-schatter voor $\boldsymbol{\theta}$. Dat motiveert de keuze van de cross-entropy tussen de twee verdelingen als lossfunctie voor veel supervised learning algoritmes [24].

Gradient descent. De voor de hand liggende wiskundige methode voor het berekenen van parameter(s) die een functie minimaliseren is berekenen waar de gradiënt gelijk is aan nul:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= 0 \\ \Leftrightarrow \boldsymbol{\theta} &= \dots\end{aligned}\tag{2.9}$$

Dit stelsel van niet-lineaire vergelijkingen met tot miljoenen onbekenden in functie van tot miljoenen datapunten is echter in het algemeen niet analytisch oplosbaar [15]. Daarom wordt er beroep gedaan op numerieke iteratieve optimalisatiemethoden.

De standaard optimalisatiemethode in deep learning is gradient descent [24]. Gradient descent houdt in het iteratief aanpassen van de parameters in de richting van de negatieve gradiënt van de lossfunctie ten opzichte van de parameters:

$$\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}).\tag{2.10}$$

Hierbij is η de *learning rate*, een hyperparameter. De negatieve gradiënt geeft de richting van de steilste afval aan; elke iteratie past $\boldsymbol{\theta}$ dus aan met een stap in de richting waarin $J(\boldsymbol{\theta})$ het sterkst daalt. Het aantal datapunten D en dus het aantal termen in $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is doorgaans te groot om bij elke update van $\boldsymbol{\theta}$ de hele dataset te gebruiken. Daarom wordt er een *minibatch* \mathcal{B} met een kleinere grootte B stochastisch gesampled uit \mathcal{D} en wordt hiermee een schatter van de gradiënt berekend [24]:

$$\begin{aligned}\hat{\mathbf{g}} &= \frac{1}{B} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^B L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)}) \\ \boldsymbol{\theta}_{i+1} &\leftarrow \boldsymbol{\theta}_i - \eta \hat{\mathbf{g}}.\end{aligned}\tag{2.11}$$

Het resulterende proces heet *stochastic gradient descent* (SGD). Indien de lossfunctie de cross-entropy loss is, zoals in de vorige paragraaf gedefinieerd, dan is (2.11) equivalent aan:

$$\hat{\mathbf{g}} = \frac{1}{B} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^B \mathbf{y}^{(i)} \log f(\mathbf{x}^{(i)}; \boldsymbol{\theta})\tag{2.12}$$

De gradiënt $\hat{\mathbf{g}}$ wordt in de praktijk voor neurale netwerken analytisch en op efficiënte wijze berekend via *backpropagation* [73]. Die methode steunt op het achterwaarts recursief toepassen van de kettingregel voor gradiënten, te beginnen bij

2. LITERATUURSTUDIE

de gradiënt van de lossfunctie t.o.v. de uitvoer van het netwerk, en zo verder tot de eerste laag. Het op die manier achterwaarts door het netwerk propageren van de gradiënt heet een *backward pass* door het netwerk, en is in de praktijk een dure operatie [75]. Hoewel ze samen de standaardmethode voor training in deep learning vormen zijn gradient descent en backpropagation niet gelijk. Gradient descent slaat op het principe waarbij θ iteratief geüpdateert wordt in de richting van $-\nabla_{\theta}J(\theta)$, terwijl backpropagation een efficiënte manier is om de analytische berekening van gradiënten in neurale netwerken te implementeren.

De ontvouwen representatie van RNN's is relevant omdat in deze voorstelling de gradiënt op exact dezelfde wijze "door de tijd" kan teruggepropageerd worden door de hidden-naar-hidden connecties als in een feedforward neuraal netwerk. Deze techniek heet *backpropagation through time* [95]. Figuur 2.4 toont de achterwaartse propagatie van de gradiënt door een ontvouwen RNN.

2.2 Natural Language Processing

Natural Language Processing of NLP omvat een reeks computationale technieken voor het automatisch verwerken, analyseren, genereren en representeren van menselijke taal. Voorbeelden van actuele toepassingen zijn systemen die automatisch vertalen, zinnen grammaticaal ontleden, dialoogsystemen, sentimentanalyse, samenvatten van tekst, enzoverder [103]. Ook het ondertitelen van afbeeldingen valt deels binnen NLP omdat een deel van de taak tekstgeneratie inhoudt.

In deze sectie zal tekst gezien worden als sequenties van woorden maar alle concepten zijn evenzeer van toepassing op sequenties van symbolen of zelfs zinnen.

Woordrepresentaties. Tot recent maakten machine-learning-algoritmes in NLP gebruik van hoogdimensionale en ijle features. Neem een woordenschat \mathcal{V} , i.e. de verzameling van alle woorden die het model kan gebruiken, van grootte V . De meest voor de hand liggende codering van woorden is een *one-hot encoding*: elk woord uit de woordenschat wordt voorgesteld door een vector van dimensie V waarvan alle elementen gelijk zijn aan nul, buiten één element dat gelijk is aan één. Omdat V vaak groot is [85] moeten taalmodellen leren werken in een zeer hoogdimensionale en ijle discrete ruimte, wat o.a. door de *curse of dimensionality* [9] moeilijk is.

Maar sinds de intrede van *word embeddings* behalen neurale netwerken die de compactere vectorrepresentaties gebruiken superieure resultaten [103]. Word embeddings zijn vectoren met dimensie $d_e << V$ van reële getallen die woorden representeren [9]. Ze worden vaak gebruikt als eerste data-verwerkingslaag in neurale netwerken: een laag die one-hot encoderingen van woorden omzet in continue vectorrepresentaties. Die lagen worden doorgaans vooraf getraind voor een hulpprobleem, zoals in het bekende "word2vec" van Mikolov et al. [53, 54, 72].

Modellen. Voor NLP-toepassingen zijn reeds veel verschillende modellen gebruikt. Zo zijn er bv. toepassingen die *convolutional neural networks* (CNN's) gebruiken om features van een hoog abstractieniveau te extraheren uit invoersequenties van

word embeddings. Die features kunnen dan voor een waaier van NLP-taken gebruikt worden, zoals in [36, 34]. Voor veel NLP-taken blijken RNN's echter bijzonder krachtige modellen [35, 94, 85, 84, 13, 5], wat de volgende punten kunnen verklaren.

- Een eerste duidelijke reden is de inherent sequentiële aard van data in NLP: tekst als sequenties van woorden. Hetzelfde geldt voor sequenties van zinnen of symbolen. Door het sequentiële verwerken van invoer kunnen RNN's de afhankelijkheid van woorden van hun context modelleren. RNN's vertonen, zoals vermeld in sectie 2.1.1, een soort geheugen waardoor ze verbanden tussen invoerwoorden op uiteenliggende locaties in sequenties kunnen modelleren.
- Verder leggen RNN's geen restricties op wat betreft de lengte van invoer- of uitvoersequenties. De invoerlengte hoeft niet gekend te zijn en kan variëren ook binnen één toepassing. De verborgen toestandsvector \mathbf{h}_t encodeert in tijdstap t informatie over een variabel aantal ($1 \leq t \leq T$) invoerwoorden in een vector van vaste dimensie k [103]. Deze vector kan dan bv. zoals in [85, 13] als invoer voor een decoder dienen die een uitvoersequentie genereert.

RNN's zijn dus als sequentiieverwerkers zeer geschikt voor veel NLP-taken. Dat wil niet zeggen dat ze het enige gebruikte type modellen zijn: recent zijn interessante resultaten bekomen met CNN's voor taken waarvoor om bovenstaande redenen vaker RNN's worden gebruikt [22, 16]. Op CNN's gaat deze literatuurstudie niet verder in, zie o.a. [24, 40] voor details. Een ander mechanisme dat voor een sprong voorwaarts in resultaten in NLP heeft gezorgd is *attention*. Attention werd voor het eerst geïntroduceerd voor automatisch vertalen in [6] en heeft ondertussen aan verbeterde resultaten bijgedragen in veel NLP-toepassingen, waaronder afbeeldingen ondertitelen [100]. In [89] halen Vaswani et al. state-of-the-art resultaten in automatisch vertalen met een model dat enkel attention-modules en volledig verbonden feedforward NNs gebruikt.

2.2.1 Tekstgeneratie

De NLP toepassing van deze thesis is het genereren van ondertitels gegeven een afbeelding met een RNN. In deze sectie zal uitgelegd worden hoe een RNN tekst kan genereren, eventueel geconditioneerd op invoer zoals een afbeelding.

Probabilistisch taalmodel

Het doel van tekstgeneratie is het benaderen van een kansverdeling over sequenties van woorden [9]

$$p_{real}(u_1, \dots, u_T) = \prod_{t=1}^T p_{real}(u_t | u_1, \dots, u_{t-1}) \quad (2.13)$$

Waarbij de factor voor $t = 1$ gelijk is aan $p_{real}(u_1)$. In supervised learning is het objectief echter altijd het voorspellen van een \mathbf{y} gegeven een \mathbf{x} . Hier slaat

2. LITERATUURSTUDIE

tekstgeneratie dus op het benaderen van een kansverdeling over sequenties van woorden gegeven invoer \mathbf{x} . Verschillende invullingen van tekstgeneratie in SL zijn denkbaar, maar de doel-labels \mathbf{y} zijn altijd sequenties van woorden (u_1, \dots, u_T) . Een woord u_t kan gezien worden als een one-hot encoding van dimensie V of als natuurlijk getal dat de index van het woord in de woordenschat aanduidt. Voor invoer \mathbf{x} wordt gelijkheid (2.13) dan:

$$p_{\text{real}}(\mathbf{y}|\mathbf{x}) = p_{\text{real}}(\mathbf{u}_{1:T}|\mathbf{x}) \quad (2.14)$$

$$= \prod_{t=1}^T p_{\text{real}}(u_t|\mathbf{x}, \mathbf{u}_{1:t-1}) \quad (2.15)$$

Het RNN benadert de factoren in (2.14), dat wil zeggen dat in elke tijdstap t de uitvoer van het netwerk een verdeling $p_{\text{model}}(u_t|\mathbf{x}, \mathbf{u}_{1:t-1}; \boldsymbol{\theta}) = p_{\text{model}}(u_t|\mathbf{h}_t; \boldsymbol{\theta})$ definieert, een benadering van $p_{\text{real}}(u_t|\mathbf{x}, \mathbf{u}_{1:t-1})$. Daarbij is de verborgen toestandsvector \mathbf{h}_t telkens een functie van de invoer \mathbf{x} en de vorige woorden $\mathbf{u}_{1:t-1}$ [8]. Concreet betekent dit dat het netwerk een vector van V kansen berekent: één voorspelde kans per woord in de woordenschat. Dat kan door de *softmax* te nemen van de uitvoer \mathbf{o}_t van de laatste laag van het netwerk [26]:

$$p_{\text{model}}(u_t = k|\mathbf{o}_t) = \frac{\exp(\mathbf{o}_t)_k}{\sum_{i=1}^V \exp(\mathbf{o}_t)_i}. \quad (2.16)$$

Een sample $\hat{u}_t \sim p_{\text{model}}$ of $u_t^* = \arg \max p_{\text{model}}$ van de categorische kansverdeling p_{model} gedefinieerd door de uitvoer van het RNN, kan dan gezien worden als een voorspelling voor het volgende woord, gegeven de invoer \mathbf{x} en de vorige verborgen toestandsvector \mathbf{h}_{t-1} . De invoer \mathbf{x} kan bijvoorbeeld een andere sequentie $(v_1, \dots, v_{T'})$ zijn zoals bij automatische vertaal- en dialoogsysteem [85, 93] of een afbeelding I [94].

De invoer \mathbf{x}_t in elke tijdstap t voor het RNN is niet per se een deel van of hetzelfde als de invoer \mathbf{x} voor het hele algoritme. Voor de duidelijkheid is in de rest van de tekst \mathbf{x} de invoer voor het gehele algoritme zoals een afbeelding I , en z_t de invoer voor het RNN in tijdstap t . Bij het genereren van tekst met RNN's is z_t gewoonlijk ofwel het vorige ground-truth-woord u_{t-1} ofwel \hat{u}_{t-1} of u_{t-1}^* van p_{model} . Er zijn verschillende manieren waarop \mathbf{x} als invoer aan het model gegeven kan worden, bijvoorbeeld door \mathbf{x} te encoderen in een vector en deze vector of een functie ervan als initiële verborgen toestandsvector \mathbf{h}_1 te gebruiken voor een RNN dat de uitvoersequentie $\hat{\mathbf{y}}$ berekent zoals in [85]. Een andere mogelijkheid is \mathbf{x} als eerste woord z_1 aan het RNN invoeren [94]. Zie figuren 2.4 en 2.5 voor voorbeelden van hoe een RNN tekst kan genereren uit \mathbf{x} .

Trainen en teacher forcing

Om een RNN te trainen in tekstgeneratie kan de maximum-likelihood-schatter van θ worden gezocht zoals in (2.8), d.w.z. met de cross-entropy als lossfunctie:

$$\begin{aligned}\theta^* &= \arg \max_{\theta} p_{model}(\mathbf{Y} \mid \mathbf{X}; \theta) \\ &= \arg \max_{\theta} \sum_{i=1}^d \sum_{t=1}^T \log p_{model}(u_t^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{u}_{1:t-1}^{(i)}; \theta).\end{aligned}\quad (2.17)$$

Het eerste sommatieteken is de som van sequenties in \mathcal{D} , het tweede de som van de woorden van een sequentie. De sequenties $\mathbf{u}^{(i)}$ zijn de ground-truth-sequenties uit de dataset. Het maximum-likelihood-principe maximaliseert dus $p_{model}(u_t^{(i)})$ geconditioneerd op de vorige ground-truth-woorden en niet op de woorden die het model zelf genereert. Tijdens het trainen om in elke tijdstap het volgende ground-truth-woord u_t te genereren, is telkens het vorige ground-truth-woord u_{t-1} de invoer voor het netwerk. Deze techniek heet *teacher forcing* [8, 24, 103, 98]. De gradiënt van de cross-entropy tussen p_{model} en p_{data} wordt voor elke t berekend en door het ontvouwen netwerk achterwaarts gepropageerd. Figuur 2.4 geeft een grafische weergave van teacher forcing en van backpropagation through time.

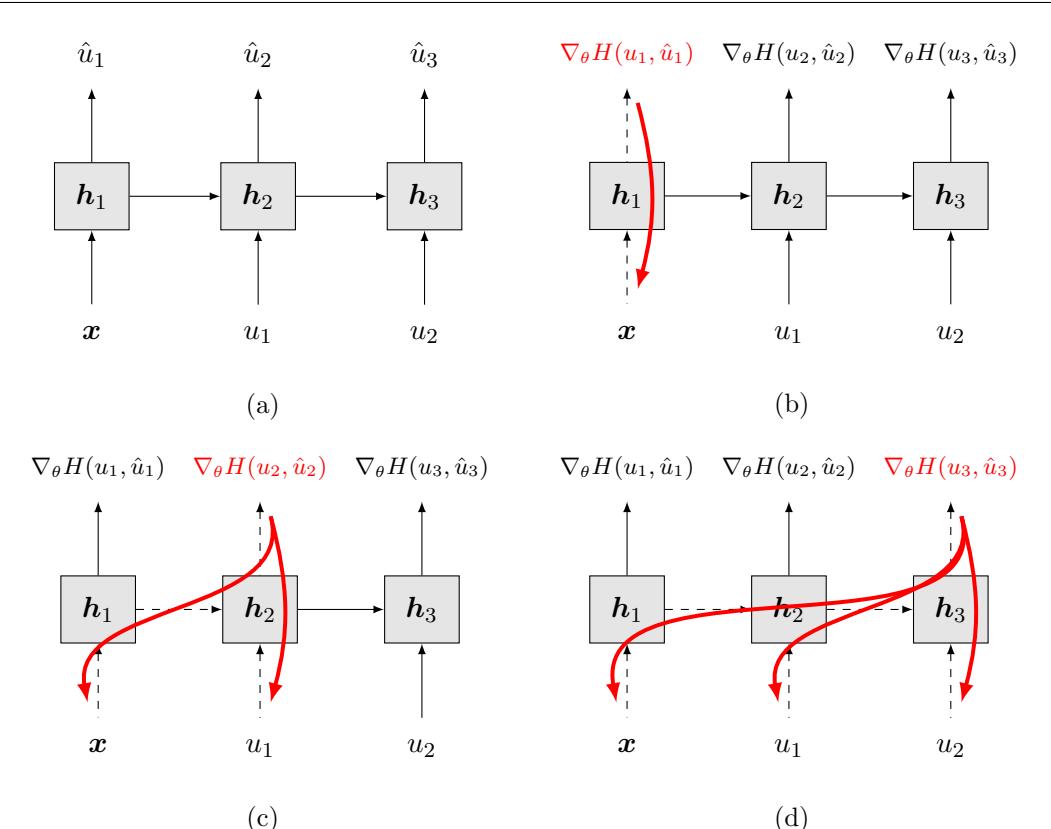
Inferentie en exposure bias

Tijdens inferentie echter, i.e. het effectief gebruiken van het getrainde model om sequenties te genereren voor \mathbf{x} die niet in \mathcal{D} voorkomen, zijn ground-truth-sequenties \mathbf{y} niet beschikbaar. Bijgevolg kan in tijdstap t het vorige ground-truth-woord u_{t-1} niet als invoer voor het RNN dienen. In de plaats daarvan dient tijdens inferentie \hat{u}_{t-1} of u_{t-1}^* als invoer, zoals op figuur 2.5 te zien is. De uitvoer van het netwerk definieert dan eigenlijk niet $p_{model}(u_t | \mathbf{x}, \mathbf{u}_{1:t-1}; \theta)$ maar $p_{model}(u_t | \mathbf{x}, \hat{\mathbf{u}}_{1:t-1}; \theta)$.

Het genereren van sequenties kan als volgt verlopen. De sequentie waaraan het model de hoogste waarschijnlijkheid toekent is $\mathbf{y}^\diamond = \arg \max_{\mathbf{y}} p_{model}(\mathbf{y} | \mathbf{x})$. Gezien het aantal mogelijke sequenties \mathbf{y} exponentieel groeit in T is de optimale \mathbf{y}^\diamond berekenen in het algemeen niet haalbaar. Enkele andere methoden zijn mogelijk: een eerste optie is in elke tijdstap een sample \hat{u}_t nemen van $p_{model}(u_t | \mathbf{h}_t; \theta)$ en die sample in $t+1$ als invoer aan het RNN geven. Een alternatief is *greedy decoding*: in plaats van \hat{u}_t te samplen telkens $u_t^* = \arg \max_k p_{model}(u_t = k | \mathbf{h}_t; \theta)$ nemen en in $t+1$ als invoer gebruiken. Noteer dat dit laatste niet noodzakelijk de optimale sequentie zal opleveren, gezien [65]:

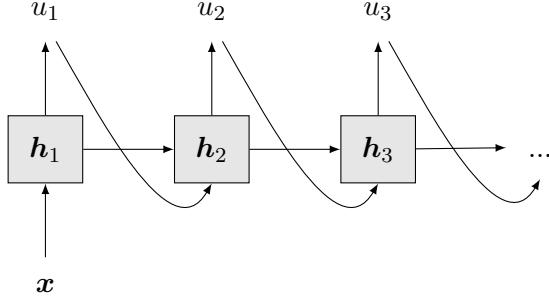
$$\prod_{t=1}^T \max_{u_t} p_{model}(u_t | \mathbf{h}_t; \theta) \leq \max_{\mathbf{u}_{1:T}} \prod_{t=1}^T p_{model}(u_t | \mathbf{h}_t; \theta). \quad (2.18)$$

Een laatste methode om \mathbf{y}^\diamond nauwkeuriger te benaderen is *beam search*: telkens de set van b kandidaatzzinnen met de volgens het model tot dan toe hoogste



Figuur 2.4: Teacher forcing en backpropagation through time

Het trainen van een RNN voor tekstgeneratie met maximum-likelihood en teacher forcing. Figuur 2.4a toont teacher forcing om de kans op het genereren van het volgende ground-truth-woord gegeven de vorige ground-truth-woorden te maximaliseren. Figuren 2.4b, 2.4c en 2.4d tonen hoe de gradiënten van de cross-entropy tussen u_t en \hat{u}_t door het ontvouwen netwerk in de tijd teruggepropageerd worden. De zwarte pijlen zijn verbanden tussen lagen en omvatten parameters (zoals $\mathbf{W}(\dots)$ in figuur 2.3). De rode pijlen zijn de gradiënten van H tegenover de parameters van de zwarte pijlen in stippellijn waar ze over gaan. Noteer dat hier de invoer voor het gehele algoritme \mathbf{x} als eerste woord wordt ingevoerd, maar ook andere manieren hiervoor bestaan.



Figuur 2.5: Tekstgeneratie met een RNN tijdens inferentie

Tijdens inferentie zijn er geen ground-truth-sequenties voorhanden dus dient $\hat{u}_{t-1} \sim p_{model}$ als invoer voor het RNN in tijdstap t .

waarschijnlijkheid bijhouden om als prefix te gebruiken bij het berekenen van het volgende woord [94] [65] [85]. Daarvoor moet het RNN echter b keren p_{model} berekenen voor een andere $\hat{u}_{1:t-1}$, de tijdscomplexiteit van beam search stijgt dus lineair met b .

Terwijl het model getraind is op invoer $u \sim p_{data}$ krijgt het tijdens inferentie als invoer $\hat{u} \sim p_{model}$. Dit voldoet niet aan de voorwaarde voor goede generalisatie in sectie 2.1.2, namelijk dat de invoerdistributies tijdens trainen en inferentie gelijkaardig zijn. Wanneer het model een woord voorspelt dat niet het ground-truth-woord is en dat niet uit p_{data} komt, krijgt het in de volgende tijdstap een foutief woord uit een voor zichzelf ongekende verdeling p_{model} waarvoor het niet getraind is. Zo “kan het model in een deel van de toestandsruimte komen dat het tijdens het trainen nooit gezien heeft” [8] en kunnen fouten in het begin van de sequentie snel accumuleren [103]. Deze discrepantie tussen inferentie en training heet *exposure bias* [65], en ook het ondertitelen van afbeeldingen zoals in [94] lijdt hieronder.

Voorgestelde oplossingen voor exposure bias

De afwijking tussen invoerverdelingen tijdens het trainen en tijdens inferentie bij het voorspellen van sequenties werd al opgemerkt. In [91] wordt een algoritme geïntroduceerd onder de naam *Data As Demonstrator* waarbij de ground-truth trainingdata gemengd wordt met voorspellingen van het model [65]. Bengio et al. passen dit idee toe op tekstgeneratie in [8]. Als oplossing voor het exposure bias-probleem stellen zij *scheduled sampling* voor: een aanpak waarbij met een met de iteraties toenemende kans de invoer voor het RNN niet ground-truth-woord u_{t-1} is maar $\hat{u}_{t-1} \sim p_{model}$. Door geleidelijk aan, naarmate de voorspellingen van het model beter worden, deze als invoer te gebruiken wordt het model meer “bewust” van hoe het tijdens inferentie gebruikt zal worden [65].

Ze verbeteren hiermee baseline resultaten op verschillende NLP taken, o.a. op

2. LITERATUURSTUDIE

afbeeldingen ondertitelen. Deze methode heeft echter ook een duidelijk nadeel: ze traint het netwerk met maximum-likelihood zoals in vergelijking (2.17) om in elke t het volgende ground-truth-woord u_t te voorspellen, ongeacht de voorafgaande woorden. Dit kan als gevolg hebben dat het model getraind wordt om foute sequenties te genereren: “wanneer de ground-truth-zin bijvoorbeeld ‘ik maakte een lange wandeling’ is en het model tot dan toe ‘ik maakte een wandeling’ voorspeld heeft, zal teacher forcing met scheduled sampling het model trainen om een tweede keer het woord ‘wandeling’ te voorspellen” [65].

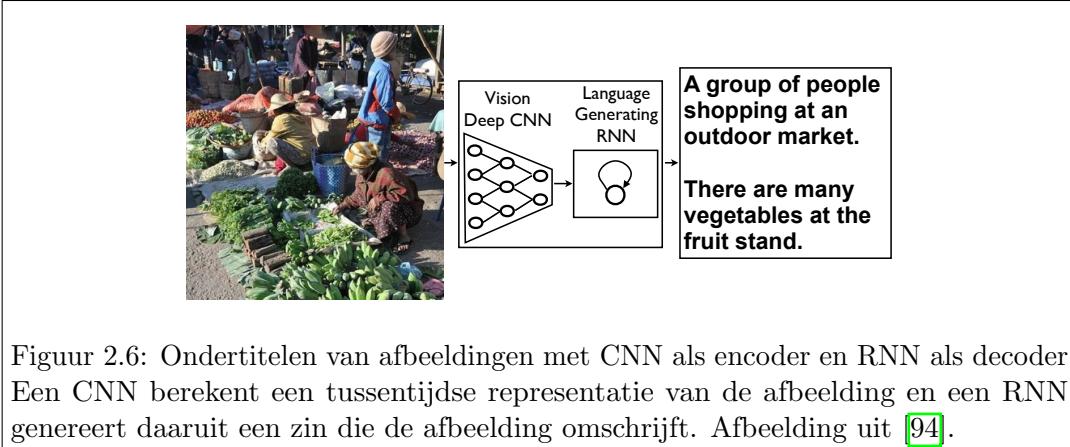
Professor forcing is een andere oplossing voorgesteld door Lamb et al. in [42] waarbij het Generative Adversarial Networks framework van [25] gebruikt wordt. Professor forcing traint naast het tekstgenererende RNN een tweede RNN als discriminator, die verborgen toestandsvectoren van het genererende RNN tijdens inferentie en tijdens trainen leert onderscheiden. De lossfunctie voor tekstgeneratie combineert de cross-entropy zoals bij teacher forcing en het doel van de discriminator te misleiden. Op die manier worden de verborgen toestandsvectoren van het genererende RNN tijdens inferentie en trainen dichter bij elkaar geduwd. Ze halen competitieve resultaten, o.a. bij het genereren van sequenties die langer zijn dan de sequenties die tijdens het trainen gezien zijn.

Een ander idee is om tekstgeneratie als reinforcement-learning-probleem te formuleren [103]. Sectie 2.2.3 gaat daar dieper op in.

2.2.2 Ondertitelen van afbeeldingen

Zoals eerder vermeld traint deze thesis een model voor het ondertitelen van gegeven afbeeldingen met evolutionaire algoritmes. Het ondertitelen van afbeeldingen of *image captioning* (IC) is een NLP-taak die inhoudt dat een functie $f : I \mapsto \mathbf{y}$ geleerd wordt die afbeeldingen I mapt op sequenties van woorden \mathbf{y} in natuurlijke taal die de inhoud van I beschrijven. De Microsoft COCO Captions (MSCOCO Captions) dataset is op dit moment de omvangrijkste en meest kwaliteitsvolle dataset voor deze taak [12, 94], en tevens de dataset waarop modellen in deze thesis getraind worden. De dataset “bevat door mensen geschreven Engelse ondertitels voor afbeeldingen uit de Microsoft Common Objects in COntext dataset” [12], een dataset voor object herkenning [49]. MSCOCO beheert eveneens een evaluatieserver en een scorebord van inzendingen. Voor 123k training- en validatie afbeeldingen zijn vijf ondertitels per afbeelding voorzien. De dataset bestaat dus uit $(I, (\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(5)}))$ -paren. Hoewel één afbeelding meerdere goede ondertitels kan hebben, past IC zo in het supervised learning-kader met als doelfunctie $\mathbf{y}^{(i)} = f(I)$, met $\mathbf{y}^{(i)}$ een van de ondertitels uit \mathcal{D} horende bij I .

Encoder-decoder. Hoewel er tal van methoden zijn voorgesteld voor IC zijn veel van de recente, en tot dusver meest succesvolle, methoden thuis te brengen in een gemeenschappelijk kader [18, 38, 94, 100, 8, 65, 70]. Een *encoder-decoder*-architectuur bestaat uit twee verschillende neurale netwerken, respectievelijk de encoder en de decoder, en genereert een \mathbf{y} gegeven een \mathbf{x} . De encoder verwerkt invoer \mathbf{x} en berekent een tussentijdse representatie \mathbf{q} met een vaste grootte, bv. een vector van dimensie



d_q . Die \mathbf{q} dient als invoer voor de decoder die uit \mathbf{q} de uitvoer \mathbf{y} berekent. De gehele geïmplementeerde functie is dan gelijk aan

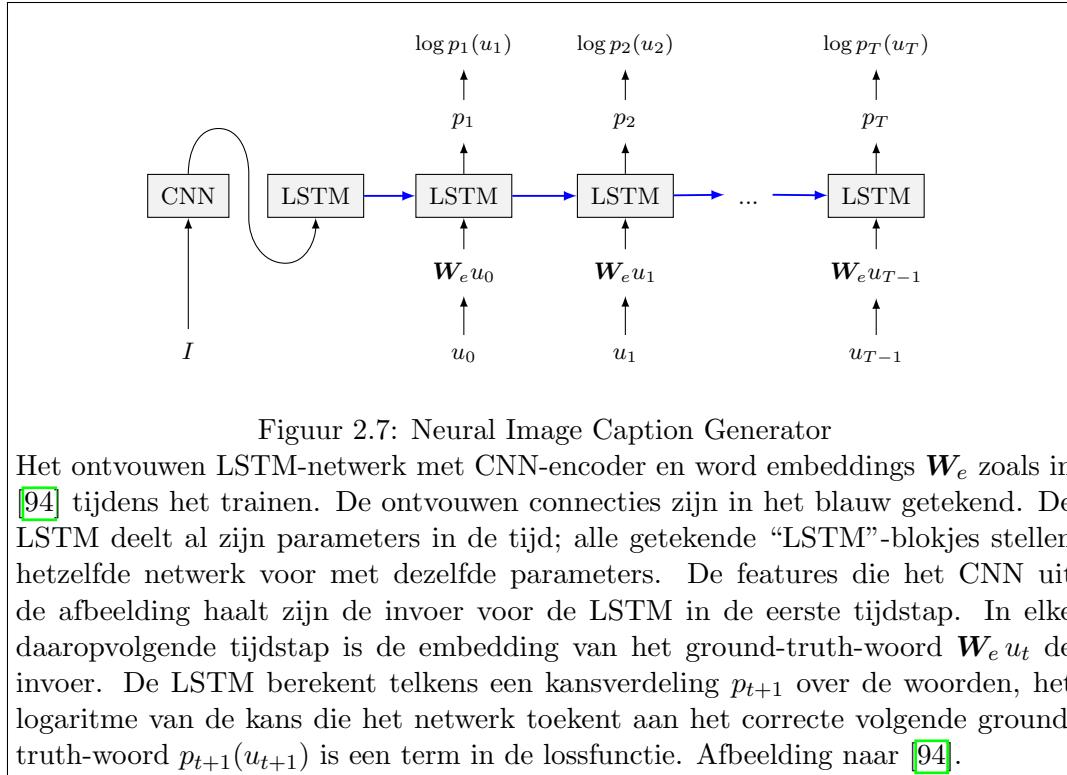
$$\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta}) = \text{dec}(\text{enc}(\mathbf{x}; \boldsymbol{\theta}_{enc}); \boldsymbol{\theta}_{dec}). \quad (2.19)$$

Dit benut het vermogen van neurale netwerken om krachtige representaties te leren of betekenisvolle abstracte features uit de invoer te halen. De encoder en decoder kunnen verschillende vormen aannemen. Dit encoder-decoder-framework wordt veel gebruikt, voor uiteenlopende toepassingen. Zo gebruiken Noh et al. in [61] een convolutional NN en een *deconvolutional* NN om afbeeldingen semantisch te segmenteren. In [85, 13] worden twee RNN's gebruikt om zinnen automatisch te vertalen: eerst verwerkt de encoder de invoerzin woord per woord. De laatste verborgen toestandsvector \mathbf{h}_t dient als tussentijdse representatie \mathbf{q} ; de initiële verborgen toestandsvector \mathbf{h}_1 van de decoder wordt gelijkgesteld aan \mathbf{q} en zo berekent de decoder de uitvoerzin. (Vinyals et al. hanteren dezelfde aanpak in [93] voor een dialoogsysteem.) Gezien de invoer bij IC een afbeelding is, en CNN's geschikt zijn om expressieve features uit afbeeldingen te halen, is het nuttig om als encoder een CNN te gebruiken. De gewenste uitvoer is een sequentie van woorden, en zoals reeds vermeld lenen RNN's zich goed voor het genereren van sequenties. Dat motiveert de keuze van een CNN en een RNN resp. als encoder en decoder. Zie figuur 2.6 voor een overzicht van deze architectuur.

Neural Image Caption Generator. Gezien Vinyals et al. in hun onderzoek [94] de toenmalige state-of-the-art resultaten overtroffen en verschillende latere voorstellen zich baseren op hun opstelling [100, 65, 70, 8] is hun aanpak enige extra duiding waard. Ook de modellen gebruikt in deze thesis zijn gebaseerd op [94].

Als encoder gebruiken ze een CNN dat voorgetraind is op de ImageNet dataset voor de ILSVRC-competitie [74, 33, 17]. Als decoder gebruiken Vinyals et al. een LSTM-netwerk. De laatste laag van het ImageNet-classificatiennetwerk is een lineaire functie $f_{clf} : \mathbb{R}^{d_q} \mapsto \mathbb{R}^{n_c}$ gevuld door een softmax over het aantal klassen (n_c)

2. LITERATUURSTUDIE



Figuur 2.7: Neural Image Caption Generator

Het ontvouwen LSTM-netwerk met CNN-encoder en word embeddings \mathbf{W}_e zoals in [94] tijdens het trainen. De ontvouwen connecties zijn in het blauw getekend. De LSTM deelt al zijn parameters in de tijd; alle getekende “LSTM”-blokjes stellen hetzelfde netwerk voor met dezelfde parameters. De features die het CNN uit de afbeelding haalt zijn de invoer voor de LSTM in de eerste tijdstap. In elke daaropvolgende tijdstap is de embedding van het ground-truth-woord $\mathbf{W}_e u_t$ de invoer. De LSTM berekent telkens een kansverdeling p_{t+1} over de woorden, het logaritme van de kans die het netwerk toekent aan het correcte volgende ground-truth-woord $p_{t+1}(u_{t+1})$ is een term in de lossfunctie. Afbeelding naar [94].

en is dus taak-specifiek. Daarom worden de activaties van de voorlaatste laag als tussentijdse representatie \mathbf{q} gebruikt. De tussentijdse representatie die het CNN uit afbeelding I heeft berekend, wordt als eerste woord z_1 aan de LSTM gegeven. De invoerwoorden gaan eerst door een embeddinglaag waar ze van one-hot encoding omgezet worden naar word embeddings. Het resulterende proces is als volgt [94]:

$$\begin{aligned} z_1 &= \text{CNN}(I) \\ z_t &= \mathbf{W}_e u_{t-2}, \quad t \geq 2 \\ \mathbf{h}_t, (p_{\text{model}})_t &= \text{LSTM}(\mathbf{h}_{t-1}, z_{t-1}), \quad t \geq 1 . \end{aligned} \tag{2.20}$$

Hierbij zijn $\mathbf{y} = (u_1, \dots, u_T)$ de one-hot encoderingen van een ground-truth-zin, u_0 een speciaal “start”-woord dat het begin van een zin markeert, u_T analoog een “einde”-woord voor het einde van een zin en \mathbf{W}_e een matrix van word embeddings. De softmax functie wordt als onderdeel van de LSTM-functie gezien. Figuur 2.7 toont de opstelling zoals hier gedefinieerd.

De parameters van het CNN worden niet meer aangepast tijdens het trainen. De parameters van het LSTM-netwerk en de word embeddings worden met gradient descent onder teacher forcing getraind: d.w.z. met lossfunctie $L(\mathbf{y}, I) = -\sum_{t=1}^T \log p_{\text{model}}(u_t | I, \mathbf{u}_{1:t-1})$.

Niet-differentieerbare evaluatiefuncties. De lossfunctie gebruikt bij het trainen van tekstgenerators met teacher forcing, is reeds besproken: de cross-entropy per woord in de sequentie. Voor het maximaliseren van de nauwkeurigheid van een classificatiemodel is de cross-entropy een goede *endpoint*, omdat exact de kans op juist classificeren wordt gemaximaliseerd. Vaak worden tekstgenerators echter beoordeeld met andere functies omdat deze beter correleren met menselijk oordeel [65, 12]. Deze functies zijn typisch gebaseerd op het voorkomen van *n-grams*, groepjes van n opeenvolgende woorden met vaste volgorde, in kandidaat- en referentiezinnen.

BLEU [62] is bijvoorbeeld een score die vaak wordt gebruikt voor het beoordelen van gegenereerde tekst tegen een referentiecorpus, o.a. bij vertaalsystemen, dialoogsystemen of IC [94, 65, 85]. CIDEr of “Consensus-based Image Description Evaluation” [90] is een van de meestgebruikte scores voor het ondertitelen van afbeeldingen [70, 35, 8].

Om de CIDEr-score van een zin ten opzichte van de referentiezinnen in een dataset te berekenen, wordt eerst een *Term Frequency - Inverse Document Frequency* of TF-IDF¹ gewicht berekend voor elk n -gram ω_k dat voorkomt in de dataset [12]. In de uitdrukking voor dat gewicht komen termen $h(\omega_k, \mathbf{y})$ voor, met h een functie die het aantal keer dat n -gram ω_k voorkomt in zin \mathbf{y} teruggeeft. Zin \mathbf{y} kan een kandidaatzin, voorspeld door het model, of een referentiezin uit de dataset zijn. Gezien een woord of een n -gram ofwel een geheel aantal keer voorkomt ofwel niet voorkomt is de CIDEr-score discreet. Dat is belangrijk omdat discrete functies niet differentieerbaar zijn en dus ook niet rechtstreeks als lossfunctie voor gradient descent bruikbaar zijn. Ook de BLEU-score en veel andere evaluatiefuncties voor tekstgeneratie zijn discreet. Bijgevolg wordt zoals eerder vermeld vaak de cross-entropy loss gebruikt. Dat creëert naast de besproken exposure bias een tweede discrepancie tussen het trainen en inferentie: het gebruik van een andere finale evaluatiefunctie dan de functie waarvoor de parameters geoptimaliseerd worden.

2.2.3 Reinforcement learning voor tekstgeneratie

Tekstgeneratie, en daarmee ook IC, boeten in aan performantie door de twee beschreven verschillen tussen trainen en inferentie. Deze verschillen zijn exposure bias en evaluatie met discrete evaluatiefuncties, waardoor genoodzaakt een andere, wel differentieerbare, lossfunctie gebruikt wordt tijdens het trainen. Tekstgeneratie formuleren als reinforcement-learning-probleem werd voorgesteld als oplossing om deze afwijkingen te overbruggen. Veel methoden waarmee tijdens de laatste jaren competitieve en state-of-the-art resultaten werden gehaald voor tekstgeneratietaken, maken gebruik van RL-technieken: afbeeldingen ondertitelen [65, 70, 50], dialoogsystemen [48], samenvatten van tekst [64, 65] of automatisch vertalen [99, 65].

Dat is relevant voor deze thesis omdat er gelijkenissen zijn tussen de probleemstelling in reinforcement learning en bij evolutionaire algoritmes, wat verder in sectie 2.3 besproken wordt. De goede resultaten van RL-algoritmes voor tekstgeneratie

¹Product van een factor die proportioneel is aan het aantal keer dat een n -gram voorkomt in de kandidaatzin, en een factor die omgekeerd proportioneel is aan het aantal keer dat dat n -gram voorkomt in alle referentiezinnen.

2. LITERATUURSTUDIE

suggereren dus dat mogelijks evolutionaire algoritmes ook interessant kunnen zijn in deze context. Bovendien kunnen ook ondervonden moeilijkheden en de oplossingen hiervoor bij RL-algoritmes relevant zijn voor het toepassen van EA's.

Ranzato et al. formuleren tekstgeneratie in een RL-kader als volgt: “het generatieve model (het RNN) kan gezien worden als agent, die met de omgeving interageert (de vorige woorden $\mathbf{u}_{1:t-1}$ en de invoer \mathbf{x} die het RNN observeert). De parameters van de agent definiëren een policy waarvan de uitvoer de acties die de agent neemt bepaalt. Bij het genereren van sequenties zoals hier verwijst een actie naar het voorspellen van het volgende woord in elke tijdstap. Na het uitvoeren van een actie updatet de agent zijn interne state (de verborgen lagen van het RNN). Eens de agent het einde van een sequentie bereikt observeert hij een beloning” [65].

Deze formulering laat toe het model op sequentieniveau te beoordelen door beloningen toe te kennen aan hele sequenties en niet enkel woord per woord zoals bij teacher forcing. Dat laat dan weer toe tijdens het trainen al de voorspellingen van het model \hat{u}_{t-1} of u_{t-1}^* als invoer te gebruiken in de volgende tijdstap en zo het exposure bias-probleem te overkomen.

Policy gradients. Verder legt RL geen restricties op omtrent de beloningen. Deze moeten niet uit een gekende functie komen en dus ook niet uit een differentieerbare functie zoals eerder gezien bij het rechtstreeks toepassen van gradient descent op het trainen van modellen voor tekstgeneratie. In de plaats van een doelfunctie direct te differentiëren bieden policy-gradient-algoritmes uit RL technieken aan om uit waargenomen beloningen toch gradiënten te berekenen, zonder assumpties over hoe deze beloningen berekend worden. De beloningen kunnen gezien worden alsof ze uit een zwarte doos komen.

Policy-gradient-methoden (PG) zijn een klasse van algoritmes binnen RL [86, 87], met als klassieke voorbeeld het *Reinforce*-algoritme van Williams [97]. Het zijn methoden die beogen beloningen te maximaliseren door de parameters van een policy-functie $\pi(a|s; \boldsymbol{\theta})$ iteratief aan te passen in de richting van de gradiënt van een te maximaliseren functie van de beloningen, ten opzichte van $\boldsymbol{\theta}$. Policy-gradient-methoden houden dus zelf een vorm van gradient descent² in en de doelfunctie moet bijgevolg differentieerbaar zijn. De te minimaliseren lossfunctie is hier gedefinieerd als de negatieve verwachte totale beloning onder de huidige policy. Williams toonde aan dat de gradiënt van deze lossfunctie ook als een eenvoudige verwachting geschreven kan worden [97]:

$$\begin{aligned} J(\boldsymbol{\theta}) &= -\mathbb{E}_{\pi_{\boldsymbol{\theta}}}[r(\pi_{\boldsymbol{\theta}})] \\ \Rightarrow \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= -\mathbb{E}_{\pi_{\boldsymbol{\theta}}}[r(\pi_{\boldsymbol{\theta}}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s; \boldsymbol{\theta})] \end{aligned} \quad (2.21)$$

²Omdat PG-methoden beloningen maximaliseren zou gradient ascent een intuïtievere term zijn. Minimalisatie van een lossfunctie met gradient descent en maximalisatie van een doelfunctie met gradient ascent zijn echter equivalent: de loss- of doelfunctie vermenigvuldigen met -1 levert het duale probleem, met dezelfde oplossing.

Met $r(\pi_{\theta})$ de totale beloning onder policy π_{θ} . De elementen uit RL mappen op tekstgeneratie, i.e. de policy-functie invullen met p_{model} en de acties met sequenties $\hat{\mathbf{y}}$, levert de volgende verwachting. Vervolgens deze verwachting schatten met een Monte-Carlo sample van p_{model} resulteert in de schatter [70]:

$$\begin{aligned}\nabla_{\theta} J(\theta) &= -\mathbb{E}_{\hat{\mathbf{y}} \sim p_{model}} [r(\hat{\mathbf{y}}) \nabla_{\theta} \log p_{model}(\hat{\mathbf{y}}; \theta)] \\ &\approx -r(\hat{\mathbf{y}}) \nabla_{\theta} \log p_{model}(\hat{\mathbf{y}}; \theta)\end{aligned}\quad (2.22)$$

En $r(\hat{\mathbf{y}})$ kan zonder probleem genomen worden als CIDEr($\hat{\mathbf{y}}, \mathbf{y}$). De beloningen volgen uit evaluaties van de policy op de doeltaak, *rollouts* genaamd. Bij tekstgeneratie betekent een rollout bijvoorbeeld een zin of enkele zinnen genereren. Om de variantie van de schatter voor de gradiënt te verkleinen wordt er vaak een referentiebeloning of *baseline* b gebruikt om de beloningen te normaliseren [86, 65]:

$$\nabla_{\theta} J(\theta) \approx -(r(\hat{\mathbf{y}}) - b) \nabla_{\theta} \log p_{model}(\hat{\mathbf{y}}; \theta). \quad (2.23)$$

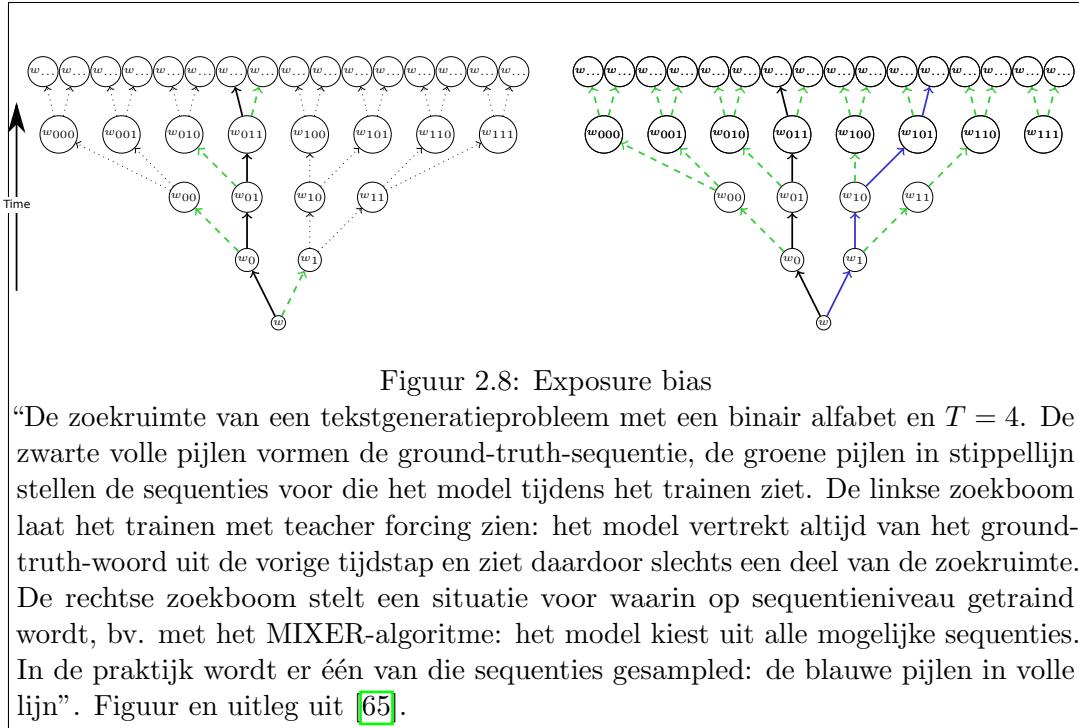
Baseline b kan verschillende vormen aannemen: zolang b niet varieert met gekozen acties a of $\hat{\mathbf{y}}$ blijft het rechterlid in (2.23) een zuivere schatter voor de gradiënt. Een voorbeeld van een eenvoudige baseline is de gemiddelde beloning tot dan toe.

MIXER. Ranzato et al. passen het bovenstaande met succes toe op enkele tekstgeneratietaken, o.a. afbeeldingen ondertitelen, en halen betere resultaten dan tot dan toe gebruikte methoden [65]. Ze merken wel op dat het tekstgeneratieprobleem een enorm grote zoekruimte heeft. Indien $V = 10^4$ dan zijn er bij teacher forcing in elke tijdstap 10^4 mogelijke uitvoerwoorden waartussen het model moet kiezen. Onder Reinforce en andere PG-methoden wordt echter een beloning toegekend aan volledige zinnen, en moet het model dus een hele zin genereren alvorens het feedback krijgt. Bij het trainen op sequentieniveau is het aantal mogelijke sequenties bijgevolg gelijk aan $V^T = 10^{40}$ voor een sequentielengte $T = 10$. Daarom leert het Reinforce-algoritme zeer traag, wanneer het start van een willekeurige policy (i.e. willekeurige θ).

Om dit verschil in grootte van de zoekruimtes, en dus in moeilijkheid van de problemen, te overbruggen stellen ze MIXER voor: een aangepaste versie van Reinforce. De eerste aanpassing houdt in dat er gestart wordt van een policy die voorgetraind is met teacher forcing. De tweede aanpassing houdt in dat een met de iteraties afnemend aantal eerste woorden van elke zin nog met cross-entropy getraind wordt, en de rest van de zin met Reinforce. Met het standaard Reinforce-algoritme zonder voortraining en zonder gecombineerd XENT-RL trainingschema krijgen ze geen convergentie binnen een “redelijke tijd” [65]. Figuur 2.8 vergelijkt de zoekruimte bij teacher forcing met de zoekruimte bij trainen op sequentieniveau, bv. met MIXER.

Self-critical sequence training. Rennie et al. bouwen verder op MIXER en stellen voor om als baseline “de beloning bekomen door het huidige model onder het

2. LITERATUURSTUDIE



algoritme gebruikt tijdens inferentie” te gebruiken [70], i.e. onder greedy decoding. Dat wil zeggen dat de gradiënt van de lossfunctie geschat wordt door:

$$\nabla_{\theta} J(\theta) \approx -(r(\hat{y}) - r(y^*)) \nabla_{\theta} \log p_{model}(\hat{y}; \theta) \quad (2.24)$$

Met \hat{y} een zin bekomen door in elke tijdstap $\hat{u}_t \sim p_{model}$ te samplen, en y^* de zin bekomen door telkens $u_t^* = \arg \max p_{model}$ te nemen. Ze noemen hun methode *self-critical sequence training* (of SCST): tijdens het trainen worden de waarschijnlijkheden van samples die een hogere beloning halen dan de huidige policy onder inferentie omhoog geduwd en vice versa. Het is niet nodig zoals in [65] een andere baseline te schatten, en gezien de baseline het huidige algoritme zelf is kan dit (op variantie na) enkel verbeterd worden tijdens het trainen [70]. Hoewel hun training bij policy gradients ook vertrekt van een voorgetrainde policy, vinden ze het niet nodig om zoals in MIXER het eerste deel van de zin nog met cross-entropy te trainen. De $r(y^*)$ -baseline vraagt wel dat er voor elke sample niet één maar twee zinnen berekend worden en vraagt dus twee forward passes: een door te samplen en een door greedy decoding. Hun methode verbetert de state-of-the-art resultaten in afbeeldingen ondertitelen van dat moment met bijna %10.

Paulus et al. passen SCST toe op modellen die tekst samenvatten en behalen eveneens state-of-the-art resultaten [64].

Andere. MIXER kent aan elk woord in de zin dezelfde beloning als de beloning van de gehele zin toe. Dat stelt de bijdrage aan de beloning van elk voorspelde

woord dus gelijk, terwijl dat niet per se zo is. Liu et al. verbeteren MIXER op dat vlak door de bijdrage aan de beloning van elk woord te schatten met de beloningen van Monte-Carlo rollouts. Deze rollouts vertrekken van de prefix van de zin tot en met het huidige woord [50]. Ze trainen hun modellen ook voor een hybridescore van CIDEr en SPICE: de laatste is een score voor ondertitels voor afbeeldingen die meer op semantische inhoud gericht is dan conventionelere scores gebaseerd op *n*-gram-overlap [2]. Ze rapporteren dat de ondertitels gegenereerd door modellen die met de hybridescore getraind zijn, verkozen worden door mensen. Dat onderschrijft het nut van de vrijheid in beloningfuncties bij policy-gradient-methoden extra.

Ook Li et al. maken gebruik van de vrijheid in beloningen voor het trainen van een dialoogsysteem met policy gradients: ze definiëren drie beloningfuncties die drie aspecten van een gewenst dialoogsysteem pogen te kwantificeren, zoals semantische coherentie [48]. In een evaluatie-experiment duiden menselijke juryleden antwoorden van modellen getraind met een combinatie van de drie beloningen aan als kwaliteitsvoller dan antwoorden van gebruikelijkere seq2seq modellen [93].

2.3 Neuro-evolutie

Omdat de thesis een evolutionair algoritme voorstelt om deep learning modellen te trainen is enige achtergrond over evolutionaire algoritmes (sectie 2.3.1) en het gebruik ervan in deep learning (sectie 2.3.2) op zijn plaats. Sectie 2.3.3 bespreekt daarna enkele recent voorgestelde evolutionaire algoritmes voor RL.

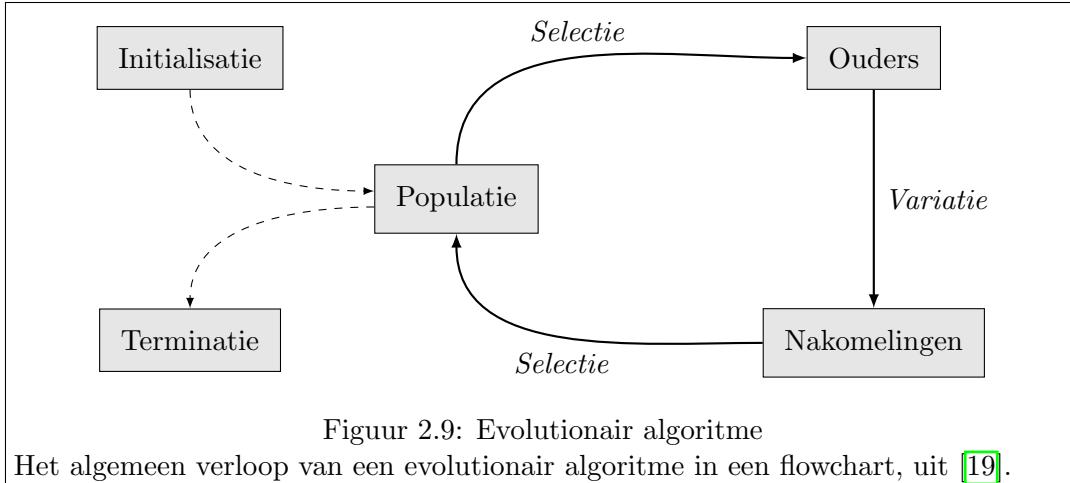
2.3.1 Evolutionaire algoritmes

Evolutionaire algoritmes (EA's) zijn een klasse van metaheuristische *black-box*-algoritmes gebaseerd op principes uit natuurlijke evolutie die hoofdzakelijk voor optimalisatieproblemen gebruikt worden [19]. Er bestaan verschillende varianten van evolutionaire algoritmes, maar ze hebben een gemeenschappelijke basisstructuur.

Zoals Eiben en Smith in [19] schrijven, genereert een EA voor een te maximaliseren doelfunctie eerst een populatie van willekeurige kandidaatoplossingen of individuen en berekent hun fitness, de waarde van de doelfunctie voor een kandidaatoplossing. Daarna herhaalt een EA de volgende stappen: naargelang hun fitness selecteert een EA de beste kandidaatoplossingen als ouders voor de volgende generatie: het EA creëert een aantal nakomelingen door het toepassen van variatie-operatoren op de ouders. De volgende generatie wordt geselecteerd uit de nieuwe nakomelingen en de vorige generatie, op basis van de leeftijd van individuen of opnieuw op basis van hun fitness. Figuur 2.9 geeft dit iteratief proces schematisch weer.

De twee drijfveren van EAs zijn enerzijds variatie-operatoren en anderzijds selectie-operatoren. **Variatie-operatoren** genereren vanuit een of meer bestaande individuen een of meer nieuwe individuen. Variatie zorgt voor diversiteit en dus innovatie in EAs. **Selectie-operatoren** duwen de fitness van individuen in de populatie omhoog door individuen met een hogere fitness een grotere kans op voortplanting en een grotere kans op overleven te geven, en zo selectiedruk uit te oefenen [19].

2. LITERATUURSTUDIE



De beschrijving black-box-algoritme duidt op de manier waarop EA's met de doelfunctie omgaan: een EA behandelt de doelfunctie als zwarte doos, enkel evaluaties van de functie in gegeven punten in de zoekruimte zijn nodig. Bijvoorbeeld gradient descent daarentegen legt meer voorwaarden op aan de doelfunctie, namelijk dat deze analytisch en differentieerbaar is, zodat de structuur van de doelfunctie benut kan worden.

Een belangrijke groep van evolutionaire algoritmes zijn evolutiestrategieën of ES [68, 78]. De representatie van individuen in ES zijn vectoren van reële getallen. Dat kunnen bijvoorbeeld vectorrepresentaties van de parameters θ van neurale netwerken zijn. Een typerende tactiek in ES is zelf-adaptatie van hyperparameters, waarbij hyperparameters mee worden gecodeerd in de genotypes en zo ook mee evolueren. *Covariance Matrix Adaptation* [28] of CMA is een bekend voorbeeld van een ES dat gebruikt maakt van zelf-adaptatie.

Een implementatie van een EA moet de volgende concepten concretiseren [19]. Gezien de meeste evolutionaire algoritmes die deze thesis bespreekt en toepast, thuis te brengen zijn onder een ES wordt de manier waarop een ES de concepten invult toegelicht.

Representatie. Een EA opereert op representaties van kandidaatoplossingen of individuen. Bijgevolg heeft het een abstracte representatie van kandidaatoplossingen nodig, alsook een manier om een kandidaatoplossing van het reële naar het abstracte domein om te zetten en vice versa. Het individu in het originele domein wordt het fenotype genoemd, de abstracte representatie het genotype. Een gen is een stukje van het genotype dat één bepaalde feature codeert.

Bij een ES zijn de genotypes vectoren van reële getallen $\gamma \in \mathbb{R}^{d_\gamma}$. Een fenotype zou kunnen zijn een neuraal netwerk met parameters $\theta = \gamma$.

Evaluatie. Een evaluatiefunctie of fitnessfunctie $F(\gamma) : \mathbb{R}^{d_\gamma} \mapsto \mathbb{R}$ is nodig om de kwaliteit van individuen te kwantificeren. Deze functie representeert het probleem waarvoor het algoritme een oplossing zoekt. De fitness is de waarde die de evaluatiefunctie toekent aan een individu. Doorgaans is het fenotype nodig voor het berekenen van de fitness en kan dit niet rechtstreeks uit het genotype: bv. de nauwkeurigheid van een neuraal netwerk waarvan de gewichten het genotype zijn. De evaluatiefunctie is probleemspecifiek en wordt dus gekozen in functie van het probleem dat opgelost wordt.

Om diversiteit te promoten en zoveel mogelijk verschillende oplossingen te exploreren, wordt soms de notie van fitness gedurende het algoritme vervangen door (of aangevuld met) nieuwheid of *novelty*. In plaats van tijdens selectie de voorkeur te geven aan individuen met een hogere fitness kiest het algoritme dan voor individuen die het meest innoverend zijn. Uiteindelijk is de bedoeling om te komen tot een oplossing met een zo hoog mogelijke fitness, maar door novelty te volgen kan het algoritme regio's vinden met een hoge fitness die het niet zou vinden door fitness te volgen omdat het in lokale optima terecht komt. Dit principe heet *novelty search*.

Populatie. De populatie bevat de genotypes van de huidige kandidaatoplossingen. Het aantal individuen in de populatie of de populatierootte μ is een hyperparameter en moet gekozen worden.

Selectie van ouders. Gebaseerd op hun fitness worden individuen gekozen als ouders voor de volgende generatie. Ze ondergaan variatie en genereren nakomelingen. Individuen met grotere fitness hebben doorgaans meer kans om als ouder gekozen te worden, maar de selectie is vaak stochastisch en individuen met een kleinere fitness krijgen vaak ook een kleine maar positieve kans om gekozen te worden.

Het meestgebruikte selectiemechanisme in ES is uniforme selectie, i.e. uit een populatie van μ individuen elk individu met een gelijke kans $p_{ouder} = \frac{1}{\mu}$ selecteren als ouder. Bij dit mechanisme moet alle selectiedruk komen van de selectie van de overlevenden uit de nakomelingen.

Een ander veelvoorkomend selectiemechanisme in evolutionaire algoritmes is toernooiselectie: om ρ ouders te kiezen uit μ individuen worden ρ keer S individuen gekozen met gelijke kans. Uit deze S individuen wordt telkens het individu met de hoogste fitness als ouder genomen. S is de toernooigrootte, een hyperparameter die mee de selectiedruk bepaalt: hoe hoger S hoe kleiner de kans dat individuen met een relatief lage fitness als ouder gekozen worden.

Variatie. Variatie-operatoren zorgen voor het genereren van nieuwe individuen uit bestaande individuen. De belangrijkste variatie-operatoren zijn mutatie en recombinatie.

Een mutatie is een stochastische verandering die uit één genotype één nieuw genotype creëert. De meest voorkomende mutatie in ES telt bij elk element in het genotype een getal, getrokken uit een normaalverdeling met gemiddelde 0 en standaardafwijking σ op, met σ de mutatierootte: een gekozen hyperparameter. In

2. LITERATUURSTUDIE

symbolen: $\gamma_{t+1} = \gamma_t + \sigma\epsilon$ met $\epsilon \sim N(0, I)$. Bij zelf-adapterende mutaties maken naast de gezochte probleemspecifieke parameters θ ook parameters σ_i deel uit van de genotypes γ : één globale σ , een σ_i per θ_i of variabelen waarmee een covariantiematrix gevormd kan worden zoals in [28].

Recombinatie is eveneens een stochastische operator maar combineert genotypes van twee of meer ouders tot een of meer nieuwe genotypes. Verschillende recombinaties zijn mogelijk: in een veelgebruikte variant wordt bijvoorbeeld elke γ_i ofwel uit de eerste ouder gekozen als $\gamma_i^{(1)}$, ofwel als $\gamma_i^{(2)}$ uit de tweede ouder.

Selectie van overlevenden. Na het produceren van nakomelingen stelt een EA de volgende generatie van individuen samen. Gezien de populatiegrootte in het algemeen constant is moet een EA individuen selecteren uit de vorige generatie en uit de nakomelingen. De volgende generatie van μ individuen kiezen uit $\lambda + \mu$ nakomelingen en individuen uit de vorige generatie kan op basis van fitness en/of op basis van leeftijd. Bij (μ, λ) -selectie, het meestgebruikte mechanisme in ES, spelen zowel leeftijd als fitness een rol: de vorige generatie wordt volledig vervangen door de beste μ individuen uit de λ nakomelingen ($\lambda > \mu$).

Ook het aantal nakomelingen per generatie λ is een hyperparameter. Eiben en Smith raden een μ/λ -verhouding van 1/7 tot 1/4 aan [19].

Elitisme is een techniek die vaak samen met selectie op basis van leeftijd en fitness gebruikt wordt: de volgende generatie bevat altijd een onveranderde kopie van het beste individu tot dan toe, om te vermijden dat de tot dan toe beste oplossing verloren gaat.

Initialisatie. Bij de meeste EA's bestaat de eerste populatie uit willekeurig geïnitialeerde individuen. Heuristieken kunnen ook gebruikt worden. De initialisatieprocedure is probleemafhankelijk en is dus niet afhankelijk van het gebruikte type EA.

Bij het ontwerpen van een EA speelt de exploratie vs. exploitatie trade-off een belangrijke rol. De gekozen variatie-operatoren zijn verantwoordelijk voor exploratie van onbekende delen van de zoekruimte, terwijl de selectiedruk uitgeoefend door de twee selectiefases de generaties meer en meer concentreert rond optima (exploitatie). Een goed evenwicht is belangrijk want een te kleine selectiedruk leidt tot trage convergentie en een te grote selectiedruk ontmoedigt innovatie en kan het algoritme in lokale optima vasthouden.

EA's zijn bijvoorbeeld al gebruikt om een zeer nauwkeurige schatting van het gewicht van een quark in een deeltjesversneller te maken [1], voor routebepaling in transport [92], het plannen van waterinfrastructuur [60] of 3D-beeldmodellering [76].

2.3.2 Evolutionaire algoritmes in deep learning

De term “neuro-evolutie” duidt op het gebruik van evolutionaire algoritmes voor het evolueren van neurale netwerken. Hoewel gradient descent en zijn varianten de laatste jaren onder andere door het succes van efficiënte implementaties van

backpropagation de norm zijn in deep learning [24], is neuro-evolutie ook al lang een actief onderzoeksgebied [55, 21, 101, 102, 4, 82, 81] en is er sinds recent een heropleving van de interesse in EA's voor deep learning [52, 83, 67, 66].

Algoritmes voor neuro-evolutie kunnen zoals in de volgende secties opgedeeld worden naargelang ze de gewichten en/of de topologie van neurale netwerken zoeken. De algoritmes die deze thesis voorstelt, vallen onder het eerste type. In wat volgt zal met gewichten en parameters van een neurale netwerk hetzelfde bedoeld worden, namelijk $\theta = (\mathbf{W}, \mathbf{b})$. De tekst zal soms de minder omvattende term “gewichten” gebruiken om het verschil tussen de parameters van een NN en de (hyper)parameters van een EA of de architectuur van een NN te benadrukken. De evaluatiefunctie in de context van EA's zal worden geschreven als $F(\gamma)$. In de context van deep learning werd vaak de lossfunctie $J(\theta)$ gebruikt, maar door de te maximaliseren fitness als de negatieve loss te nemen worden beide equivalent.

Onder meer de eigenschap dat EA's black-box-algoritmes zijn kan aantrekkelijk zijn voor toepassingen in deep learning. Zoals eerder vermeld vereist gradient descent differentieerbaarheid van de lossfunctie, wat niet altijd mogelijk is. Bij bepaalde RL-taken bv. is de lossfunctie zelfs niet gekend en in zo'n gevallen kan neuro-evolutie een alternatief zijn. De evaluatiefunctie moet bij EA's, zoals bij policy gradients in reinforcement learning, niet gekend of analytisch zijn en mag bijvoorbeeld ook stochastisch zijn. De enige vereiste is dat het EA voor een individu γ van F een getal kan terugkrijgen dat aangeeft hoe goed γ is.

Vaste topologie

Een eerste groep algoritmes zoekt enkel de gewichten $\theta = (\mathbf{W}, \mathbf{b})$ van een neurale netwerk met een vaste topologie door ze te evolueren. Dat wil zeggen dat de architectuur, het type lagen en het aantal lagen, de dimensies etc. en dus de functiecompositie zoals in vergelijking (2.1) vastligt, doorgaans gekozen door domeinexperts. De specifieke functie die het model implementeert, kan natuurlijk wel nog variëren met θ . In dit kader fungeert het EA als alternatief voor gradient descent. Gezien $\theta \in \mathbb{R}^{d_\theta}$ zijn deze algoritmes vaak varianten van evolutiestrategieën, hoewel het niet altijd eenduidig is onder welke categorie van EA's een algoritme valt.

Igel gebruikt in [32] een CMA-ES zoals hierboven vermeld voor het evolueren van de gewichten van een neurale netwerk voor een RL-taak. Hausknecht et al. vergelijken in [29] enkele algoritmes voor neuro-evolutie voor policynetwerken die Atari-videoospellen leren spelen. Atari-videoospellen leren spelen in een emulator door uit pixels acties te berekenen is een veelgebruikte RL-benchmark [7]. Hausknecht et al. vergelijken een “conventioneel neuro-evolutie”-algoritme (ES) en een CMA-ES die voor een vaste topologie de gewichten zoeken, en NEAT en HyperNEAT (volgende sectie), die ook de topologie evolueren. Het *Evolino*-algoritme van Schmidhuber et al. in [77] evolueert een deel van de gewichten van een LSTM-netwerk met een ES en gebruikt het resulterende netwerk voor *time series prediction* en classificatie van grammatica's.

2. LITERATUURSTUDIE

Ook relevant voor deze thesis is het *Limited Evaluation Evolutionary Algorithm* (LEEA) uit [59]. Traditioneel werden neuro-evolutie-algoritmes vooral op RL-problemen toegepast, maar Morse en Stanley stellen een EA voor dat individuen evalueert op minibatches uit een dataset, zodat het voor supervised learning gebruikt kan worden. Het algoritme neemt elke generatie een nieuwe minibatch uit de dataset en berekent een fitness-score van individuen in de huidige generatie voor die minibatch. Door verschillen in moeilijkheidsgraad van de minibatches zou het kunnen dat individuen die globaal gezien een hoge fitness hebben, verloren geraken doordat ze op de huidige minibatch toevallig minder goed scoren. Om dat verschil in fitness van individuen over de generaties heen, veroorzaakt door verschillen in moeilijkheidsgraad van de minibatches, te overbruggen introduceren ze *fitness inheritance*: een nakomeling erft een deel van de fitness van zijn ouder(s) waardoor schommelingen in fitness over de generaties heen gedempt worden.

Hoewel recombinatie veelvuldig gebruikt wordt in neuro-evolutie, doen de algoritmes die deze thesis voorstelt in hoofdstuk 3 dat niet. Het is namelijk niet triviaal om een recombinatie-operator voor een parametervector als genotype te definiëren die niet te destructief is voor de functionaliteit van het NN. Neuronen in opeenvolgende lagen van neurale netwerken zijn natuurlijk sterk afhankelijk van elkaar en de functie die ze berekenen wordt mee bepaald door de uitvoerverdeling van de neuronen waarvan ze invoer krijgen. Gewichten van onderling afhankelijke neuronen uit opeenvolgende lagen kunnen echter op heel verschillende plaatsen in de parametervector voorkomen. Het is dus te verwachten dat deze parametervector op willekeurige plaatsen doortrekken en mengen met andere vectoren, zoals recombinatie-operatoren uit 19 typisch doen, de functies die de neuronen berekenen significant zal veranderen. Recombinaties die niet blind ‘‘hakken en lijmen’’ vragen dan weer meer implementatielogica.

Met uitzondering van die in 29 werden bovenstaande algoritmes, net als veel van de voorgestelde algoritmes in publicaties over neuro-evolutie, getest op netwerken met $\dim(\theta)$ in de grootte-orde van $\mathcal{O}(10^2)$ tot $\mathcal{O}(10^4)$. De state-of-the-art modellen uit sectie 2.2.2 zoals in 70, 94 hebben echter een aantal parameters in de grootte-orde van $\mathcal{O}(10^6)$ tot $\mathcal{O}(10^7)$. Het is niet evident dat algoritmes en technieken die werken voor een optimalisatieprobleem in $\mathcal{O}(10^3)$ dimensies ook werken voor optimalisatieproblemen in $\mathcal{O}(10^6)$ of meer dimensies: Hausknecht et al. rapporteren conform met deze bedenking moeilijkheden bij netwerken met $\mathcal{O}(10^5)$ parameters.

Recent onderzoek wijst echter uit dat het toch mogelijk is om de parameters van grotere netwerken te zoeken met EA’s: Salimans et al. gebruiken in 75 een natuurlijke evolutiestrategie en slagen erin $\mathcal{O}(10^6)$ parameters te evolueren voor een netwerk dat o.a. competitieve scores haalt in Atari-spellen. Ook Such et al. in 83 slagen erin de parameters voor een netwerk van gelijkaardige grootte met een ES te evolueren en eveneens competitieve resultaten te halen. Sectie 2.3.3 bespreekt de algoritmes uit 75 en 83 in meer detail.

Evoluerende topologie

Een tweede groep algoritmes evolueert de topologie van de netwerken. De gewichten kunnen dan ofwel ook geëvolueerd worden zoals in [82], ofwel met een variant van gradient descent gezocht worden zoals in [52]. Bekende voorbeelden zijn NEAT en HyperNEAT [82, 81, 71, 80]. De bijdrage van algoritmes zoals NEAT en HyperNEAT bestaat onder andere uit speciaal ontworpen genotypes die slimmere operaties toelaten dan dat eenvoudige vlakke vectoren van parameters doen. Die genotypes maken het onder andere mogelijk om recombinatie-operatoren te definiëren die niet te destructief zijn, en om geavanceerdere technieken uit EA's zoals *speciation* te gebruiken. Omdat algoritmes die topologieën evolueren in deze thesis niet gebruikt worden wordt hier niet verder op ingegaan.

Wel relevant is dat Miikkulainen et al. zo een algoritme gebruiken om de topologie van neurale netwerken te evolueren voor image captioning [52]. Ze slagen erin een model te evolueren dat na trainen met gradient descent betere CIDEr- en BLEU-scores haalt op de MSCOCO dataset dan de Neural Image Caption Generator van Vinyals et al. [94].

2.3.3 Neuro-evolutie in reinforcement learning

Reinforcement-learning-taken hebben vaak geen eenvoudig uitdrukbare doelfuncties. In de plaats daarvan moeten modellen kunnen leren uit bekomen beloningen. RL reikt verschillende methoden aan, zoals policy gradients, om dat leren mogelijk te maken. Echter, ook evolutionaire algoritmes leggen geen voorwaarden op aan de herkomst van de fitness van de individuen in hun populaties. De beloningen uit RL-taken kunnen dus dienen als fitness voor kandidaatoplossingen of individuen in EA's. Door deze overeenkomst wordt neuro-evolutie vaak gebruikt voor RL-taken.

Recent werden twee EA's voorgesteld voor het trainen van netwerken voor RL-taken met een tot dan toe in neuro-evolutie ongezien aantal parameters. De goede resultaten van deze algoritmes motiveerde onder andere het onderzoek in deze thesis. De algoritmes die deze thesis voorstelt in hoofdstuk 3, en test en bespreekt in hoofdstuk 4 zijn gebaseerd op de algoritmes in [75] en [83]. Daarom beschrijft deze sectie de bijdrage van dat onderzoek.

In [46] wordt een veiligere variant op de mutatie-operator in [75] en [83] voorgesteld die rekening houdt met de sensitiviteit de parameters. Deze techniek wordt ook gebruikt in hoofdstuk 3 en wordt daarom besproken aan het einde van dit onderdeel.

Natuurlijke evolutiestrategie

Salimans et al. ontwikkelen in [75] een variant van een *Natural Evolution Strategy* (NES) [96] om de gewichten te zoeken voor neurale netwerken met een vaste topologie. De netwerken implementeren een policy-functie voor RL-taken. Het genotype is $\gamma = \theta$. Ze gebruiken de NES als alternatief voor RL-algoritmes waarbij gradient descent gebruikt wordt zoals policy gradients. Een NES is, net als een evolutionair algoritme, een black-box-algoritme. Een NES heeft een structuur die afwijkt van het

2. LITERATUURSTUDIE

schema in figuur 2.9 en is dus volgens Eiben en Smith strikt genomen geen EA. Er zijn echter toch ook gelijkenissen.

Algoritme. Een ES zou een populatie van punten $\gamma^{(i)}$ in de genotype-ruimte, i.e. parameters $\theta^{(i)}$ die een policy-functie representeren, bijhouden en evolueren. In plaats daarvan stelt een NES de populatie voor als een kansverdeling $p_\psi(\gamma; \psi)$ over punten in de genotype-ruimte die zelf afhankelijk is van andere parameters ψ . Voor een te maximaliseren evaluatiefunctie $F(\gamma)$ zoekt het algoritme dan de waarden voor ψ die $\mathbb{E}_{\gamma \sim p_\psi}[F(\gamma)]$ maximaliseren. Dat doet het door in elke iteratie een aantal punten te samplen van p_ψ en de waarde van de evaluatiefunctie, of de fitness, in elk van de gesamplede punten $\hat{\gamma}$ te berekenen. Zo kan het algoritme de lokale structuur van de zoekruimte schatten. Uit de berekende fitness-waarden berekent de NES dan een schatter voor de gradiënt van de verwachte fitness ten opzichte van parameters ψ . Vervolgens updatet het algoritme ψ met een stap in de richting van de gradiënt [75, 96].

Meer bepaald kan de gradiënt geschat worden door:

$$\nabla_\psi \mathbb{E}_{\gamma \sim p_\psi}[F(\gamma)] = \mathbb{E}_{\gamma \sim p_\psi}[F(\gamma) \nabla_\psi \log p_\psi(\gamma)]. \quad (2.25)$$

Salimans et al. nemen p_ψ als een multivariate normaalverdeling met gemiddelde γ en standaardafwijking σ , een hyperparameter. Ze nemen $\psi = \gamma$ als de enige parameter van p_ψ , dus σ wordt als vast beschouwd en niet als deel van ψ . De gradiënt kan dan geschreven worden als:

$$\nabla_\psi \mathbb{E}_{\gamma \sim p_\psi}[F(\gamma)] = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim N(0, I)}[F(\gamma + \sigma\epsilon)\epsilon]. \quad (2.26)$$

In elke iteratie t neemt het algoritme λ keer een sample $\epsilon^{(i)} \sim N(0, I)$ en berekent het voor elke sample $F^{(i)}(\gamma_t + \sigma\epsilon^{(i)})$. Deze combineert het tot een schatter voor de gradiënt, en vervolgens doet het een update met een gradient-descent-algoritme:

$$\gamma_{t+1} \leftarrow \gamma_t + \eta \frac{1}{n\sigma} \sum_{i=1}^{\lambda} F^{(i)} \epsilon^{(i)}, \quad (2.27)$$

Met η de learning rate. Lehman et al. merken in [45] op dat de NES een soort van policy-gradient-methode toepast op de parameterruimte in plaats van op de toestandsruimte zoals gekende PG-methoden. Daarom wordt de verdeling van de parameters geoptimaliseerd, en niet de verdeling van acties, om de verwachte fitness te maximaliseren.

Paralleliseerbaarheid en complexiteit. Een voordeel van dit algoritme is dat het zeer paralleliseerbaar is: meerdere *workers* kunnen parallel $\epsilon^{(i)}$ samplen en $F^{(i)}$ berekenen. Het berekenen van de $F^{(i)}$ kost het meeste tijd. Daarvoor wordt de policy-functie, geïmplementeerd door de parameters in γ_t , gebruikt voor een rollout in de RL-taak in kwestie, waaruit een fitness-waarde volgt. In hun experimenten betekent dit bijvoorbeeld dat het netwerk een level van een Atari-spel speelt, wat

verschillende forward passes door het netwerk vereist. De fitness is dan de totale beloning verkregen tijdens het level in het spel. Buiten de scalaire fitness-waarden en de noise-vectoren ϵ , die in de praktijk ook door *random seeds* vervangen kunnen worden, moeten de workers niets met elkaar communiceren. Bijgevolg kunnen de workers relatief onafhankelijk van elkaar te werk gaan met weinig overhead en is er weinig bandbreedte nodig voor communicatie.

Het algoritme heeft slechts één fitness-waarde nodig per evaluatie van een policy, en heeft bijgevolg geen moeilijkheden met ijle beloningen of het toewijzen van de beloningen aan de individuele genomen acties, zoals veel RL-methoden wel hebben [86]. Hoewel er wel een gradiënt geschat wordt, moet het algoritme geen analytische gradiënten berekenen en terugpropageren zoals bij gradient descent, wat rekenintensief is en vereist dat de doelfunctie analytisch differentieerbaar is. Bovendien vermelden ze dat hun NES beter exploratiegedrag vertoont.

In hun experimenten halen ze op bv. verschillende Atari-spellen een hogere score en op andere een lagere dan A3C uit [57], een policy-gradient-methode, met evenveel rekentijd. Het netwerk dat ze trainen voor Atari-spellen nemen ze over van [57] en heeft $\mathcal{O}(10^6)$ parameters. Ze laten zien dat de totale nodige tijd drastisch gereduceerd kan worden door tot 1440 CPU-cores in parallel te gebruiken.

Het schatten van de gradiënt. Zoals Salimans et al. suggereren, kan het rechterlid in (2.26) gezien worden als een soort stochastische eindige differentieschatter in een hoogdimensionale ruimte voor de gradiënt, zeker als σ klein is. Dat lijkt te suggereren dat de methode slecht zou schalen met $\dim(\gamma)$. Salimans et al. weerleggen dit met het argument dat $\dim(\gamma)$ slechts de dimensie van de oplossingsmethode is en dat dat niet hetzelfde is als de inherente dimensie van het optimalisatieprobleem. Ze beweren dat dit laatste is wat de moeilijkheid van een probleem bepaalt.

Zhang et al. onderzoeken de relatie tussen SGD en de NES van Salimans et al. in [104] voor supervised learning: classificatie van afbeeldingen van handgeschreven cijfers uit de MNIST dataset [43]. Ze ondervinden dat voor een netwerk met 3M parameters de correlatie tussen de door de NES geschatte gradiënt met $\lambda = 10^4$ en de gradiënt die SGD berekent slechts %3.9 is. Toch verschilt de nauwkeurigheid op de testset van een netwerk getraind met NES slechts met < %2 van een netwerk dat getraind is met SGD (%97.0 vs. %98.7, in hetzelfde aantal iteraties). Ze suggereren dat gelijkaardige algoritmes ook op complexere taken goede resultaten kunnen halen, maar laten dit voor verder onderzoek.

Ook argumenteren ze dat indien in een iteratie voor elke berekening van een $F^{(i)}$ een andere minibatch uit \mathcal{D} genomen wordt, de schatter voor de gradiënt nauwkeuriger is omdat deze volgt uit een grotere fractie van de dataset. Ze rapporteren inderdaad een kleine maar significante verbetering met een NES die per $F^{(i)}$ een nieuwe minibatch neemt, ten opzichte van één gelijke minibatch per iteratie.

Lehman et al. verschaffen andere inzichten in [45], waar ze de relatie tussen de NES en eindige differenties bespreken. Eindige differenties maken zo klein mogelijke

2. LITERATUURSTUDIE

perturbaties aan variabelen om de gradiënt in één punt in de γ -ruimte te benaderen en optimaliseren dus de evaluatiefunctie voor één parametervector. NES daarentegen optimaliseert de verwachte fitness van een gehele kansverdeling p_ψ , dus van elke sample $\hat{\gamma}$ van p_ψ . Ze argumenteren dat daardoor NES de verdeling stuurt naar regio's in de γ -ruimte die robuuster zijn tegenover perturbaties aan de parameters, zeker voor voldoende grote σ .

Evolutiestrategie

Geïnspireerd door de resultaten van de NES van Salimans et al. onderzoeken Such et al. in [83] of een eenvoudige evolutiestrategie, die geen gradiënten volgt, eveneens een alternatief zou kunnen bieden voor het trainen van neurale netwerken voor RL-taken (met een vaste topologie). Hoewel ze hun algoritme een “genetisch algoritme” noemen wordt in de literatuur die term doorgaans gereserveerd voor EA's die opereren op genotypes van bits of gehele getallen [19].

Algoritme. Het algoritme dat voorgesteld wordt, past in het kader van EA's zoals in sectie 2.3.1. Hoewel ze andere terminologie gebruiken, is hun algoritme een ES-variant waarbij het genotype de vectorvoorstelling van alle parameters van het NN is: $\gamma = \theta$ (of een gecomprimeerde variant hiervan, cfr. verder). De fitness van een individu is de beloning die het krijgt tijdens een rollout in de RL-taak in kwestie, bv. een level van een Atari-spel. Het algoritme houdt een populatie van μ individuen bij. Ouders worden uit de populatie gekozen met uniforme selectie. De gebruikte mutatie-operator is $\gamma_{t+1} = \gamma_t + \sigma\epsilon$ met $\epsilon \sim N(0, I)$, en ze gebruiken geen recombinatie. De volgende generatie wordt bepaald door (μ, λ) -selectie met elitisme. Algoritme 2.1 toont de pseudocode voor deze ES.

Such et al. introduceren eveneens een gecomprimeerde representatie van de parameters, namelijk een lijst van random seeds waaruit de gewichten geëvolueerd zijn. Dit is nuttig om parametervectoren te communiceren over lage bandbreedte en dus parallelisatie verder te faciliteren. Een mutatie (zoals aan het begin van deze sectie gedefinieerd) op de parametervector kan dan eenvoudig gemodelleerd worden: de random seed voor de pseudo-random generator waaruit $\epsilon \sim N(0, I)$ gesampled is toevoegen aan de lijst van seeds die het individu voorstelt. De vectorrepresentatie van het individu kan gereconstrueerd worden uit de random seeds.

Op gelijkaardige wijze als de NES van Salimans et al. is ook de ES van Such et al. zeer paralleliseerbaar. Bovendien heeft de ES helemaal geen gradiënten nodig, geen analytische maar ook geen geschatte, en komt dit de rekenkost verder ten goede.

Experimenten. Deze ES testen ze op RL-benchmarks zoals Atari-spellen en het sturen van motorfuncties van virtuele robots in de MuJoCo-emulator [88]. De architectuur van het netwerk dat ze trainen voor Atari-spellen nemen ze over uit [56] en heeft 4M parameters. Als populatiegrootte en aantal nakomelingen nemen ze $\mu = 20$, $\lambda = 1000$. Na trainen met een vergelijkbaar computationeel budget halen netwerken waarvan de parameters met de ES gezocht zijn, competitieve scores: op

```

1: Invoer: subprocedures muteer, init_individu, evalueer, evalueer_extra,
2:           populatiegrootte  $\mu$ , aantal nakomelingen  $\lambda$ 
3:  $elite \leftarrow \emptyset$ 
4: for  $g = 1 \dots G$  generaties do
5:   # Nakomelingen & evaluatie
6:   for  $i = 1 \dots \lambda$  nakomelingen do
7:     if  $g = 1$  then
8:        $ind \leftarrow init\_individu()$ 
9:     else
10:      # Selectie & mutatie ouders
11:       $ouder \leftarrow Populatie[random\_int(1, \mu)]$ 
12:       $ind \leftarrow muteer(ouder)$ 
13:    end if
14:     $Nakom[i] \leftarrow \{ ind, evalueer(ind, Batch) \}$ 
15:  end for
16:
17:  # Selectie overlevenden
18:   $Populatie \leftarrow sorteer\_hoogste\_score\_eerst(Nakom)[1 \dots \mu, 1]$ 
19:
20:  # Elitisme
21:  if  $g = 1$  then  $Kands \leftarrow Populatie[1 \dots 10]$ 
22:  else  $Kands \leftarrow [elite] + Populatie[1 \dots 9]$ 
23:  end if
24:  for  $i = 1 \dots 10$  do
25:    # Evalueer elite kandidaten op extra aantal episodes
26:     $KandScores[i] \leftarrow \{ Kands[i], evalueer\_extra(Kands[i]) \}$ 
27:  end for
28:   $elite \leftarrow sorteer\_hoogste\_score\_eerst(KandScores)[1, 1]$ 
29:   $Populatie \leftarrow [elite] + (Populatie - \{ elite \})$ 
30: end for
31: return  $elite$ 

```

Algoritme 2.1: Evolutiestrategie van Such et al.

2. LITERATUURSTUDIE

enkele Atari-spellen beter, op enkele slechter dan de NES uit de vorige sectie, A3C en DQN [75, 57, 56]. Ze vermelden dat de ES vaak al na een zeer klein aantal generaties hoog scorende individuen vindt in vergelijking met de andere methoden, een observatie die in overeenstemming is met wat Eiben en Smith schrijven [19]. Voor het besturen van virtuele robots echter ondervinden ze dat de ES veel trager convergeert dan de NES van Salimans et al.

Ze besluiten dat het succes van de ES betekent dat de zoekruimte van sommige RL-problemen toch niet zo vatbaar is voor doorzoeken met gradiënten. Nu ze hebben aangetoond dat netwerken van deze omvang met EA's getraind kunnen worden, vormen EA's een mogelijk alternatief voor gradient descent.

Veilige mutaties

Om het gebruik van EA's voor het trainen van grote en diepe netwerken met vaste topologie verder mogelijk te maken, introduceren en analyseren Lehman et al. enkele slimmere mutatie-operatoren [46].

Sensitiviteit. Ze argumenteren dat een van de factoren die het moeilijk maakt om de parameters van diepe of recurrent NNs te evolueren een mogelijk verschil in sensitiviteit van verschillende parameters is. Een perturbatie aan een gewicht in een van de eerste lagen van een diep netwerk propageert nog door alle daaropvolgende lagen en heeft mogelijk een enorme impact op de uitvoer, en dus de functionaliteit. Een eenvoudige mutatie zoals $\gamma_{t+1} = \gamma_t + \sigma\epsilon$ met $\epsilon \sim N(0, I)$ muteert echter alle parameters met dezelfde standaardafwijking, ongeacht de schaal van hun impact op de functionaliteit van het netwerk. Lehman et al. stellen voor om de grootte van een mutatie aan een parameter afhankelijk te maken van de grootte van de verandering in de uitvoer die een infinitesimaal kleine verandering aan die parameter veroorzaakt.

De destructiviteit van mutaties inperken is reeds een thema in de EA-theorie: de 1/5-succes regel van Rechenberg [68] bijvoorbeeld past σ zo aan dat één op vijf nakomelingen een hogere fitness heeft dan zijn ouder(s). Dat idee is gerelateerd aan het principe van zelf-adaptie (sectie 2.3.1) zoals bij CMA-ES [28]. De methode die hier voorgesteld wordt, is echter explicieter: de variatie in mutatiegrootte wordt niet aan evolutie overgelaten maar analytisch berekend en in de mutatie-operator opgenomen. Ook in deep learning wordt de notie van sensitiviteit al gebruikt: geavanceerde optimalisatiealgoritmes zoals Adam [37] die voor gradient descent worden gebruikt, zetten kleinere stapjes in θ_i waar de gradiënt van de lossfunctie groot is, omdat daar de sensitiviteit hoog is. Het verschil tussen zowel Adam als zelf-adaptie en de veilige mutaties in [46] is dat Lehman et al. voorstellen om de sensitiviteit te meten ten opzichte van de functie f_θ die het netwerk implementeert, en niet ten opzichte van de evaluatie- of lossfunctie.

Methode. De methoden voor veilige mutaties zijn gebaseerd op de assumptie dat f_θ berekenen (een forward pass, bv. 1x acties berekenen uit invoer) goedkoop is in vergelijking met evaluatie in de RL-taak (een rollout, bv. een heel level spelen). Voor

de RL-formulering van tekstgeneratie betekent éénmaal f_θ berekenen het voorspellen van een woord. Een hele evaluatie betekent een of meerdere sequenties van woorden genereren.

Voor een genotype $\gamma = \boldsymbol{\theta}$, een mutatievector $\boldsymbol{\delta} = \sigma\boldsymbol{\epsilon}$, en een neuraal netwerk met uitvoer $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$, $f_\theta : \mathbb{R}^n \mapsto \mathbb{R}^m$ kan de afwijking door $\boldsymbol{\delta}$ op een referentiebatch van B invoerpunten genomen worden als:

$$\text{afwijking}(\boldsymbol{\theta}, \boldsymbol{\delta}) = \sum_k^m \sum_i^B f(\mathbf{x}^{(i)}; \boldsymbol{\theta} + \boldsymbol{\delta})_k - f(\mathbf{x}^{(i)}; \boldsymbol{\theta})_k. \quad (2.28)$$

Indien f differentieerbaar is kan deze door zijn eerste-orde Taylorexpansie in de buurt van $\boldsymbol{\theta}$ benaderd worden:

$$\begin{aligned} f(\mathbf{x}^{(i)}; \boldsymbol{\theta} + \boldsymbol{\delta})_k &\approx f(\mathbf{x}^{(i)}; \boldsymbol{\theta})_k + \boldsymbol{\delta} \nabla_{\boldsymbol{\theta}} f(\mathbf{x}^{(i)}; \boldsymbol{\theta})_k \\ \Leftrightarrow f(\mathbf{x}^{(i)}; \boldsymbol{\theta} + \boldsymbol{\delta})_k - f(\mathbf{x}^{(i)}; \boldsymbol{\theta})_k &\approx \boldsymbol{\delta} \nabla_{\boldsymbol{\theta}} f(\mathbf{x}^{(i)}; \boldsymbol{\theta})_k. \end{aligned} \quad (2.29)$$

De grootte van de gradiënt van elk element in de uitvoer t.o.v. elk van de parameters in $\boldsymbol{\theta}$ is dus een benadering voor de sensitiviteit van de uitvoer t.o.v. die parameter. Het tweede deel van de equivalentie suggereert dat de afwijking (linkerlid) gecontroleerd kan worden door $\boldsymbol{\delta}$ te herschalen in functie van $\nabla_{\boldsymbol{\theta}} f_\theta$. De gradiënten van de elementen van de uitvoer voor de verschillende datapunten aggregeren levert een vector met een sensitiviteit per parameter op:

$$\mathbf{s} = \frac{1}{B} \sqrt{\sum_k^m \left(\sum_i^B \nabla_{\boldsymbol{\theta}} f(\mathbf{x}^{(i)}; \boldsymbol{\theta})_k \right)^2}. \quad (2.30)$$

Het verschil in uitvoer kan dan gecontroleerd worden door $\boldsymbol{\delta}$ te schalen met \mathbf{s} : $\boldsymbol{\delta}_{new} = \boldsymbol{\delta}/\mathbf{s}$, en vervolgens $\boldsymbol{\delta}_{new}$ als mutatievector te gebruiken [46].

Samengevat bestaat hun voorgestelde methode eruit telkens voor het toepassen van de mutatie-operator eerst voor een batch van datapunten of ervaringen $\nabla_{\boldsymbol{\theta}} f_\theta$ en \mathbf{s} te berekenen en dan de herschaalde mutatie $\boldsymbol{\delta}/\mathbf{s}$ toe te passen. In het vervolg van de thesis zal dit principe “veilige mutaties door uitvoergradiënten” genoemd worden. De batch van datapunten kan een kleine referentiebatch zijn. De gradiënt $\nabla_{\boldsymbol{\theta}} f_\theta$ moet voor elk gemuteerd individu éénmaal berekend worden, dus bv. in een ES met (μ, λ) -selectie per generatie één keer voor elke ouder. In de praktijk wordt $\nabla_{\boldsymbol{\theta}} f_\theta$ met backpropagation berekend.

Resultaten. Ze doen experimenten met zowel feedforward als recurrent NNs met tot $\mathcal{O}(10^6)$ parameters. Ze rapporteren dat vooral de EA's met veilige mutaties door gradiënten de taken consistent sneller en beter oplossen. Zo tonen ze aan dat zulke grote netwerken evolueren, waar neuro-evolutie het tot recent moeilijk mee had zonder indirect encodings, verder mogelijk gemaakt wordt door de veilige mutaties.

Ze concluderen dat de veilige mutaties door gradiënten een mogelijkheid bieden waarop nuttige informatie uit gradiënten toch gebruikt kan worden in neuro-evolutie.

2. LITERATUURSTUDIE

Het berekenen van gradiënten heeft echter een rekenkost en neuro-evolutie kan net een interessant alternatief voor gradient descent zijn omdat gradiënten vermeden kunnen worden. Maar Lehman et al. argumenteren dat de veilige mutaties de gradiënt van f_θ gebruiken, niet van $J(\boldsymbol{\theta})$ of $F(\boldsymbol{\theta})$, en dat dit minder duur kan zijn. Ook worden de gradiënten niet gebruikt om de exploratie van het algoritme te sturen, maar enkel om te zorgen dat de exploratie zo veilig en dus efficiënt mogelijk verloopt. Als een groot deel van de mutaties destructief zijn doet dit afbreuk aan de voordelen van parallelisatie: veel rekenwerk gebeurt dan voor niets.

Hoofdstuk 3

Evolutiestrategieën voor het ondertitelen van afbeeldingen met neurale netwerken

Zoals in sectie 2.2.3 in de literatuurstudie besproken is tekstgeneratie al met succes geformuleerd als reinforcement-learning-probleem om discrepancies tussen het trainen en inferentie te verhelpen. Meer bepaald, om rechtstreeks voor niet-differentieerbare evaluatiefuncties te kunnen optimaliseren en als oplossing voor het exposure bias probleem. Na deze formulatie als RL-probleem worden modellen doorgaans getraind met policy gradients en gradient descent. Sectie 2.3.1 behandelde vervolgens evolutionaire algoritmes en hoe deze met succes als alternatief voor gradient descent gebruikt zijn om diepe neurale netwerken te trainen voor RL-toepassingen zoals het spelen van Atari-spellen. Onderzoek zoals in [83] en [75] toont aan dat EA's met de tegenwoordig beschikbare rekenkracht in staat zijn om netwerken met $\mathcal{O}(10^6)$ parameters te trainen en state-of-the-art resultaten te halen.

Het feit dat tekstgeneratie dus met succes als RL-probleem is geformuleerd, en EA's al met succes zijn gebruikt voor RL-taken, suggerert dat ook de combinatie van de twee goede resultaten zou kunnen opleveren. Een logische verderzetting van de redenering is dus eerst tekstgeneratie formuleren als RL-taak zoals in 2.2.3 en dan EA's zoals in 2.3.1 gebruiken om de parameters van een neurale netwerk te evolueren voor deze formulatie van tekstgeneratie.

In lijn met deze motivering worden in dit hoofdstuk twee algoritmes geïntroduceerd voor het trainen van neurale netwerken om afbeeldingen te ondertitelen. Het eerste algoritme, NIC-ES, is een evolutiestrategie gebaseerd op [83]. Het tweede, NIC-NES, is een versie van de NES uit [75].

3. EVOLUTIESTRATEGIEËN VOOR HET ONDERTITELEN VAN AFBEELDINGEN MET NEURALE NETWERKEN

3.1 NIC-ES: een evolutiestrategie voor IC met neurale netwerken

Algoritme 3.1 toont de pseudocode voor de structuur van NIC-ES of *Neural Image Captioning - Evolution Strategy*: het eerste algoritme dat deze thesis voorstelt. NIC-ES is een evolutiestrategie gebaseerd op het algoritme dat Such et al. voorstellen in [83]. In sectie 2.3.3 in de literatuurstudie wordt de ES van Such et al. in meer detail besproken. De volgende subsecties bespreken en motiveren de ontwerpbeslissingen en de wijze waarop het algoritme de nodige concepten uit 2.3.1 invult.

Het belangrijkste verschil met [83] is dat NIC-ES per generatie een nieuwe minibatch neemt uit \mathcal{D} . Alle geëvolueerde nakomelingen voor de volgende generatie worden geëvalueerd op die minibatch, omdat elke nakomeling op heel de dataset evalueren te duur zou zijn.

De evaluatieprocedure, de fenotypes en de dataset zijn de enige domeinspecifieke componenten in NIC-ES: de fenotypes zijn LSTM-netwerken en NIC-ES berekent voor de evaluatie CIDEr-scores. Het algoritme kan door deze componenten aan te passen gemakkelijk gebruikt worden om parametrische modellen te trainen met minibatches voor andere taken dan IC.

De pseudocode in algoritme 3.2 laat zien hoe dit algoritme eenvoudig geïmplementeerd kan worden in een parallelle master-worker setting. Eén masterproces beheert de populatie en voert boekhoudkundige taken uit. De master publiceert taken, i.e. de huidige populatie en minibatch uit de dataset, en wacht dan op resultaten de workers: geëvolueerde parameters en bijhorende fitness-scores. Het aantal workers ligt niet vast. Ze doen het meest rekenintensieve werk: de evaluatie van individuen, wat meerdere forward passes door het neurale netwerk vraagt. Omdat het werk van de master in verhouding klein is, levert het aantal workers verhogen een bijna lineaire reductie van de rekentijd in elke generatie. Tot 92 workers, elk op een CPU-core, werden gebruikt voor de experimenten in hoofdstuk 4.

De implementatie van het algoritme in Python is voorzien om op eenzelfde fysieke of virtuele machine uitgevoerd te worden. Het zou echter slechts kleine aanpassingen vragen om het mogelijk te maken om NIC-ES op gedistribueerde hardware uit te voeren. De master en elke worker zouden dan bv. op een eigen machine kunnen lopen. Een deel van de communicatie tussen de master en workers verloopt in de huidige implementatie om de geheugendruk te beperken via de harde schijf van de machine. Dat deel zou in een gedistribueerde context dus aangepast moeten worden. Ook zouden andere overwegingen dan mogelijks belangrijker worden: terwijl de implementatie nu vooral geheugen- en rekendruk probeert te beperken zou in een gedistribueerde setting de nodige bandbreedte bijvoorbeeld een belangrijkere factor worden.

3.1.1 Individuen en representatie

Om uit een afbeelding een beschrijvende zin te genereren wordt het encoder-decoder-framework gebruikt met een CNN als encoder en een LSTM als decoder. De opstelling

Algoritme 3.1: NIC-ES: evolutiestrategie voor IC

```

1: Invoer: subprocedures muteer, init_individu, evalueer, selecteer_ouder
2:          populatiegrootte  $\mu$ , aantal nakomelingen  $\lambda$ 
3:          aantal elite kandidaten  $C$ , aantal elite individuen  $E$ 
4:          training dataset van afbeeldingen & ground-truth-zinnen  $\mathcal{D}$ 
5:          validatie dataset  $\mathcal{D}_{val}$ 
6:
7: for  $g = 1 \dots G$  generaties do
8:      $Batch \leftarrow$  willekeurige minibatch van grootte  $B$  uit  $\mathcal{D}$ 
9:
10:    # Nakomelingen & evaluatie
11:    for  $i = 1 \dots \lambda$  nakomelingen do
12:        if  $g = 1$  then
13:             $ind \leftarrow$  init_individu()
14:        else
15:            # Selectie & mutatie ouders
16:             $ouder \leftarrow$  selecteer_ouder(Populatie)
17:             $ind \leftarrow$  muteer(ouder)
18:        end if
19:         $Nakom[i] \leftarrow \{ ind, evalueer(ind, Batch) \}$ 
20:    end for
21:
22:    # Selectie overlevenden
23:    Populatie  $\leftarrow$  sorteer_hoogste_score_eerst(Nakom) $[1 \dots \mu, 1]$ 
24:
25:    # Elitisme
26:    for  $i = 1 \dots C$  do
27:        # Evalueer elite kandidaten op validatie dataset
28:         $KandScores[i] \leftarrow \{ Populatie[i], evalueer(Populatie[i], \mathcal{D}_{val}) \}$ 
29:    end for
30:     $Elites \leftarrow$  sorteer_hoogste_score_eerst(KandScores + Elites) $[1 \dots E]$ 
31:    Populatie  $\leftarrow$  Elites + Populatie
32: end for
33: return Elites $[1, 1]$ 
```

3. EVOLUTIESTRATEGIEËN VOOR HET ONDERTITELEN VAN AFBEELDINGEN MET NEURALE NETWERKEN

Algoritme 3.2: NIC-ES: parallelle versie

```

1: procedure MASTER
2:    $Populatie \leftarrow []$ ,  $Kands \leftarrow [\text{init\_individu}()]$ 
3:   for  $g = 1 \dots G$  generaties do
4:      $Batch \leftarrow$  willekeurige minibatch van grootte  $B$  uit  $\mathcal{D}$ 
5:     publiceer_taak( $Populatie, Batch, Kands$ )
6:
7:     while  $\text{length}(Nakom) < \lambda$  and  $\text{length}(KandScores) < C$  do
8:       # wacht op resultaat
9:        $res \leftarrow \text{neem\_gepubliceerd\_resultaat}()$ 
10:      if  $res$  is evolutie_resultaat then
11:         $Nakom \leftarrow Nakom + [\{res[1], res[2]\}]$ 
12:      else if  $res$  is elite_evaluatie_resultaat then
13:         $KandScores \leftarrow Nakom + [\{res[1], res[2]\}]$ 
14:      end if
15:    end while
16:
17:     $Populatie \leftarrow \text{sorteer\_hoogste\_score\_eerst}(Nakom)[1 \dots \mu, 1]$ 
18:     $Elites \leftarrow \text{sorteer\_hoogste\_score\_eerst}(KandScores + Elites)[1 \dots E]$ 
19:     $Kands \leftarrow Populatie[1 \dots C]$ 
20:     $Populatie \leftarrow Elites + Populatie$ 
21:     $Nakom \leftarrow []$ ,  $KandScores \leftarrow []$ 
22:  end for
23:  return  $Elites[1, 1]$ 
24: end procedure
25:
26: procedure WORKER
27:   while  $True$  do
28:      $Populatie, Batch, Kands \leftarrow \text{neem\_gepubliceerde\_taak}()$ 
29:     if  $\text{random\_float}() > \text{evaluatieKans}$  then
30:       if  $g = 1$  then  $ind \leftarrow \text{init\_individu}()$ 
31:       else  $ind \leftarrow \text{muteer}(\text{selecteer\_ouder}(Populatie))$ 
32:       end if
33:       publiceer_resultaat(  $\{ind, \text{evalueer}(ind, Batch)\}$  )
34:     else
35:        $kand \leftarrow \text{selecteer\_kandidaat}(Kands)$ 
36:       publiceer_resultaat(  $\{kand, \text{evalueer}(kand, D_{val})\}$  )
37:     end if
38:   end while
39: end procedure

```

wordt grotendeels overgenomen uit [70] en is gebaseerd op de de Neural Image Caption Generator uit [94], zoals beschreven in sectie 2.2.2.

Het gebruikte CNN is een ResNet-101 [30] dat voorgetraind is voor objectdetectie met Faster R-CNN op de Visual Genome dataset [69, 39], zoals bij Anderson et al. [3]. Dit CNN produceert vectoren \mathbf{q}_{img} van vaste grootte met abstracte features uit afbeeldingen. In Faster R-CNN worden deze features gebruikt om regio's in de afbeelding met herkenbare objecten te zoeken. Hier worden ze echter als geheel als invoer aan de LSTM gegeven. De keuze voor deze voortraining van het CNN werd gemaakt omdat de voorverwerkte data eenvoudig voorhanden was en omdat hier goede resultaten mee gehaald waren [3]. De parameters van het CNN worden zoals in [70] niet meer aangepast tijdens het trainen in deze thesis: ze zitten niet in het genotype.

De parameters van de volgende componenten maken wel deel uit van het genotype. Figuur 3.1 geeft een overzicht van de opstelling en de gezochte parameters.

- Een lineaire laag $f^{(img)} : \mathbb{R}^{\dim(\mathbf{q}_{img})} \mapsto \mathbb{R}^{\dim(\mathbf{w}_e)}$ met parameters $\boldsymbol{\theta}_{img}$ die de afbeelding-features \mathbf{q}_{img} van het CNN naar vectoren van dezelfde grootte als de word embeddings transformeert.
- De gewichten van de word embeddinglaag \mathbf{W}_e : i.e. V word embeddings van $\dim(\mathbf{w}_e)$ dimensies.
- De LSTM zelf, met verborgen toestandsvectoren van $\dim(\mathbf{h})$ dimensies.
- Een lineaire laag $f^{(out)} : \mathbb{R}^{\dim(\mathbf{h})} \mapsto \mathbb{R}^V$ die uit de uitvoer van de LSTM een vector van V dimensies berekent met een score per woord in \mathcal{V} .

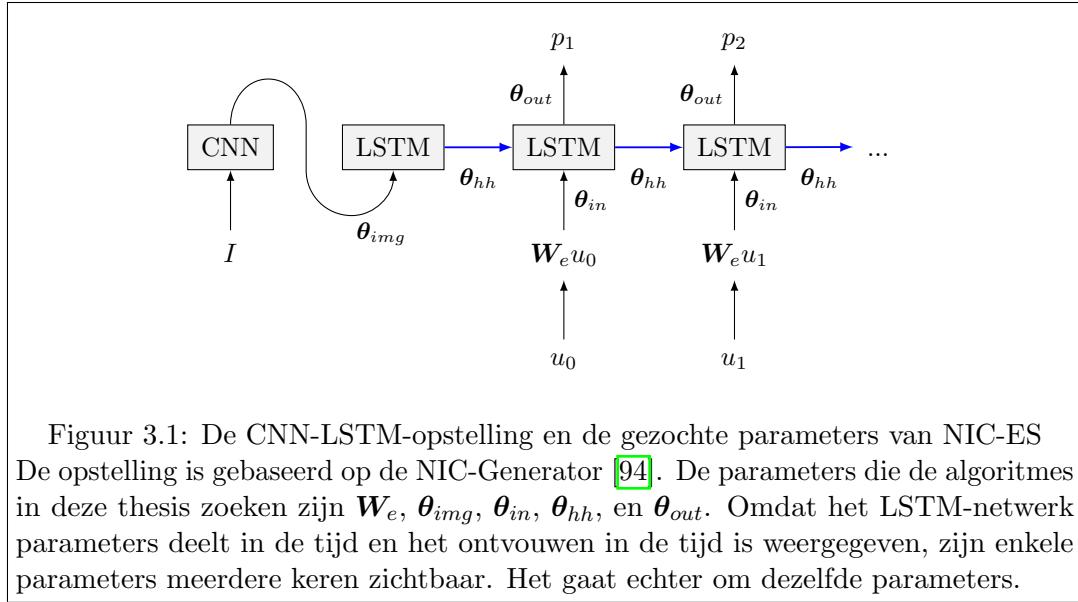
Het genotype voor de ES is een vlakke vectorrepresentatie van deze parameters $\gamma = (\mathbf{W}_e, \boldsymbol{\theta}_{img}, \boldsymbol{\theta}_{in}, \boldsymbol{\theta}_{hh}, \boldsymbol{\theta}_{out})$. Dat genotype is gelijk voor beide algoritmes in deze thesis. Vanaf nu zal met $\boldsymbol{\theta}$ het hele genotype γ bedoeld worden. De dimensies van de word embeddings en de verborgen toestanden zijn hyperparameters. Het fenotype is een geïnstantieerd LSTM-netwerk met de hierboven vernoemde componenten, met aan elk van zijn parameters de overeenkomstige waarden uit een γ toegewezen.

Zoals in de literatuurstudie vermeld introduceren Such et al. in [83] een gecomprimeerde representatie voor de individuen om de nodige bandbreedte voor hun algoritme te reduceren. In de eerste versies van NIC-ES werd deze representatie ook gebruikt, maar omdat dat niet nuttig bleek werd de vlakke vectorrepresentatie gebruikt.

3.1.2 Selectiemechanismen

Selectie van volgende generatie. Zoals in [83] wordt voor de selectie van overlevenden (μ, λ) -selectie gebruikt. Dat wil zeggen dat uit een populatie van μ individuen λ nakomelingen worden gegenereerd. Deze nakomelingen worden geëvalueerd op de huidige minibatch en de beste μ vormen de volgende generatie door de vorige μ individuen te vervangen. Dit is dus een selectieschema gebaseerd op leeftijd: individuen zitten voor één generatie in de populatie en worden daarna vervangen

3. EVOLUTIESTRATEGIEËN VOOR HET ONDERTITELEN VAN AFBEELDINGEN MET NEURALE NETWERKEN



ongeacht hun fitness. De overige $\lambda - \mu$ nakomelingen worden verwijderd. Hiervoor is dus een evaluatie van elke nakomeling nodig. Om de relatieve fitness te kunnen hergebruiken wordt de populatie bijgehouden als lijst van individuen, gesorteerd op afnemende fitness.

NIC-ES gebruikt ook elitisme: het beoordeelt de C beste kandidaten uit de vorige generatie op \mathcal{D}_{val} en houdt de E individuen met de beste validatiescore tot dan toe bij. Deze E elites worden in elke generatie aan de populatie toegevoegd om te vermijden dat NIC-ES tijd verdoet met het opnieuw ontdekken van de reeds gevonden beste oplossingen. C en E zijn hyperparameters.

Selectie van ouders. Voor de selectie van ouders uit de populatie kunnen twee tactieken gehanteerd worden: uniforme willekeurige selectie (UWS) zoals in [83] en toernooiselectie (TS) zoals uitgelegd in [19]. Sectie 2.3.1 in de literatuurstudie beschrijft deze mechanismen. De mechanismen worden geïmplementeerd in de “selecteer_ouder”-subprocedure in algoritme 3.1. Welke van de mechanismen NIC-ES gebruikt is een instelling die gekozen kan worden.

Omdat voor het (μ, λ) -schema uit de vorige paragraaf de gepubliceerde populatielijst gesorteerd is op afnemende fitness zijn voor TS geen extra evaluaties nodig. Dat wil zeggen dat de selectie van ouders op basis van dezelfde minibatch gebeurt als de selectie van overlevenden in de huidige generatie.

3.1.3 Variatie-operatoren

Zoals de ES van [83] gebruikt NIC-ES geen recombinatie maar enkel mutaties, omdat deze eenvoudig te implementeren zijn. In algoritme 3.1 worden mutaties geïmplementeerd in de subprocedure “muteer”. De eenvoudigste mutatie (verder

M-STD) is:

$$\boldsymbol{\theta}_{g+1} = \boldsymbol{\theta}_g + \sigma \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim N(0, I). \quad (3.1)$$

De standaardafwijking σ is hierbij een hyperparameter. NIC-ES biedt twee curriculumtactieken die σ dynamisch verlagen. De eerste tactiek, *patience*, houdt in dat NIC-ES σ verlaagt wanneer de tot dan toe beste score op de validatieset gedurende P (een hyperparameter) iteraties niet verbroken wordt. Bij *schedule*, de tweede tactiek, wordt op voorhand een schema gekozen: om de H (hyperparameter) iteraties neemt σ af. Daarbij neemt σ af gedurende de training, om de zoektocht naar $\boldsymbol{\theta}$ in het begin sneller te laten verlopen en om het daarna geleidelijk aan mogelijk te maken om goede gevonden waarden voor $\boldsymbol{\theta}$ te *finetunen*. De factor waarmee σ telkens afneemt is eveneens een hyperparameter.

Buiten de standaardmutatie uit vergelijking (3.1) werd NIC-ES nog uitgerust met twee mutatievarianten die rekening houden met verschillen in sensitiviteit van parameters. Zoals Lehman et al. in [46] opmerken creëert de mutatiegrootte σ een spanning tussen hoe snel evolutie kan plaatsvinden en hoe verstorend mutaties mogen zijn. Indien σ groot genomen wordt valt een groter deel van de $\boldsymbol{\theta}$ -ruimte binnen het bereik van het algoritme en is het algoritme minder vatbaar voor lokale optima of vlakke delen. Bij te grote σ echter bestaat de kans dat de $\boldsymbol{\theta}$ zodanig verstoord worden dat de performantie van individuen alleen maar daalt. Rekening houden met de sensitiviteit van parameters zou deze spanning gedeeltelijk kunnen relaxeren: NIC-ES zou een grotere σ kunnen gebruiken, waardoor evolutie sneller kan verlopen, zonder dat daarom teveel van de mutaties minder fitte nakomelingen voortbrengen. Welke van de mutatievarianten NIC-ES gebruikt is zoals de selectiemechanismen een configurerbare instelling.

Proportionele mutaties. Een intuïtieve en efficiënte manier die NIC-ES kan gebruiken om de mutaties aan te passen aan de gevoeligheid van de parameters is de elementen uit de mutatievector proportioneel maken aan de absolute waarde van de respectievelijke elementen uit de parametervector:

$$\boldsymbol{\theta}_{g+1} = \boldsymbol{\theta}_g + \sigma \boldsymbol{\epsilon} \odot |\boldsymbol{\theta}_g|, \quad \boldsymbol{\epsilon} \sim N(0, I). \quad (3.2)$$

Hierbij is $\boldsymbol{\epsilon} \odot \boldsymbol{\theta}$ het elementsgewijze vectorproduct van $\boldsymbol{\epsilon}$ en $\boldsymbol{\theta}$, en $\text{abs}(\boldsymbol{\theta})$ de elementsgewijze absolute waarde van $\boldsymbol{\theta}$. De onderliggende assumptie die deze mutatievariant (verder VM-P) motiveert is dat relatief kleine parameters tegenover dezelfde verandering $|\delta|$ ook relatief sensitieve parameters zouden kunnen zijn.

Veilige mutaties door sensitiviteit met uitvoergradiënten. Geïnspireerd door [46] ondersteunt NIC-ES ook veilige mutaties door uitvoergradiënten (VM-G), zoals in onderdeel 2.3.3 besproken. Een sensitiviteitsvector kan berekend worden voor een batch \mathcal{B} :

$$\mathbf{s} = \frac{1}{B} \sqrt{\sum_k^V \left(\sum_i^B \nabla_{\boldsymbol{\theta}} f_l(\mathbf{x}^{(i)}; \boldsymbol{\theta})_k \right)^2}. \quad (3.3)$$

3. EVOLUTIESTRATEGIEËN VOOR HET ONDERTITELEN VAN AFBEELDINGEN MET NEURALE NETWERKEN

Met $f_l(\mathbf{x}^{(i)}; \boldsymbol{\theta})$ de uitvoer van het model na l tijdstappen onder greedy decoding. In de experimenten is $l \geq 5$, om variaties in de uitvoer in het midden van een zin te meten en niet bv. in de eerste tijdstap (zal vaak het woord “een” zijn). Vector \mathbf{s} bevat een sensitiviteitswaarde per parameter in $\boldsymbol{\theta}$. Op basis daarvan muteert NIC-ES de parameters als volgt:

$$\boldsymbol{\theta}_{g+1} = \boldsymbol{\theta}_g + \sigma \frac{\boldsymbol{\epsilon}}{\mathbf{s}}, \quad \boldsymbol{\epsilon} \sim N(0, I), \quad (3.4)$$

Met $\boldsymbol{\epsilon}/\mathbf{s}$ het elementsgewijze quotiënt van de twee vectoren. Voor de berekening van $\sum_k (\sum_i \nabla_{\boldsymbol{\theta}} f_l(\mathbf{x}^{(i)}; \boldsymbol{\theta}))_k^2$ in (3.3) is echter weer $\dim(\boldsymbol{\theta})$ van belang. De gradiënt geeft een vector van $\dim(\boldsymbol{\theta})$, per uitvoerelement k van f . Voor de experimenten in hoofdstuk 4 is de uitvoer een verdeling over de woorden in het alfabet en dus $V \approx 10^4$. Voor het berekenen van (3.3) is dus een tijdelijke matrix van $\dim(\boldsymbol{\theta}) \times V$ nodig, en hiervoor was in de experimenten teveel geheugen nodig.

Om de ruimtecomplexiteit te beperken wordt voorgesteld om NIC-ES in de plaats een variant van \mathbf{s} te laten berekenen. Daarvoor wordt een tijdelijke laag toegevoegd aan het netwerk: een functie h die een vector van dimensie 100 (i.p.v. $V \approx 10^4$) berekent waarvan elk element een functie is van 100 elementen uit de uitvoer van f :

$$h(f(\mathbf{x}; \boldsymbol{\theta})) = \left(\left(\sum_{i=1}^{100} f(\mathbf{x}; \boldsymbol{\theta})_i^2 \right)^{\frac{1}{2}}, \dots, \left(\sum_{i=V-100}^V f(\mathbf{x}; \boldsymbol{\theta})_i^2 \right)^{\frac{1}{2}} \right). \quad (3.5)$$

De berekening van \mathbf{s} verloopt dan volgens (3.3) maar met $h \circ f$ in plaats van f : zo is de grootte van de nodige tijdelijke matrix 100 keer kleiner.

NIC-ES berekent deze sensitiviteit $1 \times$ per ouder, $1 \times$ per generatie, op de huidige minibatch of een deel ervan. Elke berekening van een sensitiviteitsvector vraagt de berekening van l tijdstappen van het model voor B afbeeldingen, en vervolgens backpropagation through time door de l stappen om de gradiënt te berekenen. Het is ook mogelijk om een vaste, op voorhand berekende sensitiviteitsvector door te geven die NIC-ES dan voor alle ouders in alle generaties zal gebruiken. Zo wordt de rekenkost van telkens \mathbf{s} te berekenen uitgespaard. Deze vaste sensitiviteitsvector kan bijvoorbeeld de sensitiviteit van een referentie-individu zijn, berekend op de hele dataset in plaats van enkel op een kleine batch.

3.1.4 Initialisatie

Eiben en Smith suggereren in [19] dat intelligente heuristieken om individuen te initialiseren mogelijks niet de moeite waard zijn omwille van de doorgaans relatief snelle stijging van de fitness van individuen tijdens de eerste generaties. Daarom kan NIC-ES vertrekken van een populatie van μ onafhankelijk willekeurig geïnitialiseerde individuen. Daarvoor initialiseert NIC-ES de parameters met Xavier initialisatie [23] maar zet alle bias-parameters op 0, zoals in [83]. In algoritme 3.1 gebeurt dat in de subprocedure “init_individu”.

Echter, zoals Ranzato et al. opmerken in [65] en zoals besproken in sectie 2.2.3, groeit de zoekruimte voor algoritmes die een hele sequentie genereren en aan het einde

pas beoordeeld worden exponentieel in de sequentielengte. Daardoor ondervinden ze dat het niet haalbaar is om een willekeurig geïnitialiseerde policy op sequentieniveau te beginnen trainen, en starten ze het trainen van een policy die met XENT voortgetraind is. Bij het trainen met XENT wordt een loss per woord berekend en niet per zin, wat het aantal uitvoermogelijkheden beperkt tot V in de plaats van V^T . Voortrainen met XENT is dus niet louter een intelligente initialisatieheuristiek maar traint de netwerken voor een effectief verschillend en haalbaarder optimalisatieprobleem. Ook Rennie et al. [70] volgen die aanpak.

Het is dus te verwachten dat ook NIC-ES het moeilijk zal hebben met beginnen van een willekeurig geïnitialiseerde populatie van individuen. Parameters bekomen uit voortraining met XENT gebruiken bij de initialisatie van de populatie zou, zoals Eiben en Smith ook toegeven, het zoekproces kunnen sturen naar regio's in de zoekruimte met goede oplossingen. Het is belangrijk om dit op zo een manier te doen dat er toch voldoende diversiteit voor de evolutie in de populatie aanwezig is [19]. De tactiek die NIC-ES gebruikt om te vertrekken van voortgetrainde parameters θ_{xent} is eenvoudig: de eerste generatie bestaat volledig uit kopieën van θ_{xent} . De mutatiegrootte dient dan in het begin voldoende groot genomen te worden om toch diversiteit te creëren. Om meer diversiteit te ondersteunen is het ook mogelijk de populatie te initialiseren met kopieën van meerdere $\theta_{xent}^{(i)}$, bv. verkregen uit verschillende trainingruns met XENT.

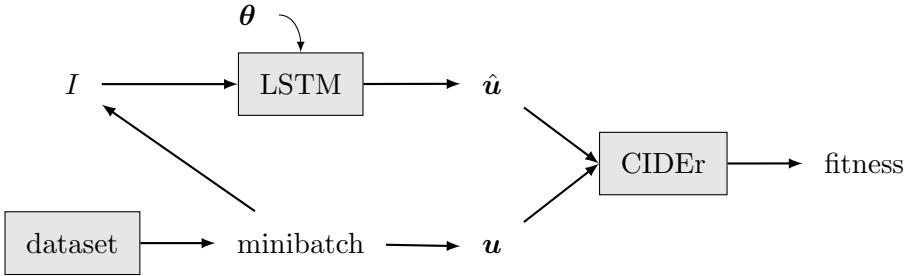
De nood om voortgetrainde parameters te kunnen gebruiken ligt eveneens aan de basis van de beslissing om enkel de parameters en niet de topologie van de netwerken te evolueren.

3.1.5 Evaluatie en fitness

Deze sectie gaat over het evalueren van netwerken, of het berekenen van hun fitness, tijdens het trainen, en niet over evaluatie achteraf op testdata. In elke generatie wordt een nieuwe minibatch \mathcal{B} van afbeeldingen en bijhorende referentiezinnen uit de dataset gesampled. Na het genereren van nakomelingen door mutatie van ouders worden deze nakomelingen geëvalueerd op de huidige minibatch (subprocedure “evalueer” in algoritme 3.1).

De evaluatie van een individu θ op een minibatch van afbeeldingen en zinnen (I, \mathbf{y}) verloopt als volgt. De parameters uit het genotype worden aan de overeenkomstige parameters in een geïnstantieerd LSTM-netwerk met de componenten uit figuur 3.1 toegewezen. Eerst berekent het CNN (met vaste parameters) uit elke afbeelding I de afbeelding-features. De LSTM neemt deze features als invoer en berekent hieruit t keer een kansverdeling $p_{model}(u_t)$ over woorden. In elke tijdstap t neemt NIC-ES een woord uit $p_{model}(u_t)$ en geeft dit woord als invoer aan het model in $t+1$. Zo genereert de LSTM onder het inferentieschema uit sectie 2.2.1 (en figuur 2.5) per afbeelding een zin $\hat{\mathbf{y}}$. Dan berekent NIC-ES de CIDEr-score van de gegenereerde zinnen $\hat{\mathbf{y}}$ t.o.v. referentiezinnen \mathbf{y} en neemt deze score als fitness: $F(\theta) = \text{CIDEr}(\hat{\mathbf{y}}, \mathbf{y})$. De evaluatie van elitekandidaten verloopt identiek, alleen berekent NIC-ES de CIDEr-score op de

3. EVOLUTIESTRATEGIEËN VOOR HET ONDERTITELEN VAN AFBEELDINGEN MET NEURALE NETWERKEN



Figuur 3.2: Evaluatieprocedure van NIC-ES

NIC-ES evolueert individu $\boldsymbol{\theta}$ op minibatch \mathcal{B} bestaande uit (I, \mathbf{y}) uit de dataset. Daarvoor instantieert het een LSTM-netwerk zoals in figuur 3.1 met parameters $\boldsymbol{\theta}$. De LSTM voorspelt voor afbeeldingen I beschrijvende zinnen $\hat{\mathbf{y}}$ en de CIDEr score van $\hat{\mathbf{y}}$ ten opzichte van \mathbf{y} wordt berekend en gebruikt als fitness voor $\boldsymbol{\theta}$. Deze fitness is afhankelijk van de specifieke minibatch.

validatieset en niet op de huidige minibatch. Figuur 3.2 toont een overzicht van de evaluatieprocedure.

Tijdens het trainen wordt dus, net zoals tijdens inferentie, de uitvoer van het model uit de vorige tijdstap als invoer in de huidige tijdstap gebruikt. Daardoor is er geen exposure bias tussen trainen en inferentie: bij teacher forcing wordt namelijk telkens het vorige ground-truth-woord, en niet de uitvoer van het model, als invoer gebruikt.

De grootte van de minibatches is een hyperparameter en volgt uit een trade-off tussen de rekentijd die nodig is voor de evaluaties en hoe statistisch representatief de minibatches zijn voor heel de dataset. Hoe groter B , hoe meer tijd de rollouts kosten, maar ook hoe groter de kans dat een individu dat goed presteert op \mathcal{B} ook goed presteert op heel de dataset. Dan is dus ook de kans kleiner dat door verschillen in minibatches globaal fitte individuen verloren gaan omdat ze toevallig minder goed scoren op de huidige minibatch. NIC-ES kan de grootte van minibatches B dynamisch laten toenemen met patience of een schedule, cfr. sectie 3.1.3.

NIC-ES biedt drie manieren om volgens de beschreven wijze uit $\boldsymbol{\theta}$ een fitness te berekenen. Welke gebruikt wordt is een instelling die de gebruiker dient te specificeren.

- **Greedy decoding:** onder greedy decoding neemt NIC-ES in elke tijdstap $u_t^* = \arg \max_k p_{model}(u_t = k)$ als woord en geeft dit woord als invoer aan het model in $t + 1$. De fitness is gelijk aan $F_{gr}(\boldsymbol{\theta}) = \text{CIDEr}(\mathbf{y}^*, \mathbf{y})$, met $\mathbf{y}^* = (u_1^*, u_2^*, \dots, u_T^*)$ en \mathbf{y} een ground-truth-zin.
- **Samplen:** een alternatief is in elke t een sample nemen van p_{model} : $\hat{u}_t \sim p_{model}(u_t)$, met als fitness $F_{sam}(\boldsymbol{\theta}) = \text{CIDEr}(\hat{\mathbf{y}}, \mathbf{y})$. Door te samplen van p_{model} krijgt NIC-ES mogelijk een beter beeld van de kansverdelingen. Het is over het algemeen voor generalisatie wenselijk dat een model niet alleen de juiste u_t berekent, maar ook dat het model hier “zeker” van is, i.e. dat het een hoge

kans toekent aan u_t . Bij greedy decoding doet de waarde van $p_{model}(u_t^*)$ er niet toe, enkel het feit dat u_t^* de grootste kans krijgt is belangrijk. Samplen van p_{model} in de plaats van greedy decoding zou een manier kunnen zijn om de kansverdeling toch in rekening te brengen.

- **SCST:** een derde optie is, gemotiveerd door de goede resultaten van deze methode met gradient descent, de self-critical beloning die Rennie et al. in hun policy-gradient-algoritme gebruiken [70]:

$$F_{scst}(\boldsymbol{\theta}) = \text{CIDEr}(\hat{\mathbf{y}}, \mathbf{y}) - \text{CIDEr}(\mathbf{y}^*, \mathbf{y}). \quad (3.6)$$

3.1.6 Complexiteit

De tijdscomplexiteit van het masterproces in algoritme 3.2 zal vooral toenemen met het aantal nakomelingen λ dat gesorteerd moet worden, de ruimtecomplexiteit zowel met λ als met $\dim(\boldsymbol{\theta})$. Ook de ruimtecomplexiteit van de workers neemt toe met $\dim(\boldsymbol{\theta})$. De sectie zal in het vervolg aannemen dat het rekenwerk van de master klein is in vergelijking met dat van de workers. Hoewel dat slechts een benadering is, doen de workers wel degelijk het meest rekenintensieve werk: de nakomelingen genereren en vooral evalueren op een minibatch zoals in figuur 3.2. Voor deze rollouts zijn een aantal forward passes door de LSTM en de andere componenten nodig dat proportioneel is aan de sequentielengte en de grootte van de minibatch. NIC-ES met standaardmutaties berekent geen gradiënten en voor de standaardversie van het algoritme zijn dus geen backward passes nodig. De totale hoeveelheid door de workers te verrichten rekenwerk per generatie stijgt met:

- Het aantal datapunten in de minibatches: B . Het gebruikte deep learning-framework PyTorch staat toe de berekeningen voor datapunten in dezelfde minibatch gedeeltelijk te paralleliseren: de rekentijd stijgt dus minder dan proportioneel met B .
- De sequentielengte T : in de praktijk variëren de lengtes van zinnen, maar hoe langer de gemiddelde sequentie is, hoe meer forward passes door het model nodig zijn.
- Het aantal nakomelingen per generatie λ : de totale rekentijd stijgt ongeveer proportioneel met het aantal nakomelingen dat gegenereerd en geëvalueerd moet worden.
- De grootte van de netwerken en het aantal parameters $\dim(\boldsymbol{\theta})$: hoewel de workers voor een grotere $\dim(\boldsymbol{\theta})$ ook een grotere mutatie moeten berekenen, volgt de grootste toename in rekentijd uit de evaluatie en niet de mutatie. Voor een groter aantal parameters zal het te evalueren netwerk namelijk groter zijn, en een forward pass door een netwerk van hetzelfde type maar met meer parameters (dieper, breder,...) vergt over het algemeen meer rekenkracht. Gezien een evaluatie van een individu uit meerdere forward passes bestaat, zal ook de evaluatie een grotere rekenkost hebben.

3. EVOLUTIESTRATEGIEËN VOOR HET ONDERTITELEN VAN AFBEELDINGEN MET NEURALE NETWERKEN

- Het aantal te evalueren elitekandidaten C en de grootte van de validatieset $|\mathcal{D}_{val}| > B$. De rekentijd die een worker nodig heeft voor de evaluatie van één kandidaat op \mathcal{D}_{val} is een ondergrens voor de totale rekentijd van een generatie, ook met meerdere workers.

De totale rekentijd voor het werk van de workers in een generatie daalt bij benadering proportioneel met het aantal gebruikte workerprocessen W zolang $W \ll \lambda$. Zoals vermeld kan NIC-ES voor veilige mutaties toch uitvoergradiënten berekenen. Tenzij er een op voorhand berekende sensitiviteitsvector gebruikt wordt, vraagt dit per generatie voor maximaal μ ouders de berekening van l tijdstappen voor B afbeeldingen en dan BPTT terugwaarts doorheen l tijdstappen.

3.2 NIC-NES: een natuurlijke evolutiestrategie voor IC met neurale netwerken

Het tweede algoritme dat deze thesis voorstelt is NIC-NES of *Neural Image Captioning - Natural Evolution Strategy*, gebaseerd op de natuurlijke evolutiestrategie van Salimans et al. [75]. Het voornaamste verschil met [75] is opnieuw het feit dat NIC-NES de θ 's evalueert op minibatches uit een dataset. Sectie 2.3.3 in de literatuurstudie beschrijft de NES van Salimans et al.

Algoritme 3.3 toont de pseudocode voor NIC-NES. Het algoritme neemt componenten over van het NIC-ES-algoritme uit de vorige sectie: de individuen en representatie, de verschillende mutatievarianten, de initialisatie en de evaluatieprocedure blijven gelijk. NIC-NES verschilt echter ook structureel van NIC-ES: het houdt geen populatie bij maar zoekt een p_ψ zodat $\mathbb{E}_{\theta \sim p_\psi}[F(\theta)]$ maximaal is. NIC-NES updatet p_ψ daarvoor iteratief met gradient ascent door een geschatte gradiënt te volgen. De verdeling p_ψ is een multivariate normaalverdeling met als gemiddelde de huidige θ en een invariante standaardafwijking σ .

Schatter voor de gradiënt. Om de schatter voor de gradiënt te berekenen genereert NIC-NES eerst λ mutatievectoren $\delta^{(i)}$ ten opzichte van de huidige θ . Zoals algoritme 3.3 duidelijk maakt, evalueert NIC-NES dan zowel $\theta + \delta^{(i)}$ als $\theta - \delta^{(i)}$, een techniek die *mirrored sampling* heet [11]. De resulterende fitnesswaarden ondergaan een rangtransformatie en worden genormaliseerd. Het gewicht w_i dat bij $\delta^{(i)}$ hoort, is dan het verschil tussen de genormaliseerde rang van de fitness van $\theta + \delta^{(i)}$ en die van $\theta - \delta^{(i)}$.

Vervolgens berekent NIC-NES θ_{j+1} uit θ_j door een stap van gradient ascent met de berekende gradiënt (subprocedure “gradient_ascent_stap” in algoritme 3.3):

$$\theta_{j+1} \leftarrow \theta_j + \eta \frac{1}{\lambda} \sum_{i=1}^{\lambda} w_i \delta^{(i)}. \quad (3.7)$$

Met η een gekozen learning rate. Naast SGD kunnen ook andere gradient ascent-methoden gebruikt worden om een update van θ te berekenen: in de experimenten

3.2. NIC-NES: een natuurlijke evolutiestrategie voor IC met neurale netwerken

wordt bijvoorbeeld Adam gebruikt [37]. NIC-NES kan deze learning rate laten afnemen tijdens het trainen volgens een schedule of met patience. Het is ook mogelijk een L2-regularisatieterm toe te voegen aan de gradiënt.

Hoewel NIC-NES gebruik maakt van gradient ascent, zijn de gradiënten stochastisch geschat en niet met backpropagation berekend. Er is dus geen backward pass door het netwerk nodig. Terwijl NIC-ES een alternatief biedt voor gradient descent (ascent) in het geheel, kan NIC-NES de subcomponent van gradient descent vervangen die de gradiënten berekent (namelijk backpropagation).

Algoritme 3.3: NIC-NES: natuurlijke evolutiestrategie voor IC

```

1: Invoer: subprocedures mutatie, evaluer, init_individu, rang_transformatie,
2:           gradient_descent_stap
3:           aantal mutaties per iteratie  $\lambda$ , learning rate  $\eta$ 
4:           training dataset van afbeeldingen & ground-truth-zinnen  $\mathcal{D}$ 
5:           validatie dataset  $\mathcal{D}_{val}$ 
6:
7:  $\theta_0 \leftarrow \text{init\_individu}()$ 
8: for  $j = 1 \dots J$  iteraties do
9:    $Batch \leftarrow$  willekeurige minibatch van grootte  $B$  uit  $\mathcal{D}$ 
10:
11:  # Mutaties & evaluatie
12:  for  $i = 1 \dots \lambda$  mutaties do
13:     $\delta^{(i)} \leftarrow \text{mutatie}(\theta_{j-1})$ 
14:     $f_i^- \leftarrow \text{evaluer}(\theta_{j-1} - \delta^{(i)}, Batch)$ 
15:     $f_i^+ \leftarrow \text{evaluer}(\theta_{j-1} + \delta^{(i)}, Batch)$ 
16:  end for
17:
18:  # Schatter voor gradiënt op basis van deltas en F
19:   $r^+, r^- \leftarrow \text{rang\_transformatie}(f^+, f^-)$ 
20:   $w \leftarrow r^+ - r^-$ 
21:   $\hat{g} \leftarrow \frac{1}{\lambda} \sum_{i=1}^{\lambda} w_i \delta^{(i)}$ 
22:   $\theta_j \leftarrow \theta_{j-1} + \text{gradient\_ascent\_stap}(\hat{g}, \eta)$ 
23:
24:  # Evaluatie op validatieset
25:   $Scores[j] \leftarrow \text{evaluer}(\theta_j, \mathcal{D}_{val})$ 
26: end for
27: return  $\theta_k, k = \arg \max_l [Scores[l]]$ 
```

Evaluatieprocedure. de evaluatieprocedure in NIC-NES (“evaluer” in algoritme 3.3) is gelijk aan die in 3.1.5. Maar gezien er nu geen selectie van individuen plaatsvindt op basis van hun relatieve fitness, hoeft de minibatch waarop de $\delta^{(i)}$

3. EVOLUTIESTRATEGIEËN VOOR HET ONDERTITELEN VAN AFBEELDINGEN MET NEURALE NETWERKEN

beoordeeld worden niet gelijk te zijn voor alle evaluaties in een iteratie. NIC-NES ondersteunt zowel gelijke als verschillende minibatches, een instelling geeft dit aan.

Paralleliseerbaarheid en complexiteit. Ook NIC-NES is zeer paralleliseerbaar: analoog aan de geparalleliseerde versie van NIC-ES (algoritme 3.2) kan algoritme 3.3 naar het master-worker-schema omgevormd worden. De workers genereren de $\delta^{(i)}$ en evalueren deze door $F(\theta \pm \delta^{(i)})$ te berekenen. Die evaluatie is het meest rekenintensieve werk, zolang λ niet te groot is.

De master combineert alle $\delta^{(i)}$ en berekent op basis van hun respectievelijke fitness in elke iteratie een schatter voor de gradiënt. Deze combinatie omvat naast het sorteren van de fitness-scores zoals in NIC-ES (maar nu voor de rangtransformatie), ook het berekenen van een gewogen gemiddelde van λ vectoren van $\text{dim}(\theta)$. Bijgevolg is het werk van de master bij NIC-NES rekenintensiever dan bij NIC-ES. Zoals Zhang et al. opmerken stijgt het rekenwerk van de master lineair in λ (en in $\text{dim}(\theta)$) en kan de master daarom een bottleneck worden voor grote λ [104]. In tegenstelling tot in NIC-ES is in NIC-NES dus niet alleen de rekentijd voor een evaluatie maar ook de rekentijd van het masterproces een stijgende functie van $\text{dim}(\theta)$. In de experimenten in hoofdstuk 4 kan λ daarom voor NIC-NES niet te groot genomen worden. Een afweging moet hier gemaakt worden, gezien een grotere λ wel zorgt voor een nauwkeurigere schatter van de gradiënt [104].

Voor de rest geeft een complexiteitsanalyse van NIC-NES grotendeels dezelfde bevindingen als voor NIC-ES.

Hoofdstuk 4

Experimenten & resultaten

Dit hoofdstuk beschrijft de experimenten van dit onderzoek waarin NIC-ES en NIC-NES uit hoofdstuk 3 gebruikt worden om afbeeldingen te ondertitelen.

In het eerste onderdeel worden de dataset, de voorbereiding en andere praktische overwegingen toegelicht. In de tweede sectie worden de experimenten beschreven. De waarden voor de hyperparameters worden vermeld. Het verloop van de training en de resultaten van NIC-ES en NIC-NES worden vergeleken met cross-entropy en policy gradients. Dan volgen experimenten die specifieke aspecten van NIC-ES en NIC-NES vergelijken: de verschillende selectiemechanismen, mutatievarianten en evaluatieprocedures uit het vorige hoofdstuk worden onder andere vergeleken. Vervolgens worden in onderdeel 4.3 de resultaten van de experimenten uit de eerste secties verder geïnterpreteerd. Ten slotte volgen nog suggesties voor verder onderzoek.

4.1 Voorbereiding en praktische overwegingen

Data. Als dataset wordt de MSCOCO Captions dataset gebruikt [12]. Deze dataset bevat 123K afbeeldingen, elk met 5 ondertitels. Daarvan worden 5K afbeeldingen, met bijhorende ondertitels, voor de validatieset (\mathcal{D}_{val}) genomen en 5K voor de testset (\mathcal{D}_{test}), zoals in [35, 70]. De rest van de dataset wordt gebruikt als trainingdata (\mathcal{D}). Karpathy's train/val/test-splits worden gebruikt: d.w.z. dat de validatie- en testset uit dezelfde afbeeldingen bestaan als in [35].

De voorverwerking van de data is overgenomen uit [70], buiten dat in de experimenten in deze thesis alle woorden die ≤ 5 keer voorkomen in \mathcal{D} vervangen worden door een "unknown"-woord (in plaats van ≤ 4). De resulterende woordenschat telt 9488 woorden. Het begin en het einde van een zin worden gemarkeerd door speciale "start"- en "einde"-woorden. De zinnen worden afgebroken na 16 woorden.

Zoals reeds vermeld is het gebruikte CNN een ResNet-101 dat voorgetraind is voor objectdetectie op de Visual Genome dataset. De Visual Genome dataset bevat afbeeldingen die ook in de MSCOCO Captions dataset voorkomen. Om contaminatie te voorkomen is er voor gezorgd dat deze afbeeldingen bij het voortrainen van het CNN en bij het trainen met NIC-(N)ES voor IC in hetzelfde deel van de dataset voorkomen (\mathcal{D} , \mathcal{D}_{val} of \mathcal{D}_{test}) [3].

Model. Sectie 3.1.1 beschrijft de gebruikte modellen en de parameters die gezocht worden. De parameters van het CNN worden niet meer aangepast. De uitvoer van het CNN heeft na *average pooling* 2048 dimensies. De word embeddings en verborgen toestandsvectoren van de LSTM hebben 128 dimensies. In [70] hebben deze 512 dimensies in de plaats van 128, maar in deze thesis werd 128 gekozen om het totaal aantal parameters te beperken. Het resulterende netwerk telt 2.9M parameters.

Om het trainen efficiënter te laten verlopen zijn op voorhand de feature-vectoren met 2048 dimensies al door het CNN berekend (1 per afbeelding) en lokaal opgeslagen. Dat is mogelijk omdat de parameters van het CNN toch niet meer veranderen bij het trainen, de feature-vector die het CNN berekent voor dezelfde afbeelding zal daarom gedurende heel de training gelijk zijn. Tijdens het trainen zijn dan geen forward passes meer door het CNN nodig, wat tijd bespaart. In plaats van afbeeldingen worden de respectievelijke 2048-dimensionale feature-vectoren ingeladen.

Implementatie en hardware. De algoritmes werden geïmplementeerd in Python, gebruik makende van het deep-learning-framework PyTorch [63]. De communicatie tussen de workers en de master verloopt voornamelijk via Redis: een gedistribueerd *key-value* databasesysteem. Dat zou het gemakkelijk maken om de implementatie aan te passen om op gedistribueerde hardware uitgevoerd te worden. Om het totale geheugengebruik te beperken worden populaties en nakomelingen van parameters θ in NIC-ES op de lokale schijf opgeslagen en zo tussen de master en workers gecommuniceerd. Omdat dat echter geen significante verbetering bleek te zijn worden in NIC-NES de mutatievectoren wel via het werkgeheugen gecommuniceerd. De code en een handleiding zijn te vinden in een Github repository¹.

Voor de experimenten met XENT en PG werd een Nvidia Tesla K80 GPU gebruikt op een Google Cloud Compute server. NIC-ES en NIC-NES werden ook via Google Cloud uitgevoerd, op servers met 24 tot 96 virtuele CPU's. Voor meer informatie over de vCPU's verwijst de tekst de lezer naar de Google Cloud documentatie². Buiten op Google Cloud werd NIC-(N)ES ook op infrastructuur van het Vlaams Supercomputer Centrum uitgevoerd. Enkele vCPU's werden telkens voor het masterproces behouden, de rest van de beschikbare vCPU's voerden elk één workerproces uit. In totaal werden tot 92 parallelle workerprocessen gebruikt.

4.2 Experimenten

In de volgende subsecties worden experimenten met NIC-ES en NIC-NES besproken. Eerst worden nog de geteste en gekozen hyperparameters vermeld. In subsectie 4.2.1 worden NIC-ES, NIC-NES, teacher forcing en een policy-gradient-methode vergeleken. In subsectie 4.2.2 worden experimenten besproken waarbij NIC-(N)ES start van een willekeurige populatie of van een willekeurig individu. Vervolgens beschrijven subsecties 4.2.3 tot 4.2.6 respectievelijk experimenten die de mutatievarianten, de evaluatiemethoden, de selectiemechanismen bij NIC-ES en gelijke vs. verschillende

¹<https://github.com/rubencart/es-img-captioning>

²<https://cloud.google.com/compute/docs/cpu-platforms>

minibatches bij NIC-NES vergelijken. De resultaten worden geïnterpreteerd en wanneer mogelijk wordt een verklaring gegeven voor de observaties.

Hyperparameters. De hyperparameters voor NIC-ES en NIC-NES werden in eerste instantie overgenomen van de experimenten van Such et al. (sectie 2.3.3), Salimans et al. (2.3.3), Lehman et al. (2.3.3) en Zhang et al. [46, 75, 83, 104], omdat die experimenten gelijkenissen vertonen met de experimenten in deze thesis. Vertrekende van overgenomen waarden voor hyperparameters werd geëxperimenteerd met waarden in de buurt van die initiële waarden. Uit een aantal kandidaat-waarden werden dan de waarden gekozen waarmee na een vast aantal training-iteraties de hoogste CIDEr-score op de validatieset gehaald werd. Daarbij was het niet mogelijk om alle combinaties van alle waarden te proberen, bij het vergelijken van waarden voor één hyperparameter werd dus de rest constant genomen. Een onderdeel van dit proces is het gevoelsmatig en manueel voorspellen welke waarden goed zouden kunnen werken. Bovendien leidt deze aanpak vooral tot parameters die voor snelle stijging zorgen en niet per se voor een hoge uiteindelijke CIDEr-score. Indien er meer resources vorhanden waren geweest, had een meer gestructureerde en systematische methode gebruikt kunnen worden.

Tabel 4.1 toont de geteste en de gekozen waarden. L2-regularisatie in NIC-NES werd pas in de laatste experimenten getest, en dus enkel in de experimenten in sectie 4.2.1 gebruikt. In de rest van de experimenten met NIC-NES werd geen regularisatie gebruikt. Tenzij in de volgende secties anders vermeld, zullen voor de rest de hyperparameters uit tabel 4.1b gebruikt zijn. De twee manieren om σ , B , η of een subset van de drie tijdens het trainen dynamisch aan te passen werden geprobeerd. Bij NIC-NES bleken de curriculumtactieken geen meerwaarde. Bij NIC-ES werd elke $H = 200$ iteraties $\sigma \leftarrow \sigma/\sqrt{2}$, $B \leftarrow B \cdot \sqrt{2}$ genomen.

4.2.1 NIC-ES, NIC-NES, teacher forcing en policy gradients

In deze sectie worden experimenten beschreven die training met NIC-ES en NIC-NES vergelijken met baselines: teacher forcing (XENT) zoals bij Vinyals et al. en de self-critical loss (SCST) van Rennie et al. [94, 70]. Bij teacher forcing (sectie 2.2.1) wordt de gradiënt van de cross-entropy tussen het ground-truth-woord en het voorspelde woord berekend. Tijdens het trainen dient telkens het vorige ground-truth-woord als invoer. SCST is een policy-gradient-methode (sectie 2.2.3), waarbij uit CIDEr-scores voor gegenererde zinnen gradiënten berekend worden. Als invoer dient hierbij telkens het woord dat het model voorspelde in de vorige tijdstap. Beide baselines gebruiken SGD en backpropagation. Figuur 4.1 toont een overzicht en tabel 4.2 toont de hoogste CIDEr-scores.

XENT en policy gradients. Het netwerk uit sectie 4.1 werd voor 30 epochs met teacher forcing en Adam met een afnemende learning rate getraind, volgens de procedure van Rennie et al. [70]. Er werd gebruikt gemaakt van een GPU, en de training duurde ongeveer 5u. Er werd geen dropout of scheduled sampling gebruikt. Vervolgens werd het resulterende netwerk (θ_{xent}) voor 27u en 25 epochs

4. EXPERIMENTEN & RESULTATEN

Hyperparameter	NIC-ES	NIC-NES
λ	[1000, 2000]	
σ	[0.001, 0.05]	
B	[32, 1024]	
Mutatie	{M-STD, VM-P, VM-G}	
$F(\boldsymbol{\theta})$	{ $F_{gr}(\boldsymbol{\theta})$, $F_{sam}(\boldsymbol{\theta})$, $F_{scst}(\boldsymbol{\theta})$ }	
Curriculum	{patience, schedule}	
μ	[20, 200]	
Selectie ouders	{UWS, TS}	
S	{5, 10}	
E	{1, 2, 3}	
C	{1, 2, 3}	
η		[0.001, 0.005]
= of \neq batches		{=, \neq }
L2-reg.		[1e-8, 0]

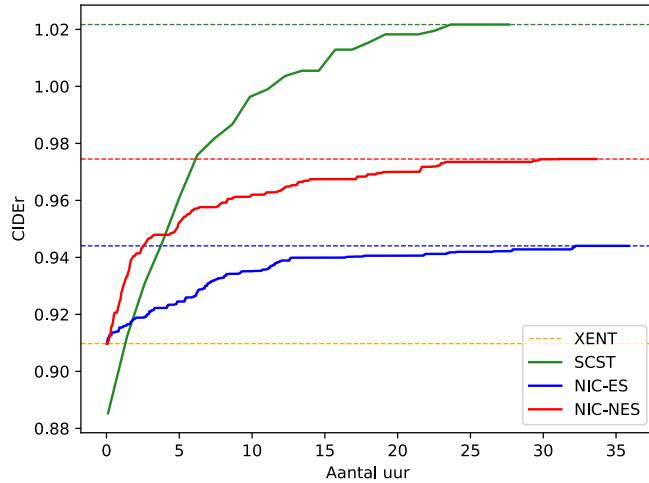
(a) Uitgeprobeerde waarden

Hyperparameter	NIC-ES	NIC-NES
λ	1000	
σ	0.005 (afn.)	0.01
B	256 (toen.)	128
Mutatie	VM-P	M-STD
$F(\boldsymbol{\theta})$	$F_{gr}(\boldsymbol{\theta})$	
Curriculum	schedule	/
μ	50	
Selectie ouders	UWS	
S	/	
E	3	
C	2	
η		0.001
= of \neq batches		\neq
L2-reg.		{2e-7, 0}

(b) Gekozen waarden

Tabel 4.1: Waarden voor hyperparameters van NIC-ES en NIC-NES

Waarden voor de hyperparameters van NIC-(N)ES: aantal nakomelingen λ , mutatiesterkte σ , batchgrootte B , mutatievariant, fitness, curriculumtactiek, populatiegrootte μ , selectie van ouders, toernooigrootte S , aantal elites E , aantal elitekandidaten C , de learning rate η , = of \neq batches en de coëfficiënt voor L2-regularisatie. Toen. en afn. willen respectievelijk zeggen toenemend en afnemend volgens een schedule of met patience. Tabel 4.1a toont de geteste waarden, en tabel 4.1b toont de gekozen waarden voor de beste resultaten.



Figuur 4.1: SCST, NIC-ES en NIC-NES: CIDEr in functie van de tijd
De hoogste CIDEr-score op de validatieset tijdens 1 run met SCST [70] en 2 runs met NIC-ES en met NIC-NES.

	XENT [94]	SCST [70]	NIC-ES	NIC-NES
\mathcal{D}_{val}	0.910	1.022	$0.944 \pm 7e-3$	$0.975 \pm 7e-4$
\mathcal{D}_{test}	0.914	1.035	0.935	0.989

Tabel 4.2: Scores van XENT, SCST, NIC-ES en NIC-NES op de validatie- en testset

verder getraind met de SCST-lossfunctie van Rennie et al., ook op een GPU. Tijdens de eerste iteraties van training met SCST daalde de score, waardoor het in figuur 4.1 lijkt alsof deze training van een ander punt dan θ_{xent} vertrokken is.

NIC-ES en NIC-NES. Vertrekende van θ_{xent} werden NIC-ES en NIC-NES elk ook tweemaal uitgevoerd op 96 vCPU's. De algoritmes werden gestopt wanneer de validatiescore voor een aantal iteraties niet meer verbeterd werd. Hoewel de experimenten in sectie 4.2.3 aantoonden dat veilige mutaties door uitvoergradiënten bij NIC-NES goed werken, werd NIC-NES in deze sectie met de standaardmutaties uitgevoerd. De reden hiervoor is dat experimenten met L2-regularisatie een hoge score gaven, en dat er niet genoeg tijd was om NIC-NES uit te voeren met zowel de veilige mutaties als met L2-regularisatie.

De algoritmes uit de vorige en deze paragraaf vergelijken in de tijd is moeilijk omdat ze op verschillende hardware uitgevoerd zijn, en er moet dus opgepast worden bij het trekken van conclusies uit deze vergelijking. De vergelijking kan echter nog steeds wel nuttig zijn om een idee te geven van de nodige tijd voor convergentie. Figuur C.1 in appendix C toont voor dezelfde runs de CIDEr-score in functie van het

4. EXPERIMENTEN & RESULTATEN

aantal datapunten in plaats van de tijd. Appendix C bevat eveneens grafieken van andere variabelen tijdens trainingruns met NIC-(N)ES, zoals de verandering van de norm van de parameters en de trainingscores. In appendix B zijn enkele foto's uit de testset te zien samen met de ondertitels die de netwerken getraind met NIC-(N)ES, teacher forcing en SCST uit deze experimenten ervoor genereren.

Discussie. Een eerste vaststelling is dat NIC-ES, NIC-NES en SCST de score nog kunnen verbeteren ten opzichte van teacher forcing. Dat is in overeenstemming met de verwachtingen. Door op sequentieniveau te trainen, zonder exposure bias, rechtstreeks voor de te optimaliseren evaluatiefunctie, worden de besproken discrepanties uit de literatuurstudie van de klassieke trainingmethode overbrugd. Het feit dat niet alleen policy-gradient-methoden, maar ook heel verschillende algoritmes zoals NIC-(N)ES, een verbetering geven tegenover teacher forcing onderschrijft extra het belang van de twee discrepanties. SCST zorgt echter voor de grootste verbetering: 13.2%, tegenover resp. 2.3% en 8.2% door NIC-ES en NIC-NES op de testset. Daaruit kan geconcludeerd worden dat het volgen van analytische gradiënten met SCST meer loont dan het gebruik van (natuurlijke) evolutiestrategieën, althans voor deze varianten van NIC-(N)ES met deze hyperparameters.

Vervolgens is de tijd nodig voor convergentie bij SCST en NIC-(N)ES van dezelfde grootte-orde. Een eenduidige vergelijking is moeilijk door de verschillende hardware waarop ze uitgevoerd werden. SCST zou, bij gebruik van meerdere GPU's, waarschijnlijk minder tijd nodig hebben dan in dit experiment. Ook NIC-(N)ES zou, bij gebruik van meer dan 96 vCPU's, sneller kunnen uitvoeren. Het zou interessant zijn om bv. het totaal aantal arithmetische operaties bij de vier algoritmes te vergelijken.

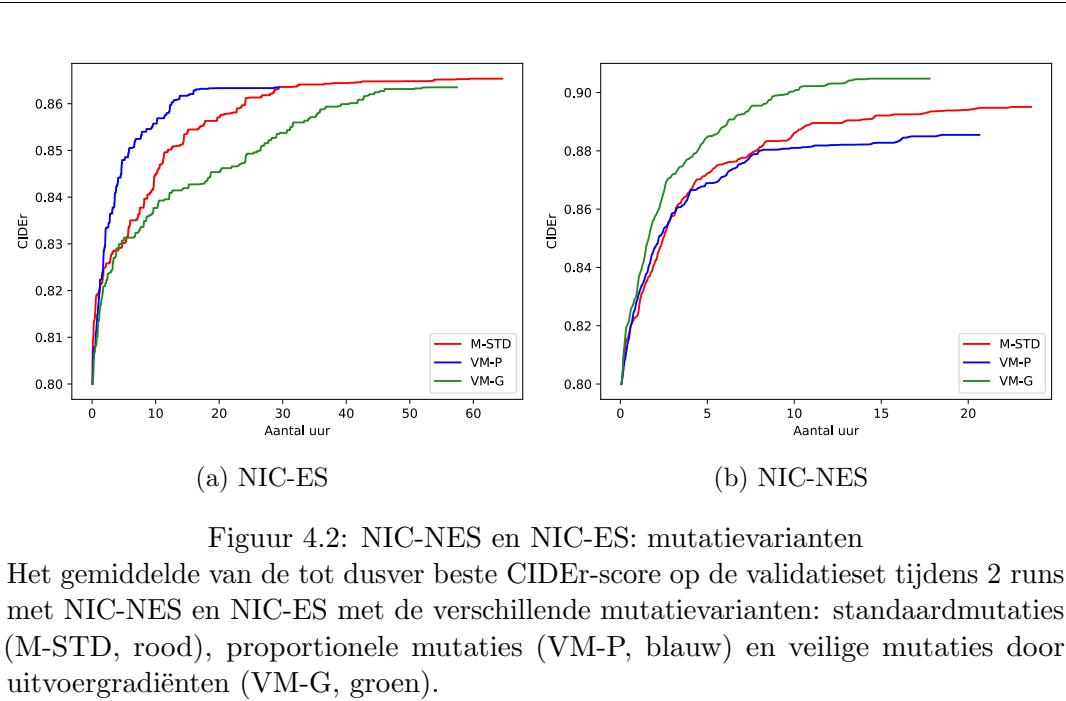
Bijkomende experimenten. Experimenten met verschillende aantallen CPU's bevestigden de hypothese dat NIC-ES en NIC-NES gebaat zijn bij executie op parallelle CPU's. Bij een toenemend aantal CPU's en workerprocessen nam de nodige tijd voor het werk van de workerprocessen ongeveer lineair af in het aantal CPU's.

Er werden ook experimenten gedaan waarbij NIC-(N)ES vertrok van de parameters verkregen uit training met SCST i.p.v. met teacher forcing. Daarbij steeg de CIDEr-score echter niet verder.

Om in NIC-ES meer diversiteit in de populatie te krijgen werd geprobeerd om de populatie te initialiseren met 5 verschillende $\theta_{xent}^{(i)}$, die elk volgden uit een andere trainingrun met teacher forcing, en met ongeveer gelijke fitness. Geen significante verbetering werd waargenomen.

Er werd ook geprobeerd om NIC-ES sequentieel op een GPU te laten lopen. De kost van alle evaluaties sequentieel te moeten uitvoeren was echter te groot, waardoor één iteratie veel langer duurde.

Zoals reeds vermeld in sectie 3.1.1 uit het vorige hoofdstuk werd ook de ge-comprimeerde representatie voor neurale netwerken getest. In sectie 2.3.3 wordt deze representatie toegelicht. De representatie vertraagde het algoritme aanzienlijk,



Figuur 4.2: NIC-NES en NIC-ES: mutatievarianten

Het gemiddelde van de tot dusver beste CIDEr-score op de validatieset tijdens 2 runs met NIC-NES en NIC-ES met de verschillende mutatievarianten: standaardmutaties (M-STD, rood), proportionele mutaties (VM-P, blauw) en veilige mutaties door uitvoergradiënten (VM-G, groen).

omdat de lengte van de lijsten van random seeds, en dus de nodig tijd voor de conversie van genotype naar fenotype, lineair groeit in het aantal iteraties. Gezien de experimenten in deze thesis op één (virtuele) machine uitgevoerd werden vormde de nodige bandbreedte voor communicatie tussen de master- en workerprocessen een minder beperkende factor en was de compressie de extra tijdskost niet waard.

4.2.2 NIC-ES en NIC-NES: willekeurige initialisatie

Indien NIC-ES vertrekt van een willekeurig geïnitialiseerde populatie, met de initialisatieprocedure zoals besproken in hoofdstuk 3, stagneert de stijging van de CIDEr-score consequent rond een score in $[0, 2]$, ook voor relatief grote waarden voor σ . Dat suggereert dat het algoritme in een lokaal optimum of in een vlak deel van de zoekruimte terecht komt. Bij NIC-NES vindt een gelijkaardige stagnatie plaats. Deze bevindingen komen overeen met wat Ranzato et al. ondervinden in [65] en zijn wellicht toe te schrijven aan de grootte van de zoekruimte van sequenties tegenover die van individuele woorden.

4.2.3 NIC-ES en NIC-NES: mutatievarianten

De verschillende voorgestelde mutatievarianten uit sectie 3.1.3 werden vergeleken bij zowel NIC-ES als NIC-NES: standaardmutaties (M-STD), proportionele mutaties (VM-P) en veilige mutaties door uitvoergradiënten (VM-G). Figuur 4.2 en tabel 4.3 tonen de resultaten.

4. EXPERIMENTEN & RESULTATEN

Mutatie	NIC-ES	NIC-NES
M-STD	0.865 ± 3e-3	0.895 ± 2e-3
VM-P	0.863 ± 2e-3	0.886 ± 2e-3
VM-G	0.864 ± 4e-3	0.905 ± 8e-3

Tabel 4.3: Validatiescores van NIC-ES en NIC-NES met mutatievarianten

Experiment. In deze en de experimenten in de volgende secties werd NIC-(N)ES gestart van een andere θ'_{xent} , die een CIDEr-score haalt van 0.80 op de validatieset (in plaats van 0.91). Die θ'_{xent} werd bekomen uit voortraining met teacher forcing en Adam zoals bij Rennie et al. voor 14 epochs, zonder afnemende learning rate. De keuze werd gemaakt omdat het trainen met NIC-(N)ES vertrekende van een minder gefinetuned netwerk gemakkelijker bleek. Zo had NIC-(N)ES meer “ruimte” tijdens het trainen en waren verschillen in performantie bij verschillende instellingen groter, hetgeen een betere vergelijking faciliteerde. In appendix C zijn figuren te zien waarop teacher forcing, SCST, NIC-ES en NIC-NES vergeleken worden, bij starten van θ'_{xent} .

De mutatiesterkte σ krijgt bij de verschillende mutaties een andere betekenis: de waarde voor σ moest dus opnieuw gekozen worden. Dat werd gedaan volgens de eerder beschreven procedure. Vervolgens werden 2 runs van beide algoritmes gedaan met elke mutatievariant, telkens op 96 vCPU’s, met buiten σ alle andere instellingen gelijk. De runs werden afgebroken wanneer een vast aantal iteraties de beste validatiescore niet meer verbroken werd. Figuur 4.2 toont voor beide algoritmes de stijging van de validatiescore in functie van de tijd, gemiddeld over de twee runs. Tabel 4.3 laat de gemiddelde hoogste CIDEr-scores en hun standaardafwijkingen zien.

Resultaten. De bevindingen zijn zeer verschillend voor beide algoritmes. Bij NIC-NES zorgt VM-G vergeleken met de standaardmutatie voor een significante toename van zowel de hoogste score als de snelheid van de stijging. De proportionele mutatie heeft het omgekeerde effect. Bij NIC-ES daarentegen is de hoogste CIDEr-score met de drie mutatievarianten ongeveer gelijk; ze liggen binnen één standaardafwijking van elkaar. De snelheid van stijging verschilt echter drastisch: de tijd die nodig is om een score van 0.86 te bereiken is bij de proportionele mutatie, de standaardmutatie en VM-G respectievelijk gemiddeld 12u, 24u en 40u. Ook opmerkelijk is dat in tegenstelling tot bij NIC-NES, bij NIC-ES de proportionele mutaties tot een verbetering ten opzichte van de standaardmutatie leiden en de mutaties met uitvoergradiënten net tot een verslechtering. Het aantal runs per mutatievariant is niet groot genoeg om statistische significantie te testen.

Zoals reeds in sectie 3.1.3 vermeld is het ook mogelijk om een vaste vector van $\dim(\theta)$ met een sensitiviteit per parameter als invoer aan NIC-(N)ES te geven. Er werden experimenten gedaan met in die vector de gemiddelde sensitiviteiten van een referentie-individu over heel de dataset. Met deze instelling steeg de CIDEr-score

trager en stagneerde de score rond een lagere score, vergeleken met alle andere mutaties.

Discussie. De verschillende effecten kunnen deels toe te schrijven zijn aan onvoldoende tuning van σ . Het feit dat bij NIC-ES de mutaties met uitvoergradiënten een hogere tijdskost hebben kan ook verklaard worden door de extra rekenkost van de berekening van de gradiënten. Deze berekening moet in NIC-ES namelijk voor elke ouder gebeuren, en in NIC-NES slechts voor één individu per iteratie.

Zoals reeds in sectie 3.1.3 vermeld, steunen de proportionele mutaties op de assumptie dat kleine parameters ook gevoelige parameters zijn. De resultaten tonen aan dat die assumptie niet klopt, of toch niet voor dezelfde notie van gevoeligheid bij VM-G en VM-P.

Vervolgens kan het gebruik van het gradient-descent-algoritme Adam bij NIC-NES het verschil in het effect verder verklaren. Adam updateert elke parameter met een aparte en adaptieve learning rate: parameters met een kleinere gradiënt worden als minder sensatief gezien en krijgen een relatief grotere update, en vice versa [37]. Ook parameters waar de gradiënt consistent is, d.w.z. in opeenvolgende iteraties ongeveer gelijk, krijgen grotere updates.

Bij proportionele mutaties zullen relatief kleine parameters in de meeste iteraties een relatief kleine mutatie krijgen, en dus consistent een klein overeenkomstig element in de geschatte gradiënt hebben. Daardoor zal Adam grotere stappen zetten in deze parameters, wat juist niet het gewenste effect is. VM-G berekent een sensitiviteit op basis van de huidige minibatch. De respectievelijke sensitiviteitswaarde van een parameter kan door verschillen in minibatches zeer verschillend zijn in opeenvolgende iteraties. Daarom zal de schatter voor de gradiënt minder consistent zijn, wat Adam wel ontmoedigt om grote stappen te zetten.

Het zoeken van een verdere, empirisch onderbouwde verklaring, ook voor het verschil bij NIC-ES, wordt gelaten voor verder onderzoek.

Hoewel de mutaties door uitvoergradiënten bij NIC-NES goed blijken te werken en ze zorgen voor een netto tijdsvermindering, moeten de nadelen ook in rekening gebracht worden. Die nadelen zijn de extra rekenkost en het feit dat er toch analytische gradiënten berekend moeten worden met backpropagation. Zo is VM-G bijvoorbeeld niet bruikbaar voor modellen die een niet-differentieerbare functie f_θ implementeren.

4.2.4 NIC-ES en NIC-NES: fitnessfuncties

Verschillende experimenten werden gedaan om de fitnessfuncties uit sectie 3.1.5 te vergelijken. De bevindingen zijn gelijk voor NIC-ES en NIC-NES.

Greedy decoding. De eerste geteste fitnessfunctie is de CIDEr-score onder greedy decoding: $F_{gr}(\theta)$. Dat gaf voor beide algoritmes de beste resultaten, die reeds in sectie 4.2.1 vermeld werden.

4. EXPERIMENTEN & RESULTATEN

Samplen. De volgende fitnessfunctie is de CIDEr-score onder samplen: $F_{sam}(\boldsymbol{\theta})$. In geen van de experimenten werd een significante stijging in CIDEr-score op de validatieset bekomen met deze fitnessfunctie, en meestal daalde de score zelfs met de iteraties. Bij zeer grote minibatches (bv. 512, 1024) en kleine σ daalde de score niet maar de grote minibatches maakten het algoritme traag en de kleine σ belemmerde vooruitgang. De CIDEr-score vertoonde grote schommelingen zonder significante stijging.

De CIDEr-score onder samplen blijkt dus geen nuttig signaal te zijn voor de werkelijke fitness van een netwerk. Een verklaring kan zijn dat de kans op u_t^* , het woord waaraan het netwerk de grootste kans toekent, samplen van p_{model} veel kleiner is dan 1. Een inspectie bevestigt dit: $p_{model}(u_t^*)$ is onder greedy decoding bij het model dat met teacher forcing getraind is, voor 1024 willekeurige afbeeldingen uit \mathcal{D} gemiddeld (over de minibatch en t) gelijk aan 0.33. Het feit dat de uitvoer een softmax is over een grote vector zou hieraan kunnen bijdragen. Deze fitnessfunctie geeft dus niet per se een incorrect maar wel een te *noisy* beeld van de werkelijke fitness.

Verder berekent elke nakomeling voor elke afbeelding in de huidige minibatch per tijdstap slechts éénmaal $p_{model}(u_t)$. Het is niet evident dat eenmalig samplen van de verdeling een representatieve score geeft: dat is zoals een kansverdeling benaderen met één enkele Monte Carlo-sample.

SCST. Ook met de SCST-fitness $F_{scst}(\boldsymbol{\theta}) = \text{CIDEr}(\hat{\mathbf{y}}) - \text{CIDEr}(\mathbf{y}^*)$ werden geen interessante resultaten gehaald. Een mogelijke verklaring is ongetwijfeld dat er meer tuning van de hyperparameters nodig was. Elke fitnessfunctie bepaalt een ander optimalisatielandschap en er waren niet genoeg tijd en resources beschikbaar om voor elke fitness alle hyperparameters even extensief te tunen.

Het volgende is een meer analytische verklaring. Bij SCST met gradient descent en backpropagation wordt de lossfunctie analytisch gedifferentieerd t.o.v. elke parameter in $\boldsymbol{\theta}$. Dat wil zeggen dat de gradiënt, en dus ook de verandering bij een update, voor eenzelfde waarde van de lossfunctie verschillend kan zijn in verschillende parameters. De gradiënt van $J(\boldsymbol{\theta}) = -F(\boldsymbol{\theta})$ t.o.v. de input van de softmax functie wordt gegeven door [70]:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \mathbf{s}_t} \approx (r(\hat{\mathbf{y}}) - r(\mathbf{y}^*)) (p_{model}(\hat{u}_t | \mathbf{h}_t; \boldsymbol{\theta})) - \mathbf{1}_{\hat{u}_t}, \quad (4.1)$$

Met $\mathbf{1}_{\hat{u}_t}$ de one-hot encoding van \hat{u}_t en $r(\hat{\mathbf{y}}) = \text{CIDEr}(\hat{\mathbf{y}}, \mathbf{y})$. Indien $r(\hat{\mathbf{y}}) > r(\mathbf{y}^*)$ dan zal gradient descent de parameters die een positieve bijdrage hebben in $p_{model}(\hat{u}_t)$ zo veranderen dat de kans op \hat{u}_t samplen groter wordt. De parameters die een positieve bijdrage hebben in $p_{model}(v)$ van alle andere woorden $v \neq \hat{u}_t$ zullen zo veranderen zodat $p_{model}(v)$ net afgezwakt wordt. Dat betekent dat na een update van gradient descent de kans groter zal zijn dan ervoor om woorden te samplen met een hogere CIDEr-score dan de woorden die het model (voor de update) onder greedy decoding voorspeld zou hebben.

In NIC-(N)ES daarentegen worden parameters en mutaties als geheel geëvalueerd en worden verschillende delen van de netwerken niet op verschillende manieren veranderd. De waarde van $F_{scst}(\theta)$ is wat belangrijk is, en niet de analytische gradiënt t.o.v. de parameters. De waarde van F_{scst} is echter geen goed signaal voor de werkelijke fitness van een netwerk. Dat blijkt onder andere uit het feit dat F_{scst} groot is voor netwerken waarbij greedy decoding niet de hoogste score oplevert. Gezien tijdens inferentie de score onder greedy decoding net zo hoog mogelijk moet zijn, is dat geen gewenste eigenschap.

4.2.5 NIC-ES: selectiemechanismen

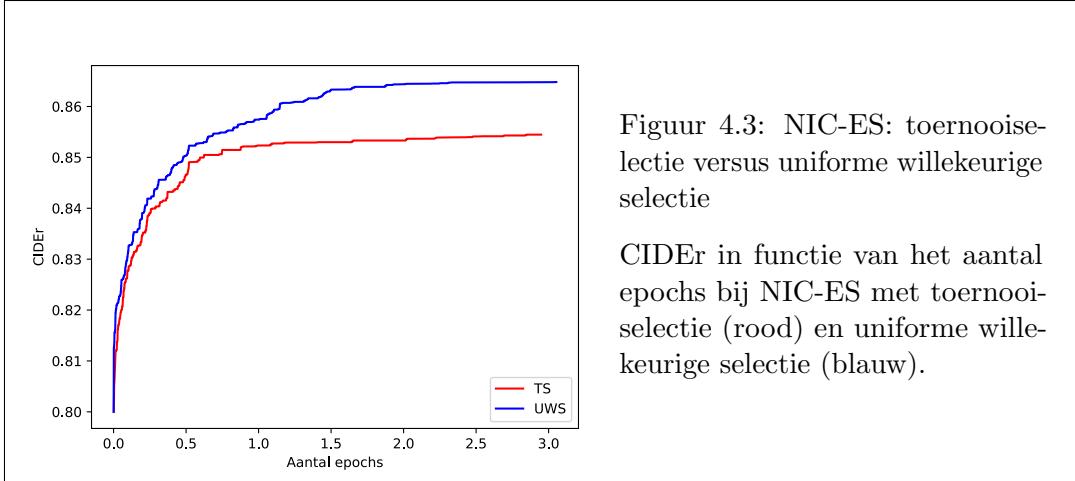
De evolutiestrategie van Such et al. [83] houdt voor Atari-spellen een populatie van 20 (+1 elite) individuen bij en genereert elke generatie 1000 nakomelingen door mutatie uit ouders geselecteerd door uniforme willekeurige selectie. Deze μ/λ -verhouding wijkt sterk af van de door Eiben en Smith aangeraden 1/7 tot 1/4 [19]: de selectiedruk ligt dus veel hoger en dat werkt diversiteit tegen.

Om deze reden werden experimenten gedaan met NIC-ES waarbij de selectiedruk lager gelegd werd. De standaardmutaties werden gebruikt. Het aantal nakomelingen werd minstens gelijk genomen: $\lambda = \{1000, 2000\}$, omdat $\dim(\theta)$ groot is en daarom een grote λ nodig is om een voldoende groot deel van de zoekruimte in de buurt van individuen uit de populatie te bedekken in elke generatie. In de experimenten werd μ gelijk genomen aan $\{20, 30, 40, 50, 100, 150, 200\}$ om gradueel dichter bij de aangeraden 1/7-verhouding te komen. Met UWS bleken $\mu = 50, \lambda = 1000$ het beste te werken, d.w.z. na een vast aantal geziene datapunten leidden deze waarden tot de hoogste CIDEr-score.

De stijging van de CIDEr voor $\mu \geq 100$ verliep zeer traag en met grote variantie, wat suggereert dat de selectiedruk te laag lag. Om de selectiedruk terug te verhogen en toch meer diversiteit te behouden dan bij $\mu = 20$ zoals in [83] werden ook experimenten gedaan waarbij de ouders uit de populatie gekozen werden met toernooiselectie. Bij toernooiselectie hebben individuen met een hogere fitness een hogere kans om als ouder gekozen te worden, wat de selectiedruk verhoogt, maar individuen met een lagere fitness hebben toch ook een kleine kans, wat de weg openlaat voor exploratie. De beste gevonden parameters zijn toernooigrootte $S = 5$ en $\mu = 100$.

NIC-ES werd met beide combinaties (UWS, $\mu = 50$) en (TS, $S = 5, \mu = 100$) en voor de rest gelijke instellingen driemaal uitgevoerd, tot de score niet meer significant veranderde voor een vast aantal iteraties. Figuur 4.3 toont het gemiddelde over de drie runs van de tot dan toe beste CIDEr-score op de validatieset voor beide selectiemechanismen. NIC-ES bereikte met UWS gemiddeld een CIDEr-score van $0.865 \pm 2e-3$, met TS een score van $0.854 \pm 1.5e-3$. Hoewel de steekproef zeer klein is, kan met de Mann-Whitney U-test met $p < 0.05$ besloten worden dat UWS met de vermelde hyperparameters beter werkt dan TS met de vermelde hyperparameters [51]. De gebruikte μ, σ, S en andere hyperparameters beïnvloeden dit resultaat natuurlijk en de conclusie geldt bijgevolg zeker niet als algemene vergelijking van UWS en TS.

4. EXPERIMENTEN & RESULTATEN



Figuur 4.3: NIC-ES: toernooiselectie versus uniforme willekeurige selectie

CIDEr in functie van het aantal epochs bij NIC-ES met toernooiselectie (rood) en uniforme willekeurige selectie (blauw).

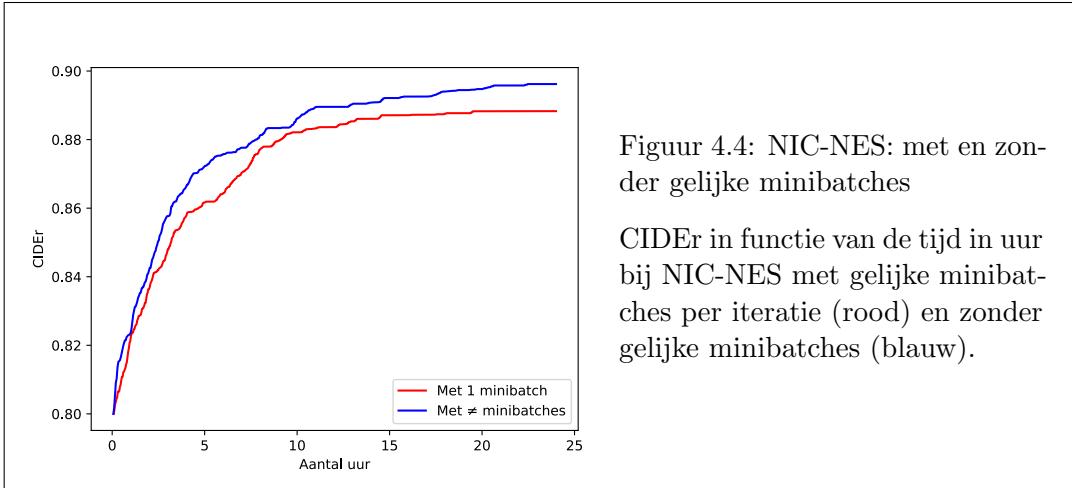
4.2.6 NIC-NES: met en zonder gelijke minibatches

Zoals al in sectie 3.2 aangegeven kan NIC-NES binnen een iteratie ofwel alle mutaties δ beoordelen op dezelfde minibatch, ofwel een nieuwe minibatch uit \mathcal{D} nemen per evaluatie van een mutatie.

De twee tactieken werden getest in twee keer 3 runs van NIC-NES elk van 24u met 92 workers op 96 vCPU's, met alle andere instellingen gelijk. NIC-NES gebruikte voor deze experimenten de standaard mutatievariant. Conform met wat Zhang et al. in [104] ondervonden stijgt de CIDEr bij NIC-NES zonder gelijke minibatches sneller in de tijd en is de uiteindelijke score hoger, hoewel het verschil niet groot is. Figuur 4.4 toont de resulterende CIDEr-curves in functie van de tijd. De scores uitdrukken in functie van het aantal geziene samples kan ook, maar is niet helemaal eerlijk omdat NIC-NES met verschillende minibatches voor dezelfde batchgrootte veel meer verschillende samples ziet dan NIC-NES met gelijke minibatches.

NIC-NES met gelijke minibatches haalde gemiddeld een CIDEr-score op de validatieset van $0.888 \pm 4e-3$. NIC-NES met verschillende minibatches haalde $0.896 \pm 1e-3$. Er kan, ondanks de zeer kleine steekproefgrootte, met een Mann-Whitney U-test met $p < 0.05$ besloten worden dat de verwachte gemiddelde score van NIC-NES met verschillende minibatches, met de huidige hyperparameters, hoger is dan die van NIC-NES met gelijke minibatches en de huidige hyperparameters.

Door de grootte van de dataset krijgt NIC-NES met gelijke minibatches niet de hele dataset gezien in minder dan 24u: slechts gemiddeld 62% van de dataset was een keer gebruikt voor de evaluaties na 24u. Het is duidelijk dat de kansverdeling p_{real} theoretisch gezien beter benaderd kan worden door heel \mathcal{D} te gebruiken. Elke evaluatie uitvoeren met een aparte willekeurige minibatch kan een manier zijn om toch op een kortere tijd de dataset integraal te benutten.



Figuur 4.4: NIC-NES: met en zonder gelijke minibatches

CIDEr in functie van de tijd in uur bij NIC-NES met gelijke minibatches per iteratie (rood) en zonder gelijke minibatches (blauw).

4.3 Verdere discussie

Algemeen. De experimenten, bv. die in sectie 4.2.1 tonen aan dat het mogelijk is om (natuurlijke) evolutiestrategieën te gebruiken om RNN's met $\mathcal{O}(10^6)$ parameters, na voortraining met teacher forcing, verder te trainen voor tekstgeneratie tot een hogere CIDEr-score. In de experimenten in sectie 4.2.2 was het echter niet mogelijk om heel de training met NIC-(N)ES te doen, vertrekkende van willekeurig geïnitialiseerde individuen. De voortraining met teacher forcing was essentieel, zoals ook Rennie et al. en Ranzato et al. ondervonden [70, 65]. Er kan dan ook niet geconcludeerd worden dat NIC-(N)ES de klassieke training met teacher forcing kan vervangen.

De experimenten in deze thesis zijn een verkenning van het gebruik van evolutiestrategieën voor tekstgeneratie met NNs. Hoewel NIC-(N)ES er niet in slaagt state-of-the-art resultaten van policy-gradient-methoden (SCST) te evenaren, bewijzen de resultaten in deze thesis dat een eenvoudige (N)ES wel degelijk in staat is om parameters van neurale netwerken te evolueren voor tekstgeneratie, en dat het een verbetering t.o.v. teacher forcing kan bieden. Dat opent de deur voor verdere ontwikkelingen, die de performantie ongetwijfeld nog kunnen verbeteren.

Zowel de rekenkracht als de tijd die voor het maken van deze thesis beschikbaar waren, stonden niet toe dieper in te gaan op belovende aspecten van NIC-(N)ES, noch om de gebruikte hyperparameters voldoende te tunen. Het feit dat met de zeer beperkte middelen en de ruw afgestelde hyperparameters in deze thesis de scores verbeterd kon worden ten opzichte van teacher forcing doet vermoeden dat er nog veel meer mogelijk is. De volgende sectie suggereert uitbreidingen of nieuwe richtingen die in het bijzonder veelbelovend zijn. Salimans et al. lieten hun NES, ter vergelijking, lopen op 1440 CPU-cores [75]. De ES van Such et al. liep op 720 cores [83]. Deze thesis had slechts een zeer beperkt computationeel budget op machines met 24-96 vCPU's, een verschil van een grootte-orde.

Hardware. De experimenten bevestigen dat NIC-ES en NIC-NES profijt halen uit parallelisatie. Naast de hoogste bereikte CIDEr-score, die bij SCST op een

4. EXPERIMENTEN & RESULTATEN

GPU hoger is dan bij NIC-(N)ES, bepaalt ook de beschikbare hardware mee de keuze voor een algoritme. Zoals vermeld rekenen algoritmes zoals NIC-(N)ES op heel andere hardware dan teacher forcing of policy gradients. Deep learning algoritmes en frameworks van tegenwoordig zijn volledig toegespitst op één soort hardware, namelijk krachtige vector processing units. Evolutionaire algoritmes, die kunnen profiteren van parallelle CPU's, kunnen een nuttige uitbreiding zijn aan de toolbox van deep learning voor wanneer de beschikbare hardware verschilt. Mits kleine aanpassingen kan NIC-(N)ES ook op gedistribueerde systemen uitgevoerd worden. Algoritmes als NIC-(N)ES zouden dus ook van cloud of grid computing gebruik kunnen maken.

Hyperparameters. Onder andere experimenten in secties 4.2.3 en 4.2.5 maakten duidelijk dat NIC-ES geen hoge robuustheid tegenover de hyperparameters vertoont, en dus veel hyperparameter tuning vraagt. NIC-NES blijkt robuuster tegenover de hyperparameters.

NIC-ES en voortijdige convergentie. Uit experimenten in sectie 4.2.5 blijkt dat NIC-ES betere resultaten haalt met een zeer hoge selectiedruk: d.w.z. een zeer kleine populatiegrootte μ zodat de μ/λ verhouding ver onder de aanbevolen 1/7 ligt. Hoe kleiner μ is, hoe meer het verloop van NIC-ES zal lijken op dat van een lokale zoektocht in de parameterruimte zoals in een *hill-climbing*-algoritme. Dat σ klein moet blijven, zodat de mutaties niet te verstorend zijn voor de functionaliteit, zorgt ervoor dat het zeer moeilijk zal zijn om te ontsnappen aan lokale optima of vlakke delen in de parameterruimte.

Het feit dat NIC-ES met de gekozen hyperparameters vatbaar lijkt te zijn voor lokale optima of plateaus kan ongetwijfeld een gevolg zijn van de manier waarop de hyperparameters gekozen zijn. Die manier beoordeelt namelijk hyperparameters die niet per se uiteindelijk tot de hoogste scores leiden, maar die wel zo snel mogelijk tot vrij hoge scores leiden. Dat is juist waar een te hoge selectiedruk voor zorgt. Meer experimenten met lagere selectiedruk en grotere σ zouden deze hypothese kunnen testen.

De vatbaarheid van NIC-ES voor plateaus of lokale optima doet vermoeden dat er niet genoeg diversiteit in de populatie aanwezig is. Het feit dat NIC-ES enkel mutaties gebruikt (met een relatief kleine σ) en geen recombinatie, draagt hier aan bij. Ook dat NIC-ES door de moeilijkheid van het probleem met één of enkele met teacher forcing voorgetrainde netwerken moet starten kan hier aan bijdragen. Een volledige willekeurig geïnitialiseerde populatie heeft natuurlijk meer diversiteit dan een populatie van nakomelingen van slechts enkele individuen.

Het zou dus kunnen dat NIC-ES eigenlijk niet het beste werkt met de kleine μ en UWS uit sectie 4.2.5, maar dat er te weinig diversiteit in de populatie is. Dat probleem geraakt niet opgelost door een grotere μ , eventueel met toernooiselectie, te gebruiken.

NIC-NES. NIC-NES krijgt zoals alle optimalisatiemethoden uiteraard ook te kampen met plateaus of lokale optima. Uit de experimenten (bv. in sectie 4.2.1) blijkt echter dat NIC-NES hier meer tegen gewapend is dan NIC-ES, het volgende kan dat verklaren. Het gebruik van het Adam-algoritme introduceert namelijk een extra stap tussen de mutatievectoren en de update aan de huidige parameters. In de mutatievectoren (bij standaardmutaties) zullen zowel bij NIC-ES als bij NIC-NES 99.7% van de elementen binnen $[-3\sigma, 3\sigma]$ liggen, omdat de mutatievectoren gesampled worden uit een normaalverdeling. Bij NIC-ES zijn alle nakomelingen bepaald door $\theta + \delta$ en daarom bepaalt σ dus rechtstreeks een (zij het niet-deterministische of “zachte”) bovengrens voor het verschil tussen een nakomeling en zijn ouder.

Bij NIC-NES echter berekent Adam uit de gecombineerde mutaties een update voor θ . Daarvoor wordt niet alleen de huidige gradiënt gebruikt, maar ook een lopend gemiddelde van statistische momenten van de vorige gradiënten [37]. Door de impact die vorige gradiënten hebben bouwt het algoritme *momentum* op en kan het door regio’s geraken waar de huidige gradiënt nul of zeer klein is. De mogelijkheid om bestaande, geavanceerde algoritmes voor gradient descent zoals Adam te gebruiken is dus een voordeel van NIC-NES.

Evaluatiefunctie. De resultaten uit sectie 4.2.4 onderschrijven het belang van goede evaluatiefuncties. De flexibiliteit van EA’s tegenover de evaluatiefunctie, die het mogelijk maakt om de niet-differentieerbare CIDEr te gebruiken, betekent niet dat zomaar alle functies ook goed werken. Dat blijkt uit de slechte resultaten met F_{sam} en F_{scst} . Ook werd ondervonden door F_{scst} dat lossfuncties die goed werken met SGD en backpropagation daarom niet goed werken als evaluatiefunctie in EA’s, door het structurele verschil in wat beide soorten algoritmes doen met de uitkomst van de loss-of evaluatiefunctie. De SCST-lossfunctie is een voorbeeld van een gespecialiseerde en speciaal ontworpen functie voor policy gradients. De evaluatiefunctie $F_{gr}(\theta)$ die NIC-(N)ES gebruikt is daarentegen eenvoudig, en kan ongetwijfeld nog verbeterd worden.

Overfitting. Bij experimenten met NIC-NES zonder gelijke minibatches (secties 4.2.1, 4.2.6) is een iteratie gebaseerd op een groot deel van de trainingset. Daardoor kan de gemiddelde fitness van alle nakomelingen in één iteratie beschouwd worden als schatter voor de fitness van het huidige individu op heel de trainingset. Deze trainingscore eindigt consequent ongeveer 5% hoger dan de validatiescore. Dat suggereert dat er overfitting plaatsvindt. Hoewel een deel van deze overfitting volgt uit de voortraining met teacher forcing zonder regularisatie, stijgt het verschil tussen de trainingscore en de validatiescore ook nog tijdens het trainen met NIC-NES. Enkele experimenten met L2-regularisatie, om deze overfitting te voorkomen, gaven goede resultaten (sectie 4.2.1). De fitness-waarden volgen echter wel uit gemuteerde versies van het huidige individu. Dat kan betekenen dat de schatter niet zuiver of niet nauwkeurig is. Een verdere analyse is dus nodig, ook voor NIC-ES.

4.4 Suggesties voor verder onderzoek

Het samenkommen van deep learning en evolutionaire algoritmen in neuro-evolutie maakt het mogelijk om, naast het ontwikkelen van helemaal nieuwe technieken of methoden, tal van bestaande ideeën uit beide gebieden over te nemen of aan te passen aan de context. In deze sectie worden enkele ideeën gesuggereerd die de efficiëntie of performantie van NIC-NES en NIC-ES zouden kunnen verbeteren.

Andere taken en andere architecturen. Het zou interessant zijn om de algoritmes toe te passen op taalmodellen die andere architecturen gebruiken dan RNN's. In sectie 2.2 werd al vermeld dat ook met CNN's en attention-modellen interessante resultaten gehaald zijn op NLP-taken. Misschien blijken EA's meer geschikt voor feedforward NNs.

Afbeeldingen ondertitelen met LSTM's is slechts een voorbeeld van een taak waarvoor een (N)ES gebruikt kan worden. Verder onderzoek kan NIC-(N)ES gebruiken voor andere taken die tekstgeneratie omvatten zoals dialogen voeren of tekst vertalen. Daarbij kan NIC-(N)ES op een vrij analoge manier toegepast worden, gezien veel van de componenten niet probleemspecifiek zijn.

Hyperparameters. In verder onderzoek kunnen de hyperparameters met een meer gestructureerde aanpak gezocht worden. Als testprobleem zou dezelfde tekstgeneratietaak genomen kunnen worden maar met een beperktere woordenschat, een netwerk van dezelfde architectuur maar met minder parameters, en een kleinere dataset. Door hyperparameters te vergelijken op volledige runs op het testprobleem in plaats van op partiële runs op het echte, moeilijke probleem, kan vermeden worden dat er hyperparameters gekozen worden die snelle stijging bevoordelen ten koste van de uiteindelijk hoogste bereikte score.

Evaluatiefunctie. Zoals vermeld in sectie 4.3 is een goede evaluatiefunctie van essentieel belang. Het zou interessant zijn om de huidige, eenvoudige functie $F_{gr}(\theta)$ uit te breiden of om te experimenteren met andere evaluatiefuncties.

Zo optimaliseren de algoritmes in deze thesis enkel voor de CIDEr-score. Er zou ook voor andere scores of voor een combinatie van scores geoptimaliseerd kunnen worden, bijvoorbeeld voor de SPICE-score [2].

Een andere mogelijkheid is om de notie van fitness geheel of gedeeltelijk te vervangen door novelty [14] [47] [83]. Novelty kan genomen worden als de euclidische afstand tussen de som van de embeddings van woorden uit gegenereerde zinnen, voor een referentiebatch. Ook de *word mover's distance* tussen gegenereerde zinnen kan gebruikt worden [41]. Zo wordt exploratie expliciet aangespoord, en kan mogelijks het diversiteitsprobleem in NIC-ES verlicht worden.

F_{gr} zou ook verbeterd kunnen worden door de kans die het model toekent aan de uitvoer een rol te geven, bv. door $\log p_{model}$ te gebruiken. Onder andere in policy gradients en bij de cross-entropy komt deze kans ook voor in de lossfunctie of in de gradiënt ervan. Zo worden modellen getraind om niet alleen goede uitvoer te geven maar daar ook zeker van te zijn.

Diversiteit. Een voor de hand liggende remedie voor het diversiteitsprobleem in NIC-ES is de introductie van een recombinatie-operator. Recombinatie zou een tweede bron van nieuwe individuen vormen. Daarenboven zijn in tegenstelling tot bij mutatie de nakomelingen uit recombinatie niet beperkt tot de directe buurt in de θ -ruimte van hun ouders.

Ook andere tactieken voor de preservatie van diversiteit kunnen gebruikt worden, zoals speciation [19].

Zelf-adaptatie. Zelf-adaptatie is een meer gesofisticeerd alternatief voor de huidige curriculumptactieken. In NIC-ES kan een σ_i per parameter aan het genotype toegevoegd worden, zo kan evolutie de mutatiesterkte per parameter bepalen. Het toevoegen van één σ voor alle parameters aan het genotype, kan ervoor zorgen dat σ ook kan toenemen in plaats van afnemen tijdens de training, om uit lokale optima te geraken [19]. In NIC-NES kan een gelijkaardig effect bekomen worden door σ , net als het gemiddelde van verdeling p_ψ , ook variabel te maken en met gradient descent aan te passen [96].

Beginnen van willekeurige individuen. Om het toch mogelijk te maken om NIC-(N)ES voor heel de training te gebruiken, zonder voortraining, kan in verder onderzoek in het begin van de training een andere evaluatieprocedure gebruikt worden. NIC-(N)ES kan in die eerste procedure het netwerk in elke tijdstap onder teacher forcing het volgende woord laten voorspellen, d.w.z. gegeven het vorige ground-truth-woord. De fitness van een netwerk kan de cross-entropy tussen de voorspelde woorden en de ground-truth-woorden zijn. Na genoeg training met de eerste evaluatieprocedure zou dan, zoals van teacher forcing naar policy gradients, overgeschakeld kunnen worden naar de moeilijkere, huidige evaluatieprocedure, die dan weer zorgt voor hogere scores.

Regularisatie. De overfitting bij NIC-NES en ook bij NIC-ES zou verder geverifieerd kunnen worden. De eerste experimenten met L2-regularisatie bij NIC-NES gaven goede resultaten, maar er zouden meer experimenten gedaan kunnen worden om het effect van regularisatie verder te testen. Salimans et al. gebruiken bv. ook L2-regularisatie [75]. Andere vormen van regularisatie zoals dropout zouden ook getest kunnen worden [79].

Hoofdstuk 5

Conclusie

Deze thesis verkende het gebruik van neuro-evolutie in tekstgeneratie. Twee algoritmes werden voorgesteld om de parameters te zoeken van neurale netwerken die ondertitels voor afbeeldingen genereren: NIC-ES en NIC-NES, respectievelijk een evolutiestrategie en een natuurlijke evolutiestrategie.

Twee argumenten motiveren het toepassen van neuro-evolutie op tekstgeneratie. Ten eerste lijdt teacher forcing, de gangbare manier om modellen te trainen voor tekstgeneratie met gradient descent en backpropagation, onder twee discrepanties tussen training en inferentie. Deze discrepanties zijn de exposure bias en het feit dat er voor een andere lossfunctie (cross-entropy) getraind wordt dan de functie waarmee modellen uiteindelijk geëvalueerd worden. Policy gradients werden al met veel succes toegepast om deze discrepanties te verhelpen. NIC-ES en NIC-NES trainen netwerken, net als policy gradients, voor tekstgeneratie geformuleerd als reinforcement-learning-probleem, en overkomen daarmee op dezelfde manier de vermelde afwijkingen.

Ten tweede kunnen NIC-ES en NIC-NES interessante alternatieven zijn voor gradient descent met backpropagation omdat ze gebruik maken van zeer verschillende hardware. Terwijl gradient descent en backpropagation sequentieel zijn en het meest gebaat bij krachtige vector processing units, zijn NIC-ES en NIC-NES inherent parallel en kunnen ze uitgevoerd worden op grote aantallen CPU's. Door een lage bandbreedte voor communicatie kunnen ze bovendien, mits kleine aanpassingen, op gedistribueerde machines uitgevoerd worden. Door de opkomst van parallelisatie en cloud computing zou het interessant zijn als deep learning ook gebruik zou kunnen maken van gedistribueerde en parallele systemen.

De experimenten toonden aan dat de CIDEr-score, ten opzichte van teacher forcing, nog verbeterd kan worden met neuro-evolutie. Voor deze experimenten werden de algoritmes gestart van een met teacher forcing voorgetraind netwerk. De nodige tijd voor convergentie was bij een policy-gradient-methode op een GPU en NIC-(N)ES op 96 vCPU's vergelijkbaar, hoewel de policy-gradient-methode op een significant hogere score eindigt.

Wanneer de algoritmes geïnitialiseerd werden met willekeurige netwerken, werd

5. CONCLUSIE

geen convergentie bekomen. Deze bevinding kan verklaard worden door de enorme grootte van de zoekruimte van zinnen tegenover die van individuele woorden. Hoewel voortraining essentieel is, bewijzen de resultaten dat EA's wel degelijk in staat zijn om netwerken met enkele miljoenen parameters te evolueren voor tekstgeneratie.

Enkele specifieke aspecten van de algoritmes werden eveneens vergeleken, zo kunnen in verder onderzoek de goed werkende aspecten overgenomen worden. Bij beide algoritmes werden mutaties getest waarvan de grootte afhankelijk is van de parameter die gemuteerd wordt, in plaats van mutaties die alle parameters met dezelfde sterkte muteren. Bij NIC-ES zorgde proportionele mutaties voor een sterke toename in snelheid van convergentie, maar geen toename in de bereikte score. De mutatie met uitvoergradiënten zorgde voor een vertraging van de convergentie. Bij NIC-NES zorgden mutaties met uitvoergradiënten voor een toename in zowel convergentiesnelheid als de hoogste score. De proportionele mutaties hadden het tegenovergestelde effect.

Bij beide algoritmes werd alleen een stijging in de score bekomen wanneer als evaluatiefunctie de CIDEr-score onder greedy decoding werd gebruikt. De CIDEr-score onder samplen van de uitvoerverdeling geeft een te noisy beeld van de werkelijke fitness. De SCST-lossfunctie bleek geen goede representatie van de performantie van een netwerk.

Vervolgens werd ondervonden dat uniforme willekeurige selectie met een relatief kleine populatie bij NIC-ES beter werkt dan toernooiselectie met een grotere populatie. NIC-ES lijkt op het eerste zicht dus het beste te werken met een heel hoge selectiedruk. Maar, zoals o.a. de lagere scores die NIC-ES tegenover NIC-NES haalt suggereren, stagneert de score bij NIC-ES voortijdig, in lokale optima of vlakke delen van de zoekruimte. Dat doet vermoeden dat er niet genoeg diversiteit in de populatie van NIC-ES aanwezig is. De noodzakelijk kleine mutatiesterkte, het feit dat er geen recombinatie gebruikt wordt, en de initialisatie met slechts één of enkele individuen liggen mee aan de oorzaak van het gebrek aan diversiteit.

In NIC-NES was het voordelig om alle evaluaties binnen een iteratie te doen op verschillende minibatches, in plaats van op één gelijke minibatch. Zo is de schatter voor de gradiënt, op basis waarvan de parameters een update krijgen, gebaseerd op een zo groot mogelijk deel van de dataset.

De algoritmes in deze thesis zijn zeer eenvoudige algoritmes voor neuro-evolutie, en bovendien werden de hyperparameters zeer ruwweg gekozen. Verder onderzoek kan de algoritmes uitbreiden en de hyperparameters fijner afstellen. Zoals in onderdeel 4.4 beschreven zijn er nog tal van mogelijke uitbreidingen, uit zowel deep learning als evolutionaire algoritmes en neuro-evolutie.

Hoewel de kloof tussen backpropagation en GPU's enerzijds en neuro-evolutie en CPU's anderzijds nog niet gedicht is op vlak van performantie in tekstgeneratie, zetten NIC-ES en NIC-NES een stap verder in die richting.

Referenties

- [1] Timo Aaltonen e.a. “Measurement of the top-quark mass with dilepton events selected using neuroevolution at CDF”. In: *Physical Review Letters* 102.15 (2009), p. 152001.
- [2] Peter Anderson, Basura Fernando, Mark Johnson en Stephen Gould. “SPICE: Semantic propositional image caption evaluation”. In: *European Conference on Computer Vision*. Springer. 2016, p. 382–398.
- [3] Peter Anderson e.a. “Bottom-up and top-down attention for image captioning and visual question answering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, p. 6077–6086.
- [4] Peter J Angeline, Gregory M Saunders en Jordan B Pollack. “An evolutionary algorithm that constructs recurrent neural networks”. In: *IEEE Transactions on Neural Networks* 5.1 (1994), p. 54–65.
- [5] Michael Auli, Michel Galley, Chris Quirk en Geoffrey Zweig. “Joint Language and Translation Modeling with Recurrent Neural Networks”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013, p. 1044–1054.
- [6] Dzmitry Bahdanau, Kyunghyun Cho en Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).
- [7] Marc G Bellemare, Yavar Naddaf, Joel Veness en Michael Bowling. “The arcade learning environment: An evaluation platform for general agents”. In: *Journal of Artificial Intelligence Research* 47 (2013), p. 253–279.
- [8] Samy Bengio, Oriol Vinyals, Navdeep Jaitly en Noam Shazeer. “Scheduled sampling for sequence prediction with recurrent neural networks”. In: *Advances in Neural Information Processing Systems*. 2015, p. 1171–1179.
- [9] Yoshua Bengio, Réjean Ducharme, Pascal Vincent en Christian Jauvin. “A neural probabilistic language model”. In: *Journal of Machine Learning Research* 3.Feb (2003), p. 1137–1155.
- [10] Yoshua Bengio, Patrice Simard, Paolo Frasconi e.a. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), p. 157–166.

- [11] Dimo Brockhoff, Anne Auger, Nikolaus Hansen, Dirk V Arnold en Tim Hohm. “Mirrored sampling and sequential selection for evolution strategies”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2010, p. 11–21.
- [12] Xinlei Chen e.a. “Microsoft COCO captions: Data collection and evaluation server”. In: *arXiv preprint arXiv:1504.00325* (2015).
- [13] Kyunghyun Cho e.a. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).
- [14] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley en Jeff Clune. “Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents”. In: *Advances in Neural Information Processing Systems*. 2018, p. 5027–5038.
- [15] Frank E Curtis en Katya Scheinberg. “Optimization methods for supervised machine learning: From linear models to deep learning”. In: *Leading Developments from INFORMS Communities*. INFORMS, 2017, p. 89–114.
- [16] Yann N Dauphin, Angela Fan, Michael Auli en David Grangier. “Language modeling with gated convolutional networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, p. 933–941.
- [17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li en Li Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, p. 248–255.
- [18] Jeffrey Donahue e.a. “Long-term recurrent convolutional networks for visual recognition and description”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, p. 2625–2634.
- [19] Agoston E Eiben, James E Smith e.a. *Introduction to evolutionary computing*. Deel 53. Springer, 2003.
- [20] Jeffrey L Elman. “Finding structure in time”. In: *Cognitive Science* 14.2 (1990), p. 179–211.
- [21] David B Fogel, Lawrence J Fogel en VW Porto. “Evolving neural networks”. In: *Biological Cybernetics* 63.6 (1990), p. 487–493.
- [22] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats en Yann N Dauphin. “Convolutional sequence to sequence learning”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, p. 1243–1252.
- [23] Xavier Glorot en Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, p. 249–256.
- [24] Ian Goodfellow, Yoshua Bengio en Aaron Courville. *Deep learning*. MIT press, 2016.

-
- [25] Ian Goodfellow e.a. “Generative adversarial nets”. In: *Advances in Neural Information Processing Systems*. 2014, p. 2672–2680.
 - [26] Alex Graves. “Generating sequences with recurrent neural networks”. In: *arXiv preprint arXiv:1308.0850* (2013).
 - [27] Alex Graves. *Supervised sequence labelling*. Springer, 2012, p. 12–38.
 - [28] Nikolaus Hansen en Andreas Ostermeier. “Completely derandomized self-adaptation in evolution strategies”. In: *Evolutionary Computation* 9.2 (2001), p. 159–195.
 - [29] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen en Peter Stone. “A neuroevolution approach to general atari game playing”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (2014), p. 355–366.
 - [30] Kaiming He, Xiangyu Zhang, Shaoqing Ren en Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, p. 770–778.
 - [31] Sepp Hochreiter en Jürgen Schmidhuber. “Long short-term memory”. In: *Neural Computation* 9.8 (1997), p. 1735–1780.
 - [32] Christian Igel. “Neuroevolution for reinforcement learning using evolution strategies”. In: *The 2003 Congress on Evolutionary Computation, 2003. CEC'03*. Deel 4. IEEE. 2003, p. 2588–2595.
 - [33] Sergey Ioffe en Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International Conference on Machine Learning*. 2015, p. 448–456.
 - [34] Nal Kalchbrenner, Edward Grefenstette en Phil Blunsom. “A convolutional neural network for modelling sentences”. In: *arXiv preprint arXiv:1404.2188* (2014).
 - [35] Andrej Karpathy en Li Fei-Fei. “Deep visual-semantic alignments for generating image descriptions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, p. 3128–3137.
 - [36] Yoon Kim. “Convolutional neural networks for sentence classification”. In: *arXiv preprint arXiv:1408.5882* (2014).
 - [37] Diederik P Kingma en Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
 - [38] Ryan Kiros, Ruslan Salakhutdinov en Richard S Zemel. “Unifying visual-semantic embeddings with multimodal neural language models”. In: *arXiv preprint arXiv:1411.2539* (2014).
 - [39] Ranjay Krishna e.a. “Visual genome: Connecting language and vision using crowdsourced dense image annotations”. In: *International Journal of Computer Vision* 123.1 (2017), p. 32–73.

- [40] Alex Krizhevsky, Ilya Sutskever en Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems*. 2012, p. 1097–1105.
- [41] Matt Kusner, Yu Sun, Nicholas Kolkin en Kilian Weinberger. “From word embeddings to document distances”. In: *International Conference on Machine Learning*. 2015, p. 957–966.
- [42] Alex M Lamb, Anirudh Goyal Alias Parth Goyal, Ying Zhang, Saizheng Zhang, Aaron C Courville en Yoshua Bengio. “Professor forcing: A new algorithm for training recurrent networks”. In: *Advances In Neural Information Processing Systems*. 2016, p. 4601–4609.
- [43] Y LeCun en C Cortes. “The MNIST database of handwritten digits”. In: <http://yann. lecun. com/exdb/mnist/> (1998).
- [44] Yann LeCun, Yoshua Bengio en Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), p. 436.
- [45] Joel Lehman, Jay Chen, Jeff Clune en Kenneth O Stanley. “ES is more than just a traditional finite-difference approximator”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2018, p. 450–457.
- [46] Joel Lehman, Jay Chen, Jeff Clune en Kenneth O Stanley. “Safe mutations for deep and recurrent neural networks through output gradients”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2018, p. 117–124.
- [47] Joel Lehman en Kenneth O Stanley. “Abandoning objectives: Evolution through the search for novelty alone”. In: *Evolutionary Computation* 19.2 (2011), p. 189–223.
- [48] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley en Jianfeng Gao. “Deep Reinforcement Learning for Dialogue Generation”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, p. 1192–1202.
- [49] Tsung-Yi Lin e.a. “Microsoft coco: Common objects in context”. In: *European Conference on Computer Vision*. Springer. 2014, p. 740–755.
- [50] Siqi Liu, Zhenhai Zhu, Ning Ye, Sergio Guadarrama en Kevin Murphy. “Improved image captioning via policy gradient optimization of spider”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, p. 873–881.
- [51] Henry B Mann en Donald R Whitney. “On a test of whether one of two random variables is stochastically larger than the other”. In: *The Annals of Mathematical Statistics* (1947), p. 50–60.
- [52] Risto Miikkulainen e.a. “Evolving deep neural networks”. In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, p. 293–312.

-
- [53] Tomas Mikolov, Kai Chen, Greg Corrado en Jeffrey Dean. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).
 - [54] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado en Jeff Dean. “Distributed representations of words and phrases and their compositionality”. In: *Advances in Neural Information Processing Systems*. 2013, p. 3111–3119.
 - [55] Geoffrey F Miller, Peter M Todd en Shailesh U Hegde. “Designing Neural Networks using Genetic Algorithms.” In: *ICGA*. Deel 89. 1989, p. 379–384.
 - [56] Volodymyr Mnih e.a. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529.
 - [57] Volodymyr Mnih e.a. “Asynchronous methods for deep reinforcement learning”. In: *International Conference on Machine Learning*. 2016, p. 1928–1937.
 - [58] David E Moriarty en Risto Mikkulainen. “Efficient reinforcement learning through symbiotic evolution”. In: *Machine Learning* 22.1-3 (1996), p. 11–32.
 - [59] Gregory Morse en Kenneth O Stanley. “Simple evolutionary optimization can rival stochastic gradient descent in neural networks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM. 2016, p. 477–484.
 - [60] John Nicklow e.a. “State of the art for genetic algorithms and beyond in water resources planning and management”. In: *Journal of Water Resources Planning and Management* 136.4 (2009), p. 412–432.
 - [61] Hyeonwoo Noh, Seunghoon Hong en Bohyung Han. “Learning deconvolution network for semantic segmentation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, p. 1520–1528.
 - [62] Kishore Papineni, Salim Roukos, Todd Ward en Wei-Jing Zhu. “BLEU: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 2002, p. 311–318.
 - [63] Adam Paszke e.a. “Automatic differentiation in pytorch”. In: *NIPS-W*. 2017.
 - [64] Romain Paulus, Caiming Xiong en Richard Socher. “A deep reinforced model for abstractive summarization”. In: *arXiv preprint arXiv:1705.04304* (2017).
 - [65] Marc’Aurelio Ranzato, Sumit Chopra, Michael Auli en Wojciech Zaremba. “Sequence level training with recurrent neural networks”. In: *arXiv preprint arXiv:1511.06732* (2015).
 - [66] Esteban Real, Alok Aggarwal, Yanping Huang en Quoc V Le. “Regularized evolution for image classifier architecture search”. In: *arXiv preprint arXiv:1802.01548* (2018).
 - [67] Esteban Real e.a. “Large-scale evolution of image classifiers”. In: *Proceedings of the 34th International Conference on Machine Learning- Volume 70*. JMLR.org. 2017, p. 2902–2911.

- [68] Ingo Rechenberg. "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution". In: *Stuttgart, Germany* (1973).
- [69] Shaoqing Ren, Kaiming He, Ross Girshick en Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in Neural Information Processing Systems*. 2015, p. 91–99.
- [70] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross en Vaibhava Goel. "Self-critical sequence training for image captioning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, p. 7008–7024.
- [71] Sebastian Risi en Kenneth O Stanley. "An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons". In: *Artificial Life* 18.4 (2012), p. 331–363.
- [72] Xin Rong. "Word2vec parameter learning explained". In: *arXiv preprint arXiv:1411.2738* (2014).
- [73] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams e.a. "Learning representations by back-propagating errors". In: *Cognitive Modeling* 5.3 (1988), p. 1.
- [74] Olga Russakovsky e.a. "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3 (2015), p. 211–252.
- [75] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor en Ilya Sutskever. "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (2017).
- [76] Jose Santamaría, Oscar Cordón en Sergio Damas. "A comparative study of state-of-the-art evolutionary image registration methods for 3D modeling". In: *Computer Vision and Image Understanding* 115.9 (2011), p. 1340–1354.
- [77] Jürgen Schmidhuber, Daan Wierstra, Matteo Gagliolo en Faustino Gomez. "Training recurrent networks by evolino". In: *Neural Computation* 19.3 (2007), p. 757–779.
- [78] H-P Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhäuser, 1977.
- [79] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever en Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), p. 1929–1958.
- [80] Kenneth O Stanley. "Compositional pattern producing networks: A novel abstraction of development". In: *Genetic Programming and Evolvable Machines* 8.2 (2007), p. 131–162.
- [81] Kenneth O Stanley, David B D'Ambrosio en Jason Gauci. "A hypercube-based encoding for evolving large-scale neural networks". In: *Artificial Life* 15.2 (2009), p. 185–212.

-
- [82] Kenneth O Stanley en Risto Miikkulainen. “Evolving neural networks through augmenting topologies”. In: *Evolutionary Computation* 10.2 (2002), p. 99–127.
 - [83] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley en Jeff Clune. “Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning”. In: *arXiv preprint arXiv:1712.06567* (2017).
 - [84] Ilya Sutskever, James Martens en Geoffrey E Hinton. “Generating text with recurrent neural networks”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, p. 1017–1024.
 - [85] Ilya Sutskever, Oriol Vinyals en Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in Neural Information Processing Systems*. 2014, p. 3104–3112.
 - [86] Richard S Sutton en Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
 - [87] Richard S Sutton, David A McAllester, Satinder P Singh en Yishay Mansour. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in Neural Information Processing Systems*. 2000, p. 1057–1063.
 - [88] Emanuel Todorov, Tom Erez en Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, p. 5026–5033.
 - [89] Ashish Vaswani e.a. “Attention is all you need”. In: *Advances in Neural Information Processing Systems*. 2017, p. 5998–6008.
 - [90] Ramakrishna Vedantam, C Lawrence Zitnick en Devi Parikh. “Cider: Consensus-based image description evaluation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, p. 4566–4575.
 - [91] Arun Venkatraman, Martial Hebert en J Andrew Bagnell. “Improving multi-step prediction of learned time series models”. In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.
 - [92] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau en Christian Prins. “A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows”. In: *Computers & Operations Research* 40.1 (2013), p. 475–489.
 - [93] Oriol Vinyals en Quoc Le. “A neural conversational model”. In: *arXiv preprint arXiv:1506.05869* (2015).
 - [94] Oriol Vinyals, Alexander Toshev, Samy Bengio en Dumitru Erhan. “Show and tell: A neural image caption generator”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, p. 3156–3164.
 - [95] Paul J Werbos e.a. “Backpropagation through time: What it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), p. 1550–1560.

- [96] Daan Wierstra, Tom Schaul, Jan Peters en Juergen Schmidhuber. “Natural evolution strategies”. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, p. 3381–3387.
- [97] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8.3-4 (1992), p. 229–256.
- [98] Ronald J Williams en David Zipser. “A learning algorithm for continually running fully recurrent neural networks”. In: *Neural Computation* 1.2 (1989), p. 270–280.
- [99] Lijun Wu, Fei Tian, Tao Qin, Jianhuang Lai en Tie-Yan Liu. “A Study of Reinforcement Learning for Neural Machine Translation”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, p. 3612–3621.
- [100] Kelvin Xu e.a. “Show, attend and tell: Neural image caption generation with visual attention”. In: *International Conference on Machine Learning*. 2015, p. 2048–2057.
- [101] Xin Yao. “Evolving artificial neural networks”. In: *Proceedings of the IEEE* 87.9 (1999), p. 1423–1447.
- [102] Xin Yao en Yong Liu. “A new evolutionary system for evolving artificial neural networks”. In: *IEEE Transactions on Neural Networks* 8.3 (1997), p. 694–713.
- [103] Tom Young, Devamanyu Hazarika, Soujanya Poria en Erik Cambria. “Recent trends in deep learning based natural language processing”. In: *IEEE Computational IntelligenCe Magazine* 13.3 (2018), p. 55–75.
- [104] Xingwen Zhang, Jeff Clune en Kenneth O Stanley. “On the relationship between the openai evolution strategy and stochastic gradient descent”. In: *arXiv preprint arXiv:1712.06564* (2017).

Bijlage A

Overzicht van de hyperparameters

Dit onderdeel geeft een overzicht van de hyperparameters van NIC-ES en NIC-NES. Tabellen 4.1 in onderdeel 4.2 bevatten de geteste en uiteindelijk gebruikte waarden.

Gemeenschappelijke hyperparameters:

- Het aantal nakomelingen per generatie λ : bij grotere λ ziet NIC-ES in elke generatie een groter deel van de lokale zoekruimte rond de $\boldsymbol{\theta}^{(i)}$ in de huidige populatie, maar de totale hoeveelheid rekenwerk van de workers stijgt proportioneel in λ .
- Het gebruikte type van mutaties: standaard (M-STD), proportioneel (VM-P) of veilige mutaties door uitvoergradiënten (VM-G).
- De standaardafwijking van de mutaties σ . De betekenis van σ verandert licht wanneer andere mutatievarianten gebruikt worden maar intuïtief blijft σ de sterkte van de mutaties.
- De gebruikte evaluatiefunctie: $F_{sam}(\boldsymbol{\theta})$, $F_{gr}(\boldsymbol{\theta})$, $F_{scst}(\boldsymbol{\theta})$.
- De dimensie van de word embeddings $\dim(\mathbf{W}_e)$ en de verborgen toestandsvectoren van de LSTM $\dim(\mathbf{h})$: hoe groter de twee dimensies, hoe groter het totaal aantal parameters in de genotypes $\dim(\boldsymbol{\theta})$.
- Het aantal datapunten in de minibatches: B .
- De grootte van de validatieset $|\mathcal{D}_{val}|$.
- Of en hoe σ , B en η (enkel NIC-NES) tijdens het trainen veranderen. Dat kan via patience, dan moet het aantal iteraties P zonder dat de validatiescore verbroken wordt gekozen worden. De tweede manier om dit te doen is schedule, dan moet ook een aantal iteraties H gekozen worden. De factoren waarmee de variabelen telkens veranderen zijn ook hyperparameters.

A. OVERZICHT VAN DE HYPERPARAMETERS

Bijkomende hyperparameters van **NIC-ES**:

- De grootte van de populatie μ : volgt uit een trade-off tussen exploitatie en exploratie. Wanneer μ groter is gaan ook minder goede individuen niet verloren, die mogelijk nog wel goede nakomelingen kunnen genereren. Dat betekent echter ook dat NIC-ES mogelijk meer tijd “verspilt” aan de generatie en evaluatie van minder goede individuen.
- Welk selectiemechanisme voor ouders gebruikt wordt: UWS of TS. Indien toernooiselectie gebruikt wordt: ook de grootte van de toernooien S . S bepaalt mee met μ/λ de selectiedruk: hoe groter S , hoe kleiner de kans dat minder goede individuen uit de populatie als ouder gekozen worden en nakomelingen in de volgende generatie kunnen hebben.
- Het aantal te evalueren elitekandidaten C per generatie en het aantal elites dat bijgehouden wordt E . Hoe groter C , hoe meer werk de workers elke generatie hebben met het evalueren van de elitekandidaten.

Bijkomende hyperparameters van **NIC-NES**:

- Het gebruikte gradient-descent-algoritme: Adam of stochastic gradient descent met momentum. Indien momentum gebruikt wordt moet ook de momentum-factor gekozen worden.
- De learning rate η voor gradient descent.
- De coëfficiënt voor een L2-regularisatieterm om overfitting te voorkomen.
- Of NIC-NES de evaluaties in een iteratie op dezelfde minibatch uitvoert of elke evaluatie op een verschillende minibatch.

Bijlage B

Voorbeelden van gegenereerde ondertitels

Figuren B.1 laten enkele foto's uit de testset zien. Hier volgen de ondertitels die een netwerk onder greedy decoding genereert voor deze foto's, na training met NIC-ES, NIC-NES, XENT en SCST zoals in sectie 4.2.1. Telkens de eerste van de 5 ground-truth-zinnen uit de dataset wordt vermeld. De voorbeelden zijn zo gekozen dat de ondertitels van de verschillende algoritmes verschillen, en niet zodat een algoritme duidelijk beter is dan een ander. De implementatie van NIC-(N)ES bevat code om ook andere ondertitels voor foto's uit de testset te inspecteren.

Ondertitels foto B.1a:

- Ground-truth: “a group of children playing baseball out side.”
- NIC-NES: “a group of young people playing a game of frisbee.”
- NIC-ES: “a group of young people playing a game of baseball.”
- XENT: “a group of people standing around a field.”
- SCST: “a group of people playing baseball on a field.”

Ondertitels foto B.1b:

- Ground-truth: “a child leans into the camera brushing their teeth.”
- NIC-NES: “a woman brushing her teeth with a green and white brush.”
- NIC-ES: “a woman with a spoon in her mouth.”
- XENT: “a woman with a knife cutting a cake.”
- SCST: “a woman eating a slice of pizza.”

Ondertitels foto B.1c:

- Ground-truth: ‘a boy attempts to hit the tennis ball with the racquet.’
- NIC-NES: “a young boy swinging a tennis racket at a tennis ball.”
- NIC-ES: “a tennis player is swinging a tennis racket.”
- XENT: “a young boy is playing tennis on a court.”

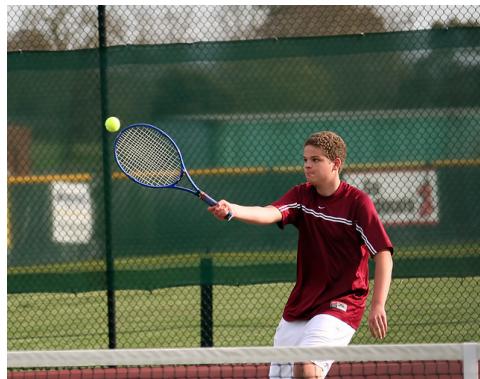
B. VOORBEELDEN VAN GEGENEREERDE ONDERTITELS



(a)



(b)



(c)



(d)

Figuur B.1: Vergelijking ondertitels

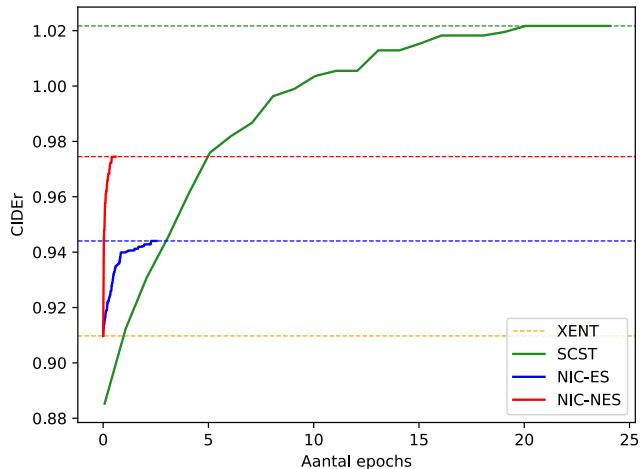
- SCST: “a young boy holding a tennis ball on a tennis court.”

Ondertitels foto [B.1d](#)

- Ground-truth: ‘a pile of teddy bears and dolls in a toy box.’
- NIC-NES: “a group of stuffed animals sitting on top of a table.”
- NIC-ES: “a stuffed animal with a stuffed toy bear and a stuffed animal.”
- XENT: “a stuffed animal with a stuffed animal on it.”
- SCST: “a group of stuffed teddy bears sitting on a table.”

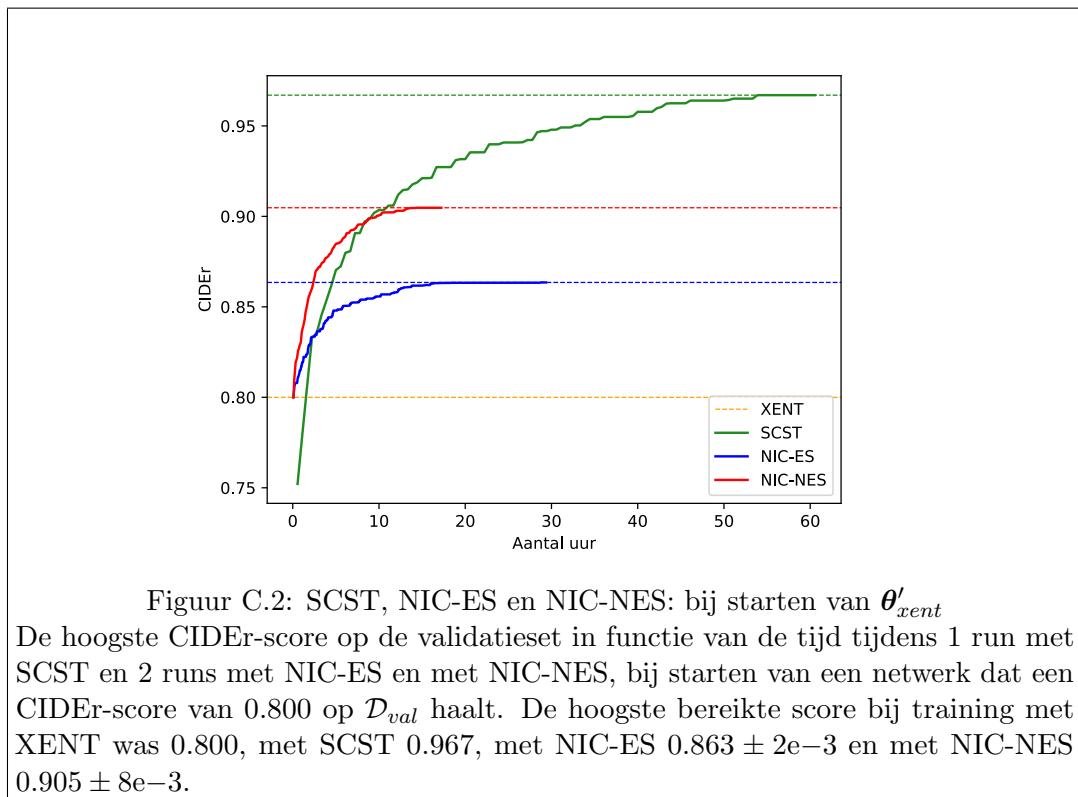
Bijlage C

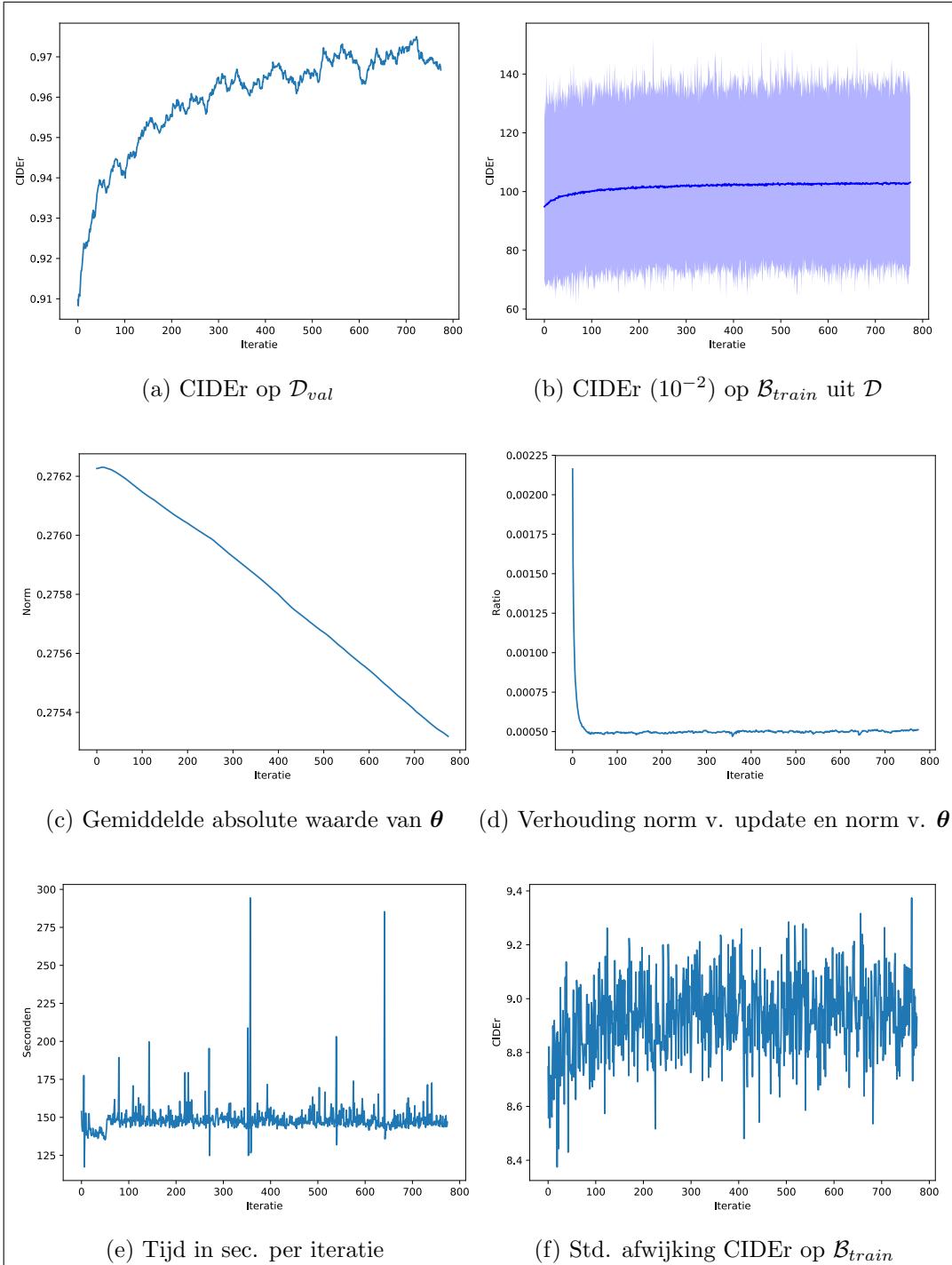
Plots van NIC-ES en NIC-NES tijdens het trainen



Figuur C.1: SCST, NIC-ES en NIC-NES: CIDEr in functie van het aantal epochs
De hoogste CIDEr-score op \mathcal{D}_{val} in functie van het aantal geziene samples (1 epoch = 113k samples), tijdens 1 run met SCST en 2 runs met NIC-ES en met NIC-NES.
De hoogste bereikte score bij training met XENT was 0.910, met SCST 1.022, met NIC-ES $0.944 \pm 7e-3$ en met NIC-NES $0.975 \pm 7e-4$. Gezien NIC-NES zonder gelijke minibatches werd uitgevoerd is de vergelijking echter niet helemaal eerlijk.

C. PLOTS VAN NIC-ES EN NIC-NES TIJDENS HET TRAINEN

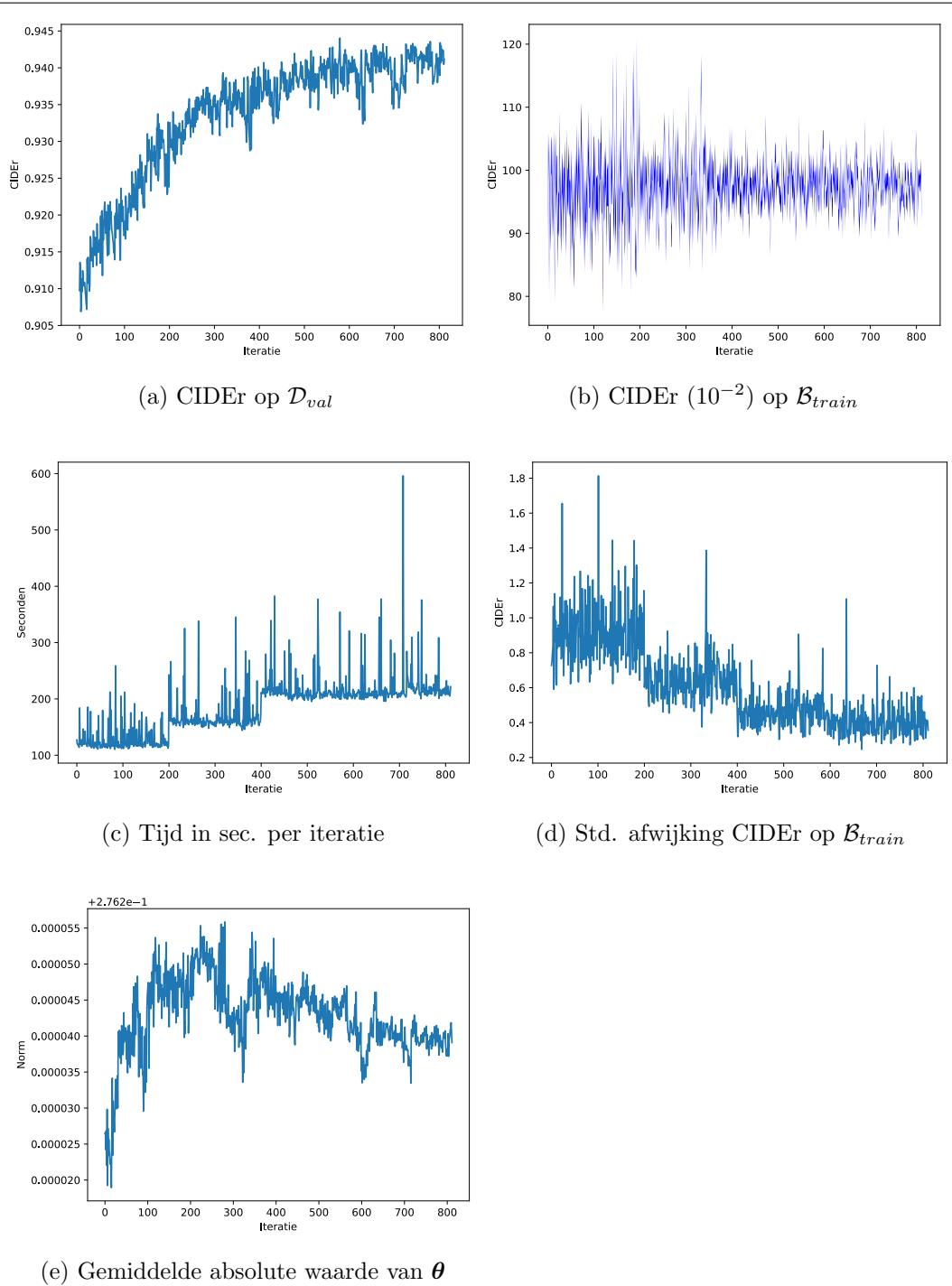




Figuur C.3: NIC-NES: trainingrun

Plots van NIC-NES tijdens een trainingrun op 96 vCPU's, met de hyperparameters uit tabel 4.1b. Figuur C.3b laat het minimum, maximum en gemiddelde zien van de CIDEr-scores op de batches \mathcal{B}_{train} uit \mathcal{D} waarop de nakomelingen geëvalueerd zijn.

C. PLOTS VAN NIC-ES EN NIC-NES TIJDENS HET TRAINEN



Figuur C.4: NIC-ES: trainingrun

Plots van NIC-ES tijdens een trainingrun op 96 vCPU's, met de hyperparameters uit tabel 4.1b. B neemt elke 200 iteraties toe met factor $\sqrt{2}$ tot 512, σ neemt elke 200 iteraties af met factor $\sqrt{2}$.

Fiche masterproef

Student: Ruben Cartuyvels

Titel: Evolutiestrategieën voor het ondertitelen van afbeeldingen met neurale netwerken

Engelse titel: Evolution strategies for neural image captioning

UDC: 681.3

Korte inhoud:

Deze thesis stelt voor om neurale netwerken voor tekstgeneratie te trainen met neuro-evolutie. Twee algoritmes worden voorgesteld: NIC-ES en NIC-NES, resp. een evolutiestrategie en een natuurlijke evolutiestrategie. Neurale netwerken worden tegenwoordig getraind voor tekstgeneratie door ze met cross-entropy te trainen om woord per woord, gegeven het vorige ground-truth-woord, het volgende ground-truth-woord te voorspellen. Tijdens inferentie moeten de netwerken echter hele zinnen ineens genereren, met de zelf voorspelde vorige woorden als invoer. Bovendien worden ze beoordeeld met andere evaluatiefuncties dan de cross-entropy, waar ze voor getraind zijn. Recent werden policy gradients met succes gebruikt om deze discrepancies te verhelpen. Zowel de klassieke training als training met policy gradients wordt echter met gradient descent en backpropagation doorgaans op dure vector processing units zoals GPU's uitgevoerd. NIC-ES en NIC-NES optimaliseren netwerken, net zoals policy gradients, voor het genereren van hele zinnen en niet van individuele woorden, en voor de relevante evaluatiefunctie. Beide algoritmes zijn zeer paralleliseerbaar, hebben geen GPU's nodig en kunnen op gedistribueerde systemen uitgevoerd worden. Daarom zouden ze kunnen profiteren van de opkomst van parallelisatie en cloud computing. In de experimenten worden NIC-ES en NIC-NES gebruikt voor het trainen van RNN's om ondertitels te genereren voor afbeeldingen uit de MSCOCO Captions dataset. Daarbij verbeteren ze de CIDEr-score op de testset t.o.v. training met cross-entropy met resp. 2.3% en 8.2%. De voortraining met cross-entropy blijft essentieel. NIC-ES en NIC-NES worden uitgevoerd op 96 CPU's in parallel en convergeren in vergelijkbare tijd met een policy-gradient-methode op een GPU. Drie varianten op mutaties en drie evaluatiefuncties worden voorgesteld en vergeleken in de experimenten, alsook twee selectiemechanismen in NIC-ES en twee manieren om minibatches te gebruiken in NIC-NES.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotor: Prof. dr. M.-F. Moens

Assessoren: Prof. dr. ir. D. Roose

Ir. T. Leeuwenberg

Begeleider: Ir. G. Spinks