FACULTEIT WETENSCHAPPEN

Distillatie van diepe reinforcement learning modellen

Arne Gevaert
Studentennummer: 01400391

Promotor: prof. dr. Yvan Saeys
Begeleider: Jonathan Peck

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de informatica

Academiejaar 2018-2019

UNIVERSITEIT GENT

Samenvatting

Neurale modellen hebben in het laatste decennium een heropleving meegemaakt. Zulke modellen worden echter algemeen beschouwd als *black boxes*. Dit betekent dat het zeer moeilijk is om te verklaren hoe het model tot een beslissing komt, of om een algemene intuïtie op te bouwen over de werking van het model. Interpretabiliteit is echter een cruciale eigenschap voor veel toepassingen waar een basisvertrouwen in het model een vereiste is, zoals o.a. in medische of juridische applicaties. Over het algemeen kunnen we besluiten dat technieken die toelaten om meer inzicht te verwerven in neurale modellen van groot belang zijn voor de wetenschap.

Knowledge distillation is een techniek die toelaat om de kennis uit een groot, complex model over te brengen naar een kleiner, simpeler model [1]. In dit werk wordt de focus gelegd op een specifieke implementatie van knowledge distillation, namelijk **policy distillation** met behulp van **vaaginferentie**. Policy distillation is een deelgebied van knowledge distillation, waarbij het idee wordt toegepast op reinforcement learning policies [2]. Vaaginferentie is de verzamelnaam voor machine learning-systemen die geïnspireerd zijn op principes uit de vaaglogica [3]. Vaaglogica is een uitbreiding van de klassieke logica die toelaat om met proposities te werken die slechts gedeeltelijk voldaan zijn. Dit laat onder andere toe om wiskundige modellen op te stellen die gemakkelijk interpreteerbaar en uitlegbaar zijn met behulp van natuurlijke, menselijke taal. Vaaginferentie-gebaseerde systemen vormen dus een ideale kandidaat voor policy distillation als we het primaire doel stellen op interpretabiliteit.

Distillation of Deep Reinforcement Learning models using Fuzzy Inference Systems

Arne Gevaert

May 30, 2019

Abstract

Policies for complex tasks have been successfully learned in the past using deep reinforcement learning. This approach uses a deep neural net to encode the policy, which achieves very good performance but is widely regarded as a black box model, meaning that its actions are not interpretable for humans. In this work, we present an algorithm to distill the policy from a deep Q-network into a more interpretable fuzzy inference system. We demonstrate this algorithm on the classic cartpole swing-up problem.

1 Introduction

Recently, significant progress has been made in the field of Deep Reinforcement Learning, with advances in a wide variety of application domains, such as arcade game playing [1], continuous control [2], and beating professional players in the game of GO [3]. Most of these advances have been made using deep neural networks, which are widely regarded as *black boxes*. This means that the inner workings of a deep neural network are hard to understand, making interpretation of the learned policy by humans a difficult task. However, interpretability is a critical property of machine learning models in many application domains, including legal and medical applications [4]. Recent EU legislation even requires many machine learning models to be interpretable by default [5].

In this work, we use policy distillation [6] to distill the learned policy from a deep Q-network to an ANFIS controller [7]. The advantage of neuro-fuzzy controllers such as ANFIS is that they can be trained much like a neural network, but can be much more interpretable. This interpretability is not intrinsic to the model however, and specific precautions must be taken to ensure that the result is as interpretable as possible [8]. For this reason, we extend the original policy distillation algorithm with a pre-processing and a post-processing step to maximize model interpretability [9].

We first explore policy distillation in section 2. Next, we give an overview of ANFIS, the neuro-fuzzy inference system used in our distillation algorithm, in section 3. In section 4 we then present the extended distillation algorithm. Then, we apply this algorithm to the OpenAI Gym Cartpole environment [10] and discuss results in section 5. Finally, we conclude in section 6.

2 Policy distillation

Distillation is a method to transfer knowledge from a *teacher* model T to a *student* model S. In supervised learning, this is done by using the output distributions of the teacher to train the student [11]. As these output distributions contain confidence levels of the teacher over all output classes, they contain more information than the original labels, allowing the student model to learn more efficiently. However, a fully trained teacher might assign almost 100% confidence to the correct class and almost 0% to the other classes, causing the extra information to be drowned out. For this reason, a *temperature softmax* is used to "soften" the output distribution of the teacher:

$$\sigma_\tau(\boldsymbol{z})_i = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)} \tag{1}$$

If the temperature τ is higher than 1, the result of this function is a smoothed out version of the original softmax function. The student is then trained on these smoothed out distributions using the classical cross-entropy loss function.
In policy distillation, this idea is applied to reinforcement learning agents. In the original work on policy distillation [6], a dataset was built by having the teacher model interact with the environment and recording the encountered states and output Q-values. The student was then trained to mimic this behaviour. As we require the student to output Q-values, which do not form valid probability distributions, we replace the cross-entropy loss with a *temperature Kullback-Leibler divergence* loss:

$$L_{KL}(\mathbf{q}^T, \mathbf{q}^S) = \sigma_\tau\left(\mathbf{q}^T\right) \ln \frac{\sigma_\tau\left(\mathbf{q}^T\right)}{\sigma\left(\mathbf{q}^S\right)} \tag{2}$$

Note that the output of a softmax function on a vector of Q-values is usually a very smooth probability distribution, as Q-values often have very small yet important differences. For this reason, the temperature τ in policy distillation is usually lower than 1, resulting in sharper probability distributions.
Recent advances in policy distillation have shown better empirical results if student trajectories are followed during distillation [12, 13]. This means that the student interacts with the environment, and is trained on the output of the teacher for visited states, rather than building a dataset by having the teacher interact with the environment. A possible explanation for this performance improvement is that the teacher policy is practically deterministic, while the student policy changes as the student is trained. This leads to a better exploration of the environment if student trajectories are followed.

3 Fuzzy control

Fuzzy control is the application of principles from fuzzy logic [14] to create control systems (*fuzzy controllers*). These fuzzy controllers usually consist of a set of fuzzy IF-THEN rules. This is a rule of the form IF x IS A THEN y IS B, where x is the input and y is the output. A is a fuzzy set, which is an extension of a classical set. In a classical set, every element is either a member or not a member of the set. In a fuzzy set, the membership of an element in the set can take any value between 0 (not a member) and 1 (entirely a member). The fuzzy set A is therefore often written as a *membership function* $A : \mathbb{R} \to [0, 1]$, mapping an input x to its membership degree. This membership degree then determines the degree of activation of the corresponding fuzzy rule. B can again be a fuzzy set, meaning the output of the system is itself

a fuzzy set, or it can be a crisp value, in which case the output of the system is calculated as a weighted average of the outputs of the fuzzy IF-THEN rules, using the degrees of activation as weights.

In this work, we use the Adaptive-Network-Based Fuzzy Inference System (ANFIS) [7]. ANFIS encodes a number of fuzzy IF-THEN rules into a neural network architecture (see figure 1), allowing parameters to be found using backpropagation and gradient descent. We can interpret this network as a set of n rules in m input variables:

$$\begin{aligned}
&\text{IF } x_1 \text{ is } A_{11} \text{ AND } \ldots \text{ AND } x_m \text{ is } A_{m1} \\
&\qquad \text{THEN } y \text{ is } f_1(x_1, \ldots, x_m) \\
&\text{IF } x_1 \text{ is } A_{12} \text{ AND } \ldots \text{ AND } x_m \text{ is } A_{m2} \\
&\qquad \text{THEN } y \text{ is } f_2(x_1, \ldots, x_m) \\
&\qquad \ldots \\
&\text{IF } x_1 \text{ is } A_{1n} \text{ AND } \ldots \text{ AND } x_m \text{ is } A_{mn} \\
&\qquad \text{THEN } y \text{ is } f_n(x_1, \ldots, x_m)
\end{aligned}$$

Where every A_{ij} is a fuzzy set, x_i are the input variables, y is the output variable (can also be a vector in the case of multiple output variables), and every $f_i(x_1, \ldots, x_m)$ is some function of the input variables. In principle, these functions can take any form, but in practice mostly linear or constant functions are used.

The network architecture consists of five layers:

a. In the first layer, membership degrees for every input variable x_i in every fuzzy set A_{ij} are calculated. Membership functions usually take the form of Gaussian functions:

$$A_{ij}(x) = e^{-\left(\frac{x-b_{ij}}{a_{ij}}\right)^2}$$

with parameters b_{ij} and a_{ij} for every fuzzy set A_{ij}.

b. De degree of activation w_i for every rule is calculated as the product of the membership degrees of the input values in the rule's fuzzy sets.

c. The degrees of activation are normalized:

$$\overline{w}_i = \frac{w_i}{\sum_j w_j}$$

d. The output value for every rule is calculated as $\overline{w}_i f_i(x_1, \ldots, x_m)$.

e. The global output of the system is calculated as the weighted average of the output values of the rules:

$$y = \sum_i \overline{w}_i f_i(x_1, \ldots, x_m)$$

4 Approach

We now present our approach to distill a DQN policy into an ANFIS controller. The consequents of fuzzy rules are chosen to be constant vectors, as these are more easily interpretable than linear functions of the input values. The algorithm consists of three steps: pre-processing, distillation and post-processing. We describe the steps in this order.

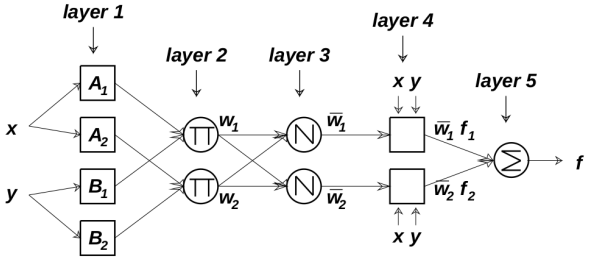


Figure 1: ANFIS with two inputs and two fuzzy rules.

4.1 Pre-processing

To initialize the parameters and amount of rules of an ANFIS controller, clustering algorithms are often used [9]. This is because such clustering algorithms are usually much less computationally intensive than the training itself, and can save a lot of computation by shortening the training phase. Also, clustering algorithms allow us to initialize ANFIS in such a way that fuzzy sets are almost disjoint, which reduces the probability of generating many similar but slightly different fuzzy sets during training, which are usually difficult to interpret.
In this work, we use subtractive clustering [15]. This algorithm takes as input a set of points $Z = \{\boldsymbol{z}_i\}$. The result is a set of center points $C = \{\boldsymbol{c}_i\} \subset Z$, with every center point defining a cluster. We can use this to initialize ANFIS as follows: We build a dataset by having the teacher interact with the environment and recording the input vectors $\boldsymbol{v}_i$ and the corresponding output vectors (Q-values) $\boldsymbol{u}_i$ produced by the teacher. These are concatenated to form a matrix M. This matrix of points is passed as input to subtractive clustering, which produces a set of center points $C = \{\boldsymbol{c}_i\} \subset M$. Every center point $\boldsymbol{c}_i$ can then be written as $(\boldsymbol{v}_{\boldsymbol{c}_i}, \boldsymbol{u}_{\boldsymbol{c}_i})$. Finally, we convert every couple $(\boldsymbol{v}_{\boldsymbol{c}_i}, \boldsymbol{u}_{\boldsymbol{c}_i})$ into a fuzzy rule: IF $\boldsymbol{x}$ IS $\boldsymbol{v}_{\boldsymbol{c}_i}$ THEN $\boldsymbol{y}$ IS $\boldsymbol{u}_{\boldsymbol{c}_i}$.

4.2 Distillation

After initialization, we distill the teacher policy into ANFIS using student trajectories. ANFIS interacts with the environment, and every state $\boldsymbol{s}_i$ that is encountered is saved into a replay memory together with the teacher's Q-values for that state $\boldsymbol{q}_i^T$. At every timestep, a random minibatch is taken from this replay memory and ANFIS is trained on this batch using the temperature KL loss (eq. 2) with $\tau = 0.1$.

4.3 Post-processing

After distilling the policy, we apply two different post-processing techniques to simplify the rule base. First, we merge similar rule consequents. A decreased amount of consequents allows us to group rules by their effect, improving the interpretability. Second, we merge similar fuzzy sets in rule antecedents. If the amount of different fuzzy sets is small enough, it is easier to give linguistic labels to fuzzy sets, again improving interpretability.

- **Merging consequents**: The similarity of consequent vectors $\boldsymbol{c}_i, \boldsymbol{c}_j$ is measured by applying the softmax-function σ and measuring cross-entropy between the resulting distributions:

  $$d(\boldsymbol{c}_i, \boldsymbol{c}_j) = -\sum_k \sigma(\boldsymbol{c}_i)_k * \log(\sigma(\boldsymbol{c}_j)_k)$$

  Couples of consequents with similarity larger than some threshold are then replaced with their average $\boldsymbol{c}^* = \frac{\boldsymbol{c}_i + \boldsymbol{c}_j}{2}$.

- **Merging fuzzy sets**: To identify similar fuzzy sets, we use the product-based Jaccard index [16]:

  $$J(A, B) = \frac{|A \cap_{T_P} B|}{|A \cap_{T_P} A| + |B \cap_{T_P} B| - |A \cap_{T_P} B|}$$

  For every input dimension, we then merge couples of fuzzy sets if they are larger than some threshold α by averaging their parameters.

5 Results

We apply the algorithm described in section 4 to the well-known cartpole environment from OpenAI Gym [10]. The results are shown in figure 2. To establish a baseline, we first train a DQN with a single layer of 128 hidden nodes and ANFIS with 30 rules using Q-learning. The neural net is able to solve the task after about 150 episodes, while ANFIS is unable to converge. Next, we use the trained DQN as a teacher and distill into a new ANFIS agent. This distilled agent is able to solve the task in only 20 episodes, while using between only 3 and 6 fuzzy rules (the experiment was replicated 50 times, hence the variation in ANFIS initialization).
Next, we apply both post-processing procedures to simplify an ANFIS rule base of 5 fuzzy rules resulting from this distillation experiment. Figure 3 shows the merging of fuzzy sets: if we merge sets with a similarity of 0.55 or more, this has little to no influence on the performance of the model. In practice, this allows us to bring the amount of fuzzy sets down from 20 (4 input variables $\times$ 5 fuzzy rules) to 9. Merging consequents is shown in figure 4. We see here that we can reduce the amount of consequents down to 2 without much influence on performance. This is the minimal reasonable amount of consequents, as there are 2 possible actions in the cartpole environment. This means we can group all the fuzzy rules into 2 groups: rules that push the cart to the left, and rules that push it to the right.

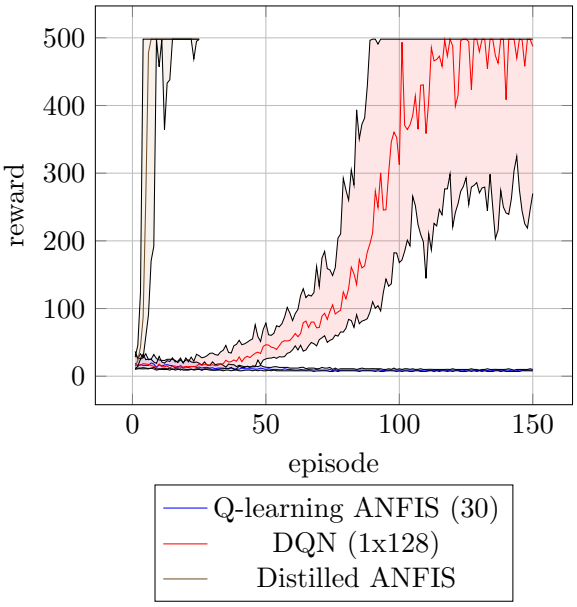


Figure 2: DQN vs. ANFIS Q-learning vs. distillation of DQN to ANFIS on the cartpole environment. The experiment was repeated 50 times, the curves show the 20th, 50th and 80th percentiles of the reward for each episode.

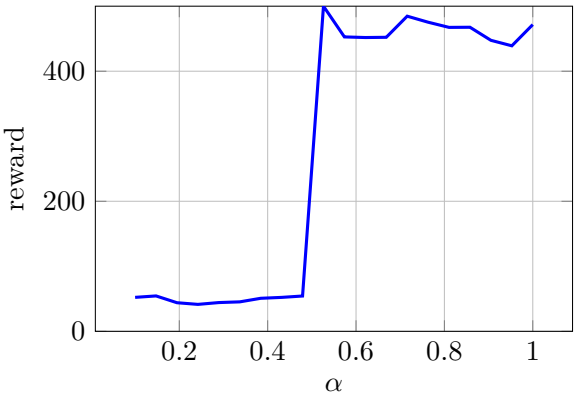


Figure 3: Merging of fuzzy sets in rule antecedents. The curve shows the achieved reward for a 5-rule ANFIS after merging fuzzy sets for varying merge threshold values α.

6 Conclusion

We have seen that ANFIS, although unable to learn a satisfying policy for the cartpole-

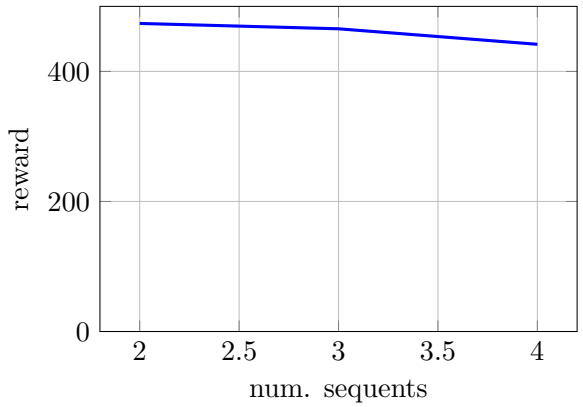


Figure 4: Merging of rule consequents. The curve shows the achieved reward for a 5-rule ANFIS for varying unique consequents.

environment through Q-learning, is able to solve the problem through distillation. It also seems possible to simplify the resulting rule base to a certain degree without significantly affecting performance. This opens new directions for interpretability research, including the application of more heuristic approaches to simplify the rule base after distillation, regularization techniques to keep the rule base interpretable during distillation, or the exploration of new, more complex fuzzy architectures.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013.

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, September 2015.

[3] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[4] John R Zech, Marcus A Badgeley, Manway Liu, Anthony B Costa, Joseph J Titano, and Eric K Oermann. Confounding variables can degrade generalization performance of radiological deep learning models. *arXiv preprint arXiv:1807.00431*, 2018.

[5] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer Publishing Company, Incorporated, 1st edition, 2017.

[6] Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy Distillation. *arXiv:1511.06295 [cs]*, November 2015.

[7] J.-S.R. Jang. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, May-June/1993.

[8] Hirofumi Miyajima, Noritaka Shigei, and Hiromi Miyajima. Approximation Capabilities of Interpretable Fuzzy Inference Systems. page 8, 2015.

[9] Rui Pedro Paiva and António Dourado. Interpretability and learning in neuro-fuzzy systems. 2004.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. 2016.

[11] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [cs, stat]*, March 2015.

[12] Simon Schmitt, Jonathan J. Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, and S. M. Ali Eslami. Kickstarting Deep Reinforcement Learning. *arXiv:1803.03835 [cs]*, March 2018.

[13] Wojciech Marian Czarnecki, Siddhant M. Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Simon Osindero, Nicolas Heess, and Razvan Pascanu. Mix&Match - Agent Curricula for Reinforcement Learning. *arXiv:1806.01780 [cs, stat]*, June 2018.

[14] L. A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, 30(3):407–428, September 1975.

[15] Stephen L Chiu. Fuzzy model identification based on cluster estimation. *Journal of Intelligent & fuzzy systems*, 2(3):267–278, 1994.

[16] M. Setnes, R. Babuska, U. Kaymak, and H. R. van Nauta Lemke. Similarity measures in fuzzy rule base simplification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(3):376–386, June 1998.

Vulgariserende samenvatting

Machine learning is de verzamelnaam voor algoritmes die kunnen leren uit (grote hoeveelheden) data. Zulke algoritmes laten ons toe om computers problemen te laten oplossen die door rechtstreekse programmering niet of zeer moeilijk haalbaar zijn. Zo is het bijvoorbeeld niet makkelijk om een aantal regels te bedenken waarop de computer zich kan baseren om te beslissen of een foto een afbeelding van een hond of van een kat bevat. In plaats daarvan, kunnen we een machine learning-algoritme een verzameling foto's geven, en daarbij voor iedere foto specifiëren welk dier ze bevat. Het algoritme kan dan een model leren uit die data, en voor nieuwe foto's voorspellen of ze een hond of een kat bevatten.

Zulke algoritmes worden niet enkel toegepast om foto's te categoriseren. We kunnen gelijkaardige principes gebruiken om een complexe taak goed te leren uitvoeren, zoals schaken, of autorijden. We doen dat door het algoritme te "belonen" voor goede acties, en te "straffen" voor slechte acties. In dat geval spreekt men van *reinforcement learning*.

Er bestaan verschillende machine learning-algoritmes met uiteenlopende eigenschappen, maar over het algemeen kunnen we een belangrijke trade-off stellen: hoe complexer het algoritme, hoe beter de resultaten, maar ook hoe minder *interpreteerbaar* het resulterende model. Dit betekent dat het voor een mens zeer moeilijk te begrijpen kan zijn *waarom* een machine learning-algoritme een bepaalde beslissing maakt. Dit kan voor problemen zorgen indien machine learning wordt toegepast op kritieke applicaties.

In deze thesis proberen we het probleem van deze trade-off te verminderen, door reinforcement learning-algoritmes te *distilleren*. Dit betekent dat we een complex, niet-interpreteerbaar

model eerst een taak laten oplossen, om vervolgens de kennis van dat model over te brengen naar een minder complex, meer interpreteerbaar model. We kunnen het complexe model hier beschouwen als een leerkracht, en het minder complexe model als een leerling. De leerling kan uit zichzelf een probleem moeilijk oplossen, maar met de hulp van een leerkracht lukt het wel. Deze aanpak laat ons toe om simpele modellen problemen te laten oplossen die ze rechtstreeks niet zouden aankunnen.

Dankwoord

Bij het schrijven van deze thesis kon ik rekenen op de hulp van verschillende mensen. Eerst en vooral wil ik mijn promotor dr. Yvan Saeys en mijn begeleider Jonathan Peck bedanken voor de vele ondersteuning en het gerichte advies tijdens het onderzoekswerk, en voor het accepteren van mijn voorstel van thesisonderwerp. Dankzij hen heb ik me dit academiejaar kunnen focussen op een onderwerp dat mij persoonlijk erg aanspreekt.

Verder wil ik ook mijn ouders bedanken voor hun steun, vertrouwen, en alle kansen die ze mij gegeven hebben en die mij hier gebracht hebben. Ook bedank ik al mijn vrienden en familie voor de vele morele steun doorheen het academiejaar. In het bijzonder wil ik Ronald Merckx bedanken voor het persoonlijke advies waar ik dit jaar op kon rekenen (*een betje verlies eje altoid*).

Toelating tot bruikleen

De auteur geeft de toelating deze masterproef voor consultatie beschikbaar te stellen en delen van de masterproef te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de bepalingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze masterproef.

Arne Gevaert 01/06/2019

Inhoudsopgave

Hoofdstuk 1

Inleiding

Neurale modellen in machine learning vinden hun oorsprong bij het perceptron [4] in de jaren 1950, en zijn dus geen recente uitvinding. Dankzij significante ontwikkelingen in de beschikbaarheid van grote datasets [5, 6] en parallelle rekenkracht in de vorm van GPU's hebben neurale modellen in het laatste decennium echter een heropleving meegemaakt. Zo hebben diepe neurale netwerken de state-of-the-art verbeterd in een groot aantal toepassingsgebieden, bijvoorbeeld in spraakherkenning [7], beeldanalyse [8, 9, 10], natuurlijke taalverwerking [11] en reinforcement learning [12].

De positieve resultaten die door diepe neurale netwerken geboekt worden, hebben ervoor gezorgd dat ze sterk aan populariteit gewonnen hebben. Neurale netwerken worden echter algemeen beschouwd als *black boxes*. Dit betekent dat het zeer moeilijk is om te verklaren hoe het model tot een beslissing komt, of om een algemene intuïtie op te bouwen over de werking van het model. Interpretabiliteit is echter een cruciale eigenschap voor veel toepassingen waar een basisvertrouwen in het model een vereiste is, zoals o.a. in medische of juridische applicaties. Recente legislatuur van de Europese Unie vereist ook een basisverklaarbaarheid van predicties gemaakt door machine learning-modellen [13]. Over het algemeen kunnen we besluiten dat technieken die toelaten om meer inzicht te verwerven in neurale modellen van groot belang zijn voor de wetenschap.

De eerste pogingen om meer inzicht te verwerven in de werking van neurale netwerken werden geïntroduceerd in de jaren 1990, in de vorm van feature selection-technieken om te achterhalen welke features het belangrijkst waren voor een multi-layer perceptron om tot een beslissing te komen [14]. Naast algemene feature selection-technieken werden ook specifieke technieken ontwikkeld die berusten op de topologie van neurale netwerken, zoals Optimal Brain Damage [15] en Optimal Brain Surgeon [16]. Ondanks deze ontwikkelingen is er vandaag nog steeds een grote nood aan technieken voor interpretatie van neurale netwerken.

Naast deze ontwikkelingen in het gebied van feature selection, werden ook in de jaren 90 de eerste ontwikkelingen gemaakt in wat nu "knowledge distillation" genoemd wordt. De oorsprong van dit werk kan gelegd worden bij Craven et al. [17] die interpreteerbare beslissingsregels extraheerden van neurale netwerken. Deze onderzoeksrichting is recent opnieuw opgenomen om een ensemble van modellen te distilleren in een enkel neuraal netwerk [1]. Over het algemeen kunnen we knowledge distillation als volgt omschrijven: een expressief, *black box*-model (de *teacher*) wordt getraind op een dataset of een taak. Vervolgens wordt een nieuw, meestal simpeler model (de *student*) getraind op de output van de teacher. De outputs van de teacher worden ook *soft targets* genoemd. Deze soft targets kunnen geïnterpreteerd worden als probabiliteitsdistributies over output labels in supervised learning, of over acties in reinforcement learning. Als deze distributies een hoge entropie bezitten, bevatten ze veel meer informatie dan de originele labels. Bijvoorbeeld, in MNIST kan een soft target beschrijven dat een afbeelding een 2 voorstelt, maar *lijkt op* een 7. Hierdoor kan de student getraind worden met minder data en met een hogere learning rate, waardoor het simpele model een betere performantie kan bereiken dan indien het op de originele data getraind zou worden.

In dit werk wordt de focus gelegd op een specifieke implementatie van knowledge distillation, namelijk **policy distillation** met behulp van **vaaginferentie**. Policy distillation is een deelgebied van knowledge distillation, waarbij het idee wordt toegepast op reinforcement learning policies [2]. Dit brengt enkele complicaties met zich mee, zoals het feit dat

de "dataset" geen vaste vorm heeft, aangezien de data waarop getraind wordt afhankelijk is van de policy zelf tijdens de training. Vaaginferentie is de verzamelnaam voor machine learning-systemen die geïnspireerd zijn op principes uit de vaaglogica [3]. Vaaglogica is een uitbreiding van de klassieke logica die toelaat om met proposities te werken die slechts gedeeltelijk voldaan zijn. Merk op dat dit verschilt van probabiliteitstheorie: in de probabiliteitstheorie betekent een probabiliteit van 0.5 bijvoorbeeld dat er 50% kans is dat een bepaalde uitspraak (volledig) waar is. In vaaglogica betekent een waarheidswaarde van 0.5 dat een bepaalde uitspraak *voor 50% voldaan is*. Vaaglogica laat onder andere toe om wiskundige modellen op te stellen die gemakkelijk interpreteerbaar en uitlegbaar zijn met behulp van natuurlijke, menselijke taal. Vaaginferentie-gebaseerde systemen vormen dus een ideale kandidaat voor policy distillation als we het primaire doel stellen op interpretabiliteit.

De rest van de inleiding brengt een introductie tot machine learning, reinforcement learning, markov decision processes, Q-learning en meer specifiek *Deep Q-Networks* (DQN) [18]. In hoofdstuk 2 passen we SVCCA [19], een bestaande techniek voor post-hoc interpretaties van neurale modellen, toe op DQN. Vervolgens wordt in hoofdstuk 3 een meer gedetailleerde literatuurstudie gemaakt over knowledge distillation-technieken, en de complicaties die we ondervinden als we dit idee uitbreiden naar reinforcement learning (policy distillation). Hoofdstuk 4 voorziet een algemene introductie tot de vaaglogica en vaaginferentie, en in hoofdstuk 5 bespreken we het ontwikkelde distillatie-algoritme en wordt gerapporteerd over de uitgevoerde experimenten en hun resultaten. Ten slotte brengen we een conclusie en een overzicht van toekomstig werk in hoofdstuk 6.

1.1 Machine learning

1.1.1 Situering en definities

Machine learning is de branche in de informatica die algoritmes bestudeert die een bepaalde taak uitvoeren zonder expliciet geprogrammeerd te worden. In plaats daarvan

maken machine learning-algoritmes gebruik van statistische modellen en een (voldoende grote) dataset, ook de *trainingdata* genoemd. Dankzij betere beschikbaarheid van krachtige hardware en grote datasets ondervinden we een explosieve groei in toepassingen van deze technieken, met succesvolle resultaten in o.a. computervisie [8, 9, 10], spraakherkenning [7], aanbevelingssystemen [20, 21], zelfrijdende voertuigen [22, 23], etc.

Onder machine learning kan men verschillende deelgebieden onderscheiden op basis van welke data beschikbaar is en de aard van de probleemstelling.

- **Supervised learning:** In supervised learning is een dataset beschikbaar waarin zowel invoerobjecten als verwachte uitvoer gegeven worden. Deze invoerobjecten kunnen van eender welke vorm zijn, zoals bv. foto's, videofragmenten, tekstfragmenten, etc. Bij ieder invoerobject is een bijhorend uitvoerlabel gegeven. Dit uitvoerlabel beschrijft welke uitvoer verwacht wordt bij die invoer. Indien het bv. de taak is om voor foto's van handgeschreven cijfers te beslissen welk cijfer ze bevatten, vormen de foto's de invoerobjecten en is elk uitvoerlabel een getal dat aangeeft welk cijfer op de foto afgebeeld staat. De bedoeling van het algoritme is dan om een model op te stellen van de relatie tussen invoer en uitvoer, zodat het uitvoerlabel van nieuwe invoerdata voorspeld kan worden. Supervised learning kan op zijn beurt opgedeeld worden in classificatie en regressie: bij classificatie is het aantal mogelijke labels beperkt en is het de bedoeling om een invoerobject in de juiste categorie te plaatsen, bv. objectherkenning in foto's [9, 10]. Bij regressie is het label een continue waarde waar een schatting van gemaakt moet worden, bv. schatting van huisprijzen [24].

- **Unsupervised learning:** In tegenstelling tot supervised learning worden bij unsupervised learning geen labels gegeven bij de trainingdata. In plaats daarvan wordt enkel een dataset van invoerwaarden gegeven. Het doel van unsupervised learning is om structuur te ontdekken in deze dataset, bv. het identificeren van clusters van gelijkaardige objecten. Deze informatie kan gebruikt worden om kennis te extraheren uit de dataset zelf, of om nieuwe objecten in deze clusters te categoriseren.

- **Reinforcement learning:** In dit deelgebied van machine learning ligt de focus op het gedrag van actoren in een omgeving. De bedoeling is om door middel van ervaring en trial-and-error een gedrag aan te leren dat een bepaalde cumulatieve beloning maximaliseert. Reinforcement learning heeft een brede verzameling van toepassingen, gaande van het verslaan van mensen in spellen [25] tot autonome voertuigen [22] en industriële procescontrole [26].

1.1.2 ML modellen en interpretabiliteit

Om met machine learning problemen aan te pakken, kunnen veel verschillende modellen gebruikt worden, elk met hun specifieke eigenschappen en voor- en nadelen. We overlopen enkele van de meest gebruikte modellen en bespreken ook hun intrinsieke interpretabiliteit.

Lineaire modellen

Lineaire modellen vormen de minst complexe modellen die in de praktijk gebruikt worden. In deze modellen wordt een hypervlak gemaakt in de invoerruimte, op basis waarvan een uitvoerwaarde berekend wordt. Een hypervlak is een veralgemening van een rechte lijn in meer dan 2 dimensies, en bestaat uit alle punten die voldoen aan de vergelijking:

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d = 0$$

Waarbij d het aantal invoerdimensies is en $[x_1, \ldots, x_d]^\intercal$ een vector in de invoerruimte. De parameters van dit model bestaan uit de coëfficienten $w_1, \ldots, w_d$ en de intercept w_0.

Lineaire modellen kunnen zowel voor regressie als voor classificatie gebruikt worden. Om een hypervlak te gebruiken voor binaire classificatie, kunnen we kijken naar het teken van de vergelijking voor een bepaalde invoerwaarde:

$$g(\boldsymbol{x}) = w_0 + w_1 x_1 + \cdots + w_d x_d$$

$$y = sgn(g(\boldsymbol{x})) = \begin{cases} +1 & \text{indien } g(\boldsymbol{x}) \geq 0 \\ -1 & \text{indien } g(\boldsymbol{x}) < 0 \end{cases}$$

Indien geclassificeerd moet worden met meerdere klassen, worden in de praktijk meestal meerdere hypervlakken gebruikt, waarbij een hypervlak het onderscheid maakt tussen één klasse en de rest van de klassen, of één klassen en een andere klasse. In het geval van regressie beschouwen we het hypervlak als een functie van invoerwaarden. De uitvoer van deze functie is dan de schatting van de echte uitvoerwaarde:

$$y = w_0 + w_1 x_1 + \cdots + w_d x_d$$

Lineaire modellen worden over het algemeen als relatief interpreteerbaar beschouwd, aangezien aan elke invoerdimensie een coëfficiënt wordt geassocieerd. Dimensies met grote coëfficiënten kunnen dan als belangrijk geïnterpreteerd worden, terwijl zeer kleine coëfficiënten wijzen op minder belangrijke dimensies die misschien uit het model gehaald kunnen worden. Merk hierbij wel op dat voor een geldige interpretatie de variabelen op eenzelfde schaal gebracht moeten worden, aangezien de coëfficiënten anders moeilijk vergelijkbaar zijn.

Beslissings- en regressiebomen

Een ander relatief simpel model is de beslissingsboom. Een beslissingsboom bestaat uit een boomstructuur waarbij iedere inwendige knoop een propositie bevat (meestal een ongelijkheid van een invoerdimensie en een scalaire waarde). Ieder blad van de boom bevat een klasse. Wanneer een nieuw datapunt geclassificeerd moet worden, wordt de boom doorlopen van de wortel naar beneden. Bij iedere inwendige knoop wordt de propositie getest, en wordt het corresponderende pad van het resultaat genomen. Dit wordt herhaald tot een blad bereikt wordt. De klasse in dit blad is dan de voorspelde klasse voor het datapunt. Figuur 1.1 geeft een voorbeeld van een beslissingsboom.

Beslissingsbomen kunnen gemakkelijk aangepast worden om regressie te implementeren. Hiervoor worden de klassen in de bladeren gewoon vervangen door continue uitvoerwaarden. De voorspelde waarde van het datapunt is dan de waarde in het blad waar het algoritme eindigt. Zo implementeert de regressie een stuksgewijs constante functie. Als

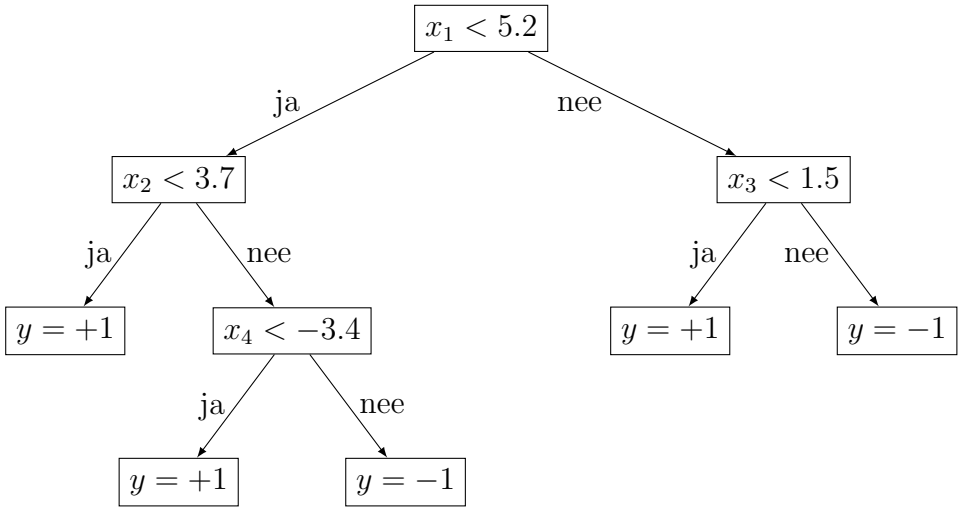
alternatief kan ook een lineair model in ieder blad geplaatst worden, in dat geval implementeert de boom een stuksgewijs lineaire functie.

Aangezien het toevoegen of verwijderen van een knoop een niet-afleidbare ingreep is, kunnen beslissingsbomen niet getraind worden met behulp van gradient descent (zie sectie 1.1.3). Voor het trainen van beslissings- en regressiebomen bestaan gespecialiseerde algoritmen [27].

Beslissings- en regressiebomen worden over het algemeen als redelijk interpreteerbaar beschouwd, op voorwaarde dat de grootte van de boom beperkt blijft. Het nadeel aan beslissingsbomen is echter dat ze instabiel zijn, d.w.z. een kleine wijziging aan invoerdata kan grote gevolgen hebben op de vorm van de boom. Om dit probleem tegen te gaan worden vaak ensembles van beslissingsbomen gebruikt [28], maar dit ten koste van de interpretabiliteit.

Deep Learning

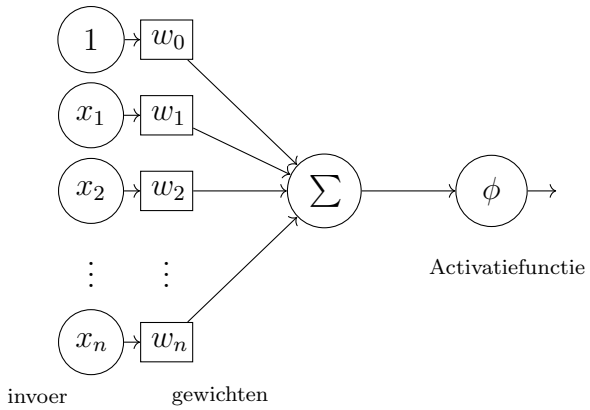
Deep learning is een meer geavanceerd model dan de vorige, en kent recent veel succes in zeer uiteenlopende gebieden. In deep learning wordt gebruik gemaakt van een diep neuraal netwerk. Aan de basis hiervan ligt het perceptron van Rosenblatt [4]: dit is een wiskundig model van een enkel neuron (zie figuur 1.2). Een perceptron neemt als invoer een vector en een bias-waarde waarvan een gewogen som wordt genomen (merk op dat we de bias-waarde ook kunnen interpreteren als een extra invoer met waarde 1 die in de gewogen som wordt opgenomen, dit wordt in de figuur weergegeven door de invoerwaarde 1). Deze gewogen som wordt geactiveerd door een niet-lineaire activatiefunctie ϕ. Enkele voorbeelden van activatiefuncties worden weergegeven in figuur 1.4. In het oorspronkelijk algoritme werd gebruik gemaakt van de Heaviside stapfunctie [29] om een onderscheid te maken tussen twee lineair afscheidbare klassen. De gewichten worden door het Perceptronalgoritme gevonden door ze iteratief aan te passen op basis van de uitkomst van het perceptron en de gewenste uitkomst voor gegeven invoerdata. Het is bewezen dat wanneer de invoervectoren uit twee



Figuur 1.1: Een simpele beslissingsboom. In een regressieboom worden de discrete labels in de bladeren vervangen door continue uitvoerwaarden.

lineair afscheidbare klassen komen, het Perceptronalgoritme convergeert en een lijn creëert die de twee klassen scheidt. Deze eigenschap wordt ook de *Perceptron convergence theorem* [30] genoemd.

Een diep neuraal netwerk wordt nu opgesteld door verschillende perceptrons te combineren in lagen (vandaar ook de benaming *multilayer perceptron*, of MLP [31]). In zijn simpelste vorm bestaat een neuraal netwerk uit een invoerlaag, een uitvoerlaag, en één of meerdere tussenlagen (Eng.: *hidden layers*). Iedere laag bestaat uit een aantal neuronen, en ieder

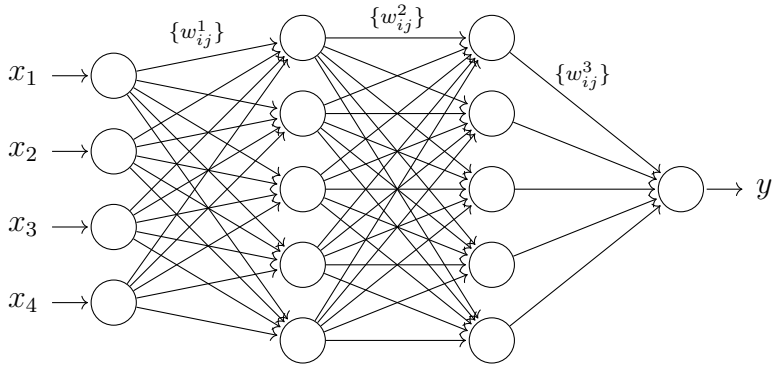


Figuur 1.2: Grafische voorstelling van het perceptron

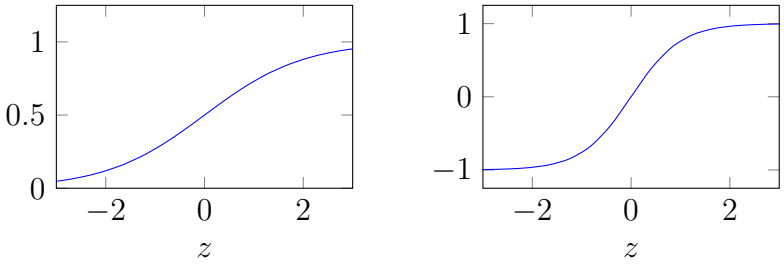
neuron in laag i is verbonden met ieder neuron in laag $i+1$. Aan iedere verbinding is een gewicht verbonden waarmee de waarde van het neuron in laag i vermenigvuldigd wordt. Alle gewogen waarden die in een neuron van laag $i+1$ toekomen worden gesommeerd en in een activatiefunctie ingevoerd. De uitvoerwaarde van deze activatiefunctie is dan de waarde van dat neuron, en deze waarde wordt vervolgens naar laag $i+2$ gestuurd. Op deze manier kan men een arbitrair aantal lagen op elkaar stapelen om de complexiteit van het model te verhogen. Figuur 1.3 toont een voorbeeld van een diep neuraal netwerk met 2 tussenlagen.

Iedere laag van een neuraal netwerk is equivalent met een matrixvermenigvuldiging gevolgd door een elementsgewijze toepassing van de activatiefunctie. De activatiefunctie zorgt ervoor dat de laag een niet-lineaire transformatie vormt op de invoerdata. Indien geen activatiefuncties gebruikt zouden worden, zou het netwerk een opeenvolging zijn van lineaire matrixvermenigvuldigingen, en dus als geheel equivalent zijn met een lineair model. Enkele veel gebruikte activatiefuncties zijn de logistische sigmoïde $\sigma(z) = 1/(1+e^{-z})$, de hyperbolische tangens $\tanh(z)$ en de Rectified Linear Unit (of ReLU) $\max(0, z)$. Deze functies worden weergegeven in figuur 1.4.

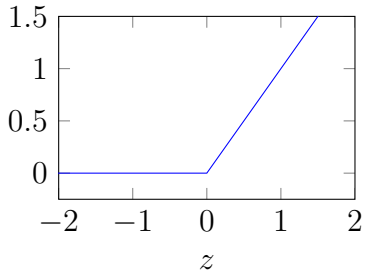
Om te berekenen in welke richting de gewichten van een neuraal netwerk aangepast moeten worden, wordt gebruik gemaakt van *backpropagation*. Hier produceert het neuraal netwerk eerst een uitvoerwaarde y voor een gegeven invoervector $\boldsymbol{x}$. Op basis van deze



Figuur 1.3: Een neuraal netwerk met 2 tussenlagen.



(a) Logistische sigmoïde. **(b)** Hyperbolische tangens.



(c) ReLU: Rectified Linear Unit.

Figuur 1.4: Veel gebruikte activatiefuncties: de logistische sigmoïde $\sigma(z)$, de hyperbolische tangens $\tanh(z)$ en de ReLU-functie $\max(0, z)$

uitvoerwaarde en een bepaalde gewenste uitvoerwaarde $\hat{y}$ wordt een fout $\mathcal{E}(y, \hat{y})$ berekend. Vervolgens wordt de fout teruggepropageerd door het neuraal netwerk om voor iedere parameter w_{ij}^l de partiële afgeleide van de fout naar die parameter $\frac{\partial \mathcal{E}(y,\hat{y})}{\partial w_{ij}^l}$ te berekenen. Deze informatie wordt dan gebruikt door een optimalisatie-algoritme om optimale gewichten voor het netwerk te vinden (zie sectie 1.1.3).

Verschillende theoretische resultaten wijzen aan dat het gebruik van "diepe" architecturen, waarbij veel verschillende hidden layers gebruikt worden, grote voordelen heeft [32]. Zo is al aangetoond dat een neuraal netwerk de invoerruimte kan opdelen in een aantal lineaire responsregio's dat exponentieel stijgt met het aantal hidden layers [33]. Hoewel diepe neurale netwerken enorm veel succes kennen indien de hoeveelheid data en rekenkracht groot genoeg is, worden ze over het algemeen beschouwd als *black box*-modellen. Het grote aantal parameters en de niet-lineaire transformaties die bij iedere laag gebeuren, zorgen er namelijk voor dat de data fundamenteel vervormd raakt, en de functie van één bepaald neuron is op zich niet duidelijk maar is meestal afhankelijk van een aantal andere neuronen. Daardoor is er veel interesse in het verklaren van de werking van zulke neurale netwerken, zodat ze ook gebruikt kunnen worden in een context waar interpretabiliteit een eis is.

1.1.3 Optimalisatie

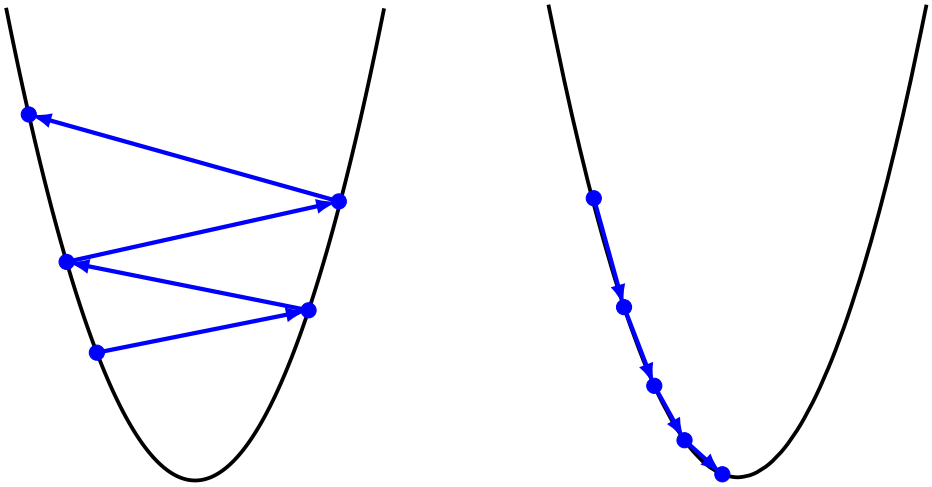
Het "trainen" van modellen op een bepaalde dataset of taak komt in de praktijk neer op het vinden van parameters voor dat model die een bepaalde kostfunctie (Eng.: *loss*) minimaliseren. Deze kostfunctie geeft weer hoe groot de fout is die door het model gemaakt wordt. Wanneer dus voor iedere parameter een waarde gevonden wordt zodat deze functie een minimum bereikt, verwachten we dat het model een kleine fout maakt en dus goed presteert. Om zo'n minimum te vinden, wordt gebruik gemaakt van optimalisatie. Hoewel wiskundige optimalisatietechnieken zoals dynamisch programmeren [34] of integer programming [35] in theorie hiervoor gebruikt zouden kunnen worden, is dit in de praktijk meestal niet haalbaar vanwege het grote aantal parameters in een model en de grote computationele kost die hiermee gepaard zou gaan. Daarom wordt in de praktijk meestal

gebruik gemaakt van een heuristische optimalisatietechniek om een optimum te benaderen. Voorbeelden hiervan zijn genetische algoritmes [36], local search, simulated annealing [37], etc. Veruit het meest gebruikte heuristische optimalisatie-algoritme in machine learning is gradient descent, of een variant daarvan.

Gradient descent kan gebruikt worden wanneer een gradiënt beschikbaar is van de kostfunctie t.o.v. de parameters van het model. Het algoritme zoekt dan iteratief een minimum van de functie door in elke stap de gradiënt van de kostfunctie te berekenen, en vervolgens de parameters een kleine aanpassing te geven in de richting van de steilste afdaling (dit is gewoon de tegenovergestelde richting van de gradiënt). De stapgrootte van de aanpassing van de parameters wordt de *learning rate* genoemd (zie figuur 1.5). Als deze hyperparameter te klein gekozen wordt, zal het algoritme zeer traag convergeren. Als ze echter te groot is lopen we het risico dat het minimum gemist kan worden. Het is dus de bedoeling om een optimale waarde van deze parameter te zoeken. Aangezien de fout van het model in het begin van de training over het algemeen het grootst is, wordt de learning rate in veel gevallen ook groot gekozen bij de start om dan in de loop van de training af te nemen. Deze techniek wordt *adaptive learning rate* genoemd [38]. De intuïtie hierachter is dat het model "grovere" stappen zet in het begin, om op het einde meer aan de details te werken.

1.1.4 Generalisatie

Wanneer een machine learning-model getraind wordt op een bepaalde dataset, is het gewoonlijk de bedoeling dat dit model later gebruikt kan worden om voorspellingen te maken op nieuwe, ongeziene data. Als het model goed presteert op een trainingdataset, betekent dit echter niet dat de prestatie op ongeziene data altijd even goed zal zijn. Dit komt doordat de trainingdata over het algemeen een ruiscomponent bevat (Eng.: noise). Deze ruis brengt irrelevante informatie in de dataset (bv. datapunten die samen een duidelijke curve volgen, hebben individueel meestal nog een kleine afwijking van die curve). Wanneer het model deze ruiscomponent mee gaat modelleren, spreekt men van *overfitting* [39] (zie figuur 1.6). Het gevolg hiervan is dat het model zeer goed presteert op de trainingdataset,



Figuur 1.5: Gradient descent. Links: te grote learning rate zorgt ervoor dat het minimum gemist wordt. Rechts: kleinere learning rate zorgt voor convergentie.

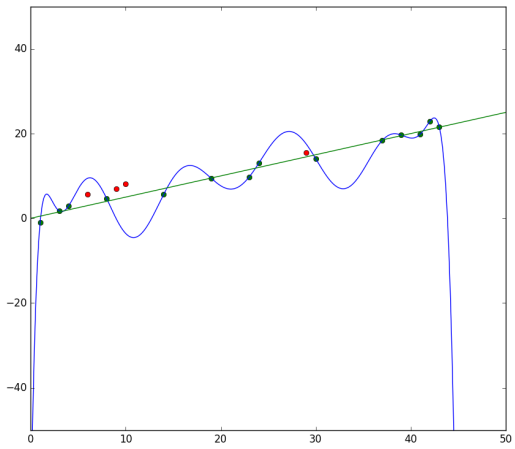
maar veel slechter op ongeziene data. Men zegt dat het model *niet generaliseert*. Om te vermijden dat dit gebeurt, bestaan regularisatietechnieken, waarvan we nu enkele van de bekendste beschouwen.

L1/L2 regularisatie

De meest gebruikte regularisatietechnieken zijn L1- en L2-regularisatie. Deze technieken breiden de kostfunctie uit met een term die grote absolute waarden van parameters penaliseert. Zo wordt vermeden dat het model zeer extreme parameterwaarden aanneemt, wat in de meeste gevallen wijst op overcomplexiteit. Bij L1-regularisatie wordt de kostfunctie $\mathcal{L}$ als volgt uitgebreid:

$$\mathcal{L}_{L1}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) + \lambda \sum_i |w_i|$$

Waarbij gesommeerd wordt over alle parameters w_i van het model. λ is een hyperparameter die bepaalt hoe zwaar de invloed van de penalisatieterm op het resultaat moet zijn. Bij L2-regularisatie wordt de kostfunctie op analoge wijze uitgebreid, waarbij gesommeerd wordt over de kwadraten van de parameters in plaats van de absolute waarden:



Figuur 1.6: Overfitting. De data volgt een duidelijk lineair patroon, maar het model neemt een veel complexere vorm aan. Dit model verklaart de trainingdata zeer goed, maar zal voor nieuwe data veel slechter presteren.

$$\mathcal{L}_{L2}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) + \lambda \sum_i w_i^2$$

L1-regularisatie heeft meer de neiging om parameters op 0 te plaatsen dan L2-regularisatie, wat zorgt voor een automatische vorm van feature-selectie. Dit kan handig zijn indien niet zeker geweten is of alle features relevante informatie bevatten. Indien dit wel het geval is, kan L2-regularisatie beter presteren.

K-fold cross-validatie

De basistechniek om in te schatten of een model goed generaliseert, is het splitsen van de dataset in een training- en een testset. Het model wordt dan enkel getraind op de trainingset en de performantie op de testset wordt gemeten. Aangezien de testset ongeziene data bevat, levert dit een schatting op van de generalisatie van het model.

K-fold cross-validatie is een techniek die toelaat om een betere schatting te maken van deze generalisatie [40]. De dataset wordt dan, voor een gegeven parameter k, gesplitst in k

delen. Het algoritme bestaat uit k iteraties, waarbij telkens één van de k delen als testset geselecteerd wordt en alle andere delen als trainingset. Bij iedere iteratie wordt het model dan opnieuw getraind. Zo bekomt men k verschillende schattingen voor de generalisatie, die als geheel een betere schatter vormen. Zo kan men beter inschatten of er sprake is van overfitting, wat toelaat om hyperparameters af te stellen om overfitting te vermijden.

1.2 Reinforcement Learning

In veel toepassingsgebieden zoals robotica, resource management en aanbevelingssystemen, wordt men geconfronteerd met problemen waarbij complexe, instabiele systemen nauwkeurig gecontroleerd moeten worden. Hierbij moet een controller (of *agent*) geprogrammeerd worden die de toestand van een omgeving (of *environment*) observeert en vervolgens een bepaald controlesignaal produceert om de environment naar een volgende toestand te sturen. Het direct programmeren van zo'n agent is in veel gevallen zeer moeilijk of zelfs onhaalbaar, meestal om één van twee redenen: (i): het systeem is zeer niet-lineair of bevat veel ruis, waardoor het opstellen van een accuraat wiskundig model onhaalbaar is, of (ii): de taak is zo complex dat een precieze definitie van "goed gedrag" niet gegeven kan worden.

Bij reinforcement learning probeert men deze problemen op een andere manier aan te pakken. Hierbij wordt een agent niet rechtstreeks geprogrammeerd, maar is het de bedoeling om door trial-and-error te *leren* welke acties in welke toestand wenselijk zijn. Dit gebeurt door het introduceren van een *reward*-signaal: een (meestal scalaire) grootheid die de agent ontvangt van de environment, en die moet weergeven hoe goed de huidige toestand van het systeem is. Zo kan een agent een bepaald gedrag aanleren zonder dat het gewenste gedrag of het gedrag van de environment gemodelleerd moeten worden.

In dit hoofdstuk geven we een introductie tot het algemene gebied van reinforcement learning [41]. Sectie 1.2.1 introduceert het Markov decision process, het wiskundig model dat aan de grondslag van reinforcement learning ligt. Vervolgens worden technieken

op basis van dynamisch programmeren behandeld in sectie 1.2.2. In sectie 1.2.3 wordt Q-learning geïntroduceerd, een techniek die kan worden aangehaald wanneer het systeem te groot wordt om te modelleren met dynamisch programmeren. Ten slotte bespreken we DQN [18] in sectie 1.2.4, een techniek die deep learning combineert met Q-learning en die in de laatste jaren verbluffende resultaten heeft teweeggebracht.

1.2.1 Markov Decision Processes

De hierboven beschreven ideeën over agent, omgeving, actie en reward kunnen wiskundig samengevat worden in een Markov decision process. MDP's vormen de wiskundige basis van reinforcement learning. Ze encapsuleren alle basisideeën van sequentiële beslissingsvorming maar zijn tegelijk nog abstract genoeg om precieze wiskundige stellingen over te formuleren. Ze vormen dus het ideale raamwerk voor reinforcement learning, en zowat alle reinforcement learning-algoritmen kunnen dan ook beschouwd worden als algoritmische oplossingen voor MDP's. In deze sectie brengen we de nodige definities voor MDP's en enkele fundamentele stellingen en technieken om MDP's op te lossen.

Definities

Een Markov decision process (MDP) bestaat uit twee hoofdonderdelen: de agent en de environment. De agent is de entiteit die beslissingen neemt, en moet dus aanleren welke beslissingen optimaal zijn op een bepaald moment. De environment omvat alles buiten de agent. Het proces bestaat uit een constante interactie tussen de agent en de environment: de environment presenteert een bepaalde toestand, de agent reageert hierop met een actie, en de environment reageert terug met een nieuwe toestand en een bijhorend rewardsignaal. Het is de taak van de agent om dit rewardsignaal te maximalizeren over tijd.

Meer formeel kunnen we een MDP definiëren vertrekkende van een toestandsverzameling $\mathcal{S}$: dit is de verzameling van alle mogelijke toestanden die het systeem kan aannemen. Voor iedere toestand $s \in \mathcal{S}$ definiëren we een actieverzameling $\mathcal{A}(s)$ van alle mogelijke

acties die de agent kan uitvoeren wanneer de environment zich in de toestand s bevindt. Indien deze verzamelingen voor alle toestanden gelijk zijn, spreken we simpelweg van de actieverzameling $\mathcal{A}$.

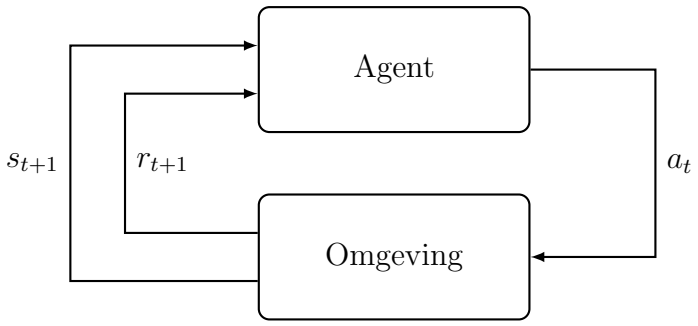
De agent en environment interageren met elkaar in een reeks van discrete tijdsstappen $t = 0, 1, 2, ...$ (merk op dat ook een continue definitie van tijd gehanteerd kan worden, maar deze technieken worden hier buiten beschouwing gelaten). In iedere stap t ontvangt de agent een toestand $s_t \in \mathcal{S}$ en kiest een actie $a_t \in \mathcal{A}$. In de volgende stap reageert de environment terug met een toestand $s_{t+1} \in \mathcal{S}$ en een beloning $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$. Op die manier genereren de agent en environment een reeks van toestanden, acties en beloningen $s_0, a_0, s_1, r_1, a_1, s_2, r_2, a_2, ...$ Dit proces wordt weergegeven in figuur 1.7.

In dit onderzoek werken we enkel met *eindige* MDPs, wat betekent dat de verzamelingen $\mathcal{S}$, $\mathcal{A}$ en $\mathcal{R}$ telkens eindig zijn. De rewards r_t en toestanden s_t worden dan gekarakteriseerd door probabiliteitsdistributies die enkel afhangen van de vorige toestand en actie:

$$p(s_{t+1}, r_{t+1}|s_t, a_t) = p(s_{t+1}, r_{t+1}|s_0, a_0, s_1, a_1, \ldots s_t, a_t) \tag{1.1}$$

$$\sum_{s_{t+1}\in\mathcal{S}} \sum_{r_{t+1}\in\mathcal{R}} p(s_{t+1}, r_{t+1}|s_t, a_t) = 1, \forall s_t \in \mathcal{S}, a_t \in \mathcal{A} \tag{1.2}$$

Waarbij $p(s_{t+1}, r_{t+1} \mid s_t, a_t)$ de kans is om reward r_{t+1} te ontvangen en naar toestand s_{t+1} te gaan, indien vanuit toestand s_t actie a_t gekozen wordt. Deze functie wordt ook de *toestandstransitiefunctie* genoemd. Formule 1.1 duidt aan dat de distributies van volgende toestanden en beloningen *enkel* afhankelijk zijn van de vorige toestand en actie, en dus



Figuur 1.7: Interactie tussen agent en omgeving in een Markov Decision Process.

volledig door die twee variabelen bepaald worden. Deze eigenschap noemt men ook de *Markov*-eigenschap, wat aanleiding geeft tot de naam *Markov* decision process.

Return

Het doel van de agent is om de beloningen die hij verkrijgt van de environment te maximalizeren. Om dit idee formeel te karakteriseren, definiëren we een nieuw begrip: de *expected return*. Noteren we de sequentie van rewards verkregen vanaf tijdsstap t als $r_{t+1}, r_{t+2}, ...,$ dan wordt de expected return gedefiniëerd als volgt:

$$g_t = r_{t+1} + r_{t+2} + \cdots + r_T \tag{1.3}$$

Waarbij T de laatste tijdsstap aanduidt. Hierbij veronderstellen we dat er zo'n laatste tijdsstap bestaat, m.a.w. we veronderstellen dat de interactie tussen de agent en de environment opgedeeld kan worden in eindige sequenties. Deze sequenties noemen we *episodes*. In dit geval is iedere episode compleet onafhankelijk van iedere andere episode. Iedere episode begint in een bepaalde starttoestand, die vast kan zijn of gekozen kan worden uit een bepaalde distributie van starttoestanden, onafhankelijk van de uitkomst of het verloop van vorige episodes.

Niet iedere taak kan op zo'n manier in episodes opgedeeld worden, zoals bijvoorbeeld de continue controle van de temperatuur in een datacenter. In dit geval moeten we de definitie van expected return aanpassen. In dit geval zou de eindtoestand namelijk $T = \infty$ zijn, waardoor de expected return in veel gevallen ook oneindig zou zijn. Om deze problematiek te vermijden, voeren we het concept *discounting* in. Discounting zorgt ervoor dat de agent meer belang hecht aan rewards die in de nabije toekomst verwacht worden. Formeel wordt nu de expected *discounted* return gemaximaliseerd:

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1.4}$$

Waarbij γ een parameter is tussen 0 en 1, ook wel de *discount factor* genoemd. Deze factor bepaalt de afweging die de agent maakt tussen dichte en verre rewards: indien $\gamma = 0$, zal

de agent enkel de eerstvolgende reward proberen te maximaliseren. Naarmate γ nadert naar 1, houdt de agent meer rekening met rewards die pas later verwacht zullen worden. Indien $\gamma = 1$, verkrijgen we gewoon opnieuw definitie 1.3 en wordt dus de volledige som van rewards gemaximaliseerd.

Merk op dat, hoewel definitie 1.4 een oneindige som is, deze waarde toch eindig zal zijn indien de reward-sequentie begrensd blijft en $\gamma < 1$. In de praktijk zullen we deze veronderstellingen ook altijd maken.

Een belangrijke eigenschap van de expected discounted return is de relatie tussen opeenvolgende tijdsstappen:

$$\begin{aligned} g_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \dots) \\ &= r_{t+1} + \gamma g_{t+1} \end{aligned}$$

Deze gelijkheid ligt aan de grondslag van veel RL-algoritmes.

Policies en Value-functies

Het gedrag dat door een agent wordt uitgevoerd, kunnen we formeel beschrijven als een *policy*. Een policy is een afbeelding van iedere toestand naar een kansverdeling die voor iedere actie beschrijft wat de kans is dat die actie genomen zal worden. We noteren de policy als $\pi(a|s)$, de kans dat actie a genomen wordt als de huidige toestand s is. Om een policy te vinden die de expected return maximaliseert, zal het nuttig blijken om bepaalde *value*-functies te benaderen. Deze functies beschrijven hoe "goed" het is om in een bepaalde toestand terecht te komen, of om een bepaalde actie uit te voeren. Formeel definiëren we de volgende twee functies:

$$v_\pi(s) = \mathbb{E}_\pi[g_t|s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right] \tag{1.5}$$

$$q_\pi(s,a) = \mathbb{E}_\pi[g_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right] \tag{1.6}$$

Vergelijking 1.5 beschrijft de expected return als de agent start in toestand s en daarna de policy π volgt. Deze functie wordt ook de *state-value functie* van π genoemd. Vergelijking 1.6 is zeer gelijkaardig, en beschrijft de expected return als de agent start in toestand s, daar de actie a kiest, en vervolgens de policy π volgt. Deze functie noemen we de *action-value functie* van π.

Net als de expected discounted return vertonen ook de state-value en action-value functies recursieve relaties tussen opeenvolgende tijdsstappen. Voor de state-value functie kunnen we deze relatie als volgt afleiden:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[g_t|s_t = s] \\ &= \mathbb{E}_\pi[r_t + \gamma g_{t+1}|s_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma \mathbb{E}_\pi[g_{t+1}|s_{t+1} = s']\big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right], \forall s \in S \end{aligned} \tag{1.7}$$

Hier staan a, s' en r onder de sommatie resp. voor de mogelijke acties, volgende toestanden en verkregen rewards. Op analoge wijze kunnen we afleiden dat de volgende relatie geldt voor de action-value functie:

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)\left[r + \gamma \sum_{a'} \pi(s', a') q_\pi(s', a')\right] \tag{1.8}$$

Vergelijkingen 1.7 en 1.8 worden ook de *Bellman*-vergelijkingen genoemd van resp. de state-value functie en de action-value functie.

Voor eindige MDP's kunnen we nu een optimale policy definiëren met behulp van de state-value en action-value functies. Met behulp van de state-value functies vormen we een partiële ordening over policies, waarbij de ordeningsrelatie gedefiniëerd wordt als volgt:

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s), \forall s \in \mathcal{S}$$

Men kan aantonen [41] dat er altijd een policy π_* bestaat die groter is of gelijk aan alle policies. Deze policy is niet altijd uniek, maar alle optimale policies delen wel altijd dezelfde state- en action-value functies v_* en q_*. Daarom duiden we alle optimale policies aan met π_*. De optimale state- en action-value functies zijn gedefiniëerd als volgt:

$$v_*(s) = \max_\pi v_\pi(s)$$
$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

Ook de optimale state- en action-value functies voldoen aan de Bellman-vergelijkingen 1.7 en 1.8. Omdat deze functies echter optimaal zijn, kunnen de Bellman-vergelijkingen uitgedrukt worden onafhankelijk van een specifieke policy. Deze vergelijkingen zijn de *Bellman-optimaliteitsvergelijkingen*:

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right] \tag{1.9}$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right] \tag{1.10}$$

Deze vergelijkingen vormen niet-lineaire stelsels van vergelijkingen, en als de toestandstransitiefunctie van het achterliggende MDP gekend is, kunnen deze vergelijkingen opgelost worden met klassieke methoden voor het oplossen van niet-lineaire stelsels. In de meeste praktische toepassingen zijn deze methoden echter niet computationeel haalbaar, wat aanleiding geeft tot de verschillende RL-algoritmen die in de volgende hoofdstukken besproken worden.

1.2.2 Dynamisch programmeren

Indien de werking van een gegeven eindig MDP perfect gekend is, d.w.z. indien de verzamelingen $\mathcal{S}$, $\mathcal{A}$ en $\mathcal{R}$, en de toestandstransitiefunctie p volledig gegeven zijn, kan dynamisch programmeren toegepast worden om een oplossing te vinden in de vorm van een optimale policy. In de praktijk zijn methoden op basis van dynamisch programmeren meestal niet

bruikbaar, omdat het MDP niet perfect gekend is of omdat het te groot is en de computationele kost voor dynamisch programmeren te groot wordt. Desondanks zijn DP-algoritmen van zeer groot theoretisch belang in reinforcement learning.

Het idee van DP-algoritmes voor RL is het benaderen van de value-functies die in sectie 1.2.1 besproken worden. Indien een optimale value-functie (hetzij v_* of q_*) gekend is, kunnen we op triviale wijze de verwachte value maximalizeren en zo een optimale policy verkrijgen. Door gebruik te maken van de Bellman-vergelijkingen kunnen we de value-functies benaderen.

Policy evaluation

Policy evaluation is een algoritme dat gebruikt kan worden om de state-value functie v_π te berekenen voor een gegeven policy π. Het kan samengevat worden als het herhaaldelijk toepassen van de Bellman-vergelijking 1.7 voor iedere toestand $s \in \mathcal{S}$. Zo wordt een sequentie van value-functies gevormd waarvoor v_π een vast punt is. Men kan aantonen dat deze sequentie ook werkelijk convergeert naar de correcte v_π.

Algoritme 1 Policy Evaluation

Input π (policy), θ

Output $V(s) \approx v_\pi(s)$

Initialiseer $V(s)$ arbitrair, behalve $V(s) = 0$ voor alle terminale toestanden s.
repeat
    $\Delta \leftarrow 0$
    for all $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max \Delta, |v - V(s)|$
    end for
until $\Delta < \theta$

Value iteration

Het Policy Evaluation-algoritme kan met een kleine aanpassing omgezet worden om een optimale policy en bijhorende value-functie te berekenen, in plaats van de value-functie van een gegeven policy. Het idee achter value iteration is simpel: indien we weten hoe goed het is om de huidige policy te volgen vanuit een bepaalde toestand, kunnen we voor iedere actie berekenen hoe goed het is om die actie uit te voeren en daarna de policy te volgen. Indien we een actie vinden waarvoor deze waarde beter is dan de waarde van de policy zelf, kunnen we de policy hier beter aanpassen.

Formeel kunnen we dit idee als volgt formuleren. Indien we in toestand s eerst actie a kiezen en vervolgens de policy π volgen, wordt de value van dit gedrag gegeven door:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}[r_{t+1} + \gamma v_\pi(s_{t+1}) | s_t = s, a_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

Als deze waarde groter is dan v_π kan men aantonen dat de policy die in toestand s actie a selecteert en in alle andere toestanden de policy π volgt, beter is dan π zelf. Dit resultaat is een gevolg van het policy improvement theorem: indien π en π' twee deterministische policies zijn, zodat voor alle $s \in \mathcal{S}$, $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, dan geldt $v_{\pi'}(s) \geq v_\pi(s), \forall s \in \mathcal{S}$.

Door gebruik te maken van dit idee, kunnen we een gegeven (arbitraire) policy iteratief verbeteren, door telkens de maximale waarde te kiezen voor de value-functie in plaats van de waarde die bereikt wordt onder de gegeven policy. Dit resulteert opnieuw in een sequentie van value-functies die ditmaal convergeert naar de optimale value-functie v_*, waaruit we direct de optimale policy kunnen afleiden.

Merk op dat we voor één iteratie van Policy Evaluation of Value Iteration de volledige toestandsruimte overlopen, en iedere mogelijke actie in iedere toestand beschouwen. Dit

betekent dat de computationele kost van een iteratie voor beide algoritmes $\mathcal{O}(|\ \mathcal{S} \times \mathcal{A}\ |)$ is. Voor de meeste praktische toepassingen is de toestandsruimte veel te groot om volledig af te lopen bij iedere iteratie. Daarom wordt dynamisch programmeren in de praktijk niet veel gebruikt, maar is het vooral waardevol voor theoretische toepassingen en afleidingen. In de praktijk wordt meer gebruik gemaakt van algoritmes zoals Q-learning.

Algoritme 2 Value Iteration

Input θ

Output $\pi \approx \pi_*(s)$

Initialiseer $V(s)$ arbitrair, behalve $V(s) = 0$ voor alle terminale toestanden s.
repeat
    $\Delta \leftarrow 0$
    for all $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max \Delta, |v - V(s)|$
    end for
until $\Delta < \theta$
Output policy π zodat $\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

1.2.3 Q-learning

Alle vorige algoritmes gaan uit van de kritieke veronderstelling dat een volledig model van de environment gekend is in de vorm van een MDP. In de meeste praktische toepassingen is dit niet het geval, omdat de transitieprobabiliteiten van het MDP niet exact gekend zijn, of omdat het MDP simpelweg te groot is om volledig op te slaan. In deze gevallen moeten we berusten op *model-free* algoritmes: algoritmes die werken zonder een volledig model van het MDP. Eén van de bekendste model-free reinforcement learning algoritmes is Q-learning [42], gebaseerd op het idee van *temporal difference* [43].

Temporal difference

Temporal-difference learning is een techniek die toelaat om de state-value functie van een policy in een MDP te schatten zonder dat het volledige gedrag van het MDP gekend is. Het idee is als volgt: de agent start met een arbitraire value-functie en volgt de policy doorheen het MDP. Na iedere actie die de agent neemt, observeert de agent de overgang van toestand s naar s' met een reward r. Met behulp van de Bellman-vergelijking voor state-value functies kan de agent dan een *target* afleiden, een nieuwe schatting voor de state-value functie in s. Uit de afleiding van 1.7 weten we namelijk dat:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[r_{t+1} + \gamma g_{t+1} | s_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1}) | s_t = s] \end{aligned} \tag{1.11}$$

Als we in vergelijking 1.11 de (ongekende) exacte waarde $v_\pi(s)$ substitueren door de huidige schatting $V(s)$, bekomen we een nieuwe schatting uit de vorige schatting. We kunnen dus de huidige schatting voor $V(s)$ in deze richting veranderen, wat resulteert in de update regel:

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{1.12}$$

Waarbij α de *learning rate* wordt genoemd. Deze learning rate bepaalt de mate waarin de nieuwe schatting voor $V(s)$ verwerkt wordt in de waarde die we opslaan. De waarde tussen de vierkante haakjes in vergelijking 1.12 is het verschil tussen de oude waarde voor $V(s)$ en de nieuwe, bij veronderstelling betere schatting voor $V(s)$: $r + \gamma V(s')$. Dit verschil wordt de *TD error* genoemd.

Q-learning

Hetzelfde idee van temporal difference kan gebruikt worden om de optimale action-value functie $q_*(s, a)$ te benaderen in plaats van de state-value functie van een bepaalde policy.

Dit is wat Q-learning doet: op analoge wijze als voor TD-learning kunnen we vanuit de Bellman optimaliteitsvergelijkingen een target afleiden voor de schatting van de waarde van de optimale action-value functie voor een gegeven toestand s en actie a. De update-regel ziet er dan als volgt uit:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Naast deze update-regel wordt in Q-learning ook vaak gebruik gemaakt van een ϵ-*greedy* policy: de actie met hoogste Q-waarde wordt met probabiliteit $1-\epsilon$ gekozen, en een willekeurige actie met probabiliteit ϵ. Dit laat het algoritme toe om te exploreren en zo nieuwe strategieën te ontdekken. Het volledige algoritme staat in pseudocode in Algoritme 3.

Algoritme 3 Q-learning

Input α, ϵ

Output $Q(s,a) \approx q_*(s,a)$

Initialiseer $Q(s,a)$ arbitrair, behalve $Q(s,\cdot) = 0$ voor alle terminale toestanden s.
for all episodes **do**
    Initialiseer toestand s
    while episode niet geëindigd **do**
        Kies actie a uit ϵ-greedy policy afgeleid uit Q.
        Voer actie a uit en observeer r, s'.
        $Q(s,a) \leftarrow Q(s,a) + \alpha[R_{t+1} + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
    end while
end for

1.2.4 DQN

Hoewel Q-learning toelaat om een goede policy te leren in een MDP dat niet volledig gekend is, is de standaard-implementatie voor de meeste praktische toepassingen computationeel onhaalbaar. Dit komt doordat het algoritme zeer geheugen-intensief is: voor iedere mogelijke toestand van het MDP en iedere actie moet een waarde bijgehouden worden. In de praktijk is het aantal toestanden meestal veel te groot om volledig op te slaan in het geheugen. Om die reden wordt in reinforcement learning vaak gebruik gemaakt van functiebenaderingsalgoritmen [44, 18]. In DQN wordt gebruik gemaakt van een diep neuraal netwerk om de Q-functie te benaderen.

Q-netwerk

Het idee van DQN is om een diep neuraal netwerk (in deze context ook *Q-netwerk* genoemd) te trainen zodat het de optimale Q-functie benadert: $Q(s, a; \theta) \approx Q^*(s, a)$. Hier stelt θ de parameters van het Q-netwerk voor die geoptimaliseerd moeten worden. Om het netwerk te kunnen trainen, moet een kostfunctie gedefinieerd worden die dan met gradient descent geminimaliseerd moet worden. Hiervoor kunnen we gebruik maken van de Bellman-optimaliteitsvergelijking 1.10. De kostfunctie die we verkrijgen is als volgt:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s, a; \theta_i))^2\right]$$

Hier is i de iteratie, ρ een probabiliteitsdistributie over toestanden en acties die het gedrag van de agent beschrijft, en y_i de target voor de i-de iteratie:

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a]$$

Met $\mathcal{E}$ het MDP. De notaties $s, a \sim \rho(\cdot)$ en $s' \sim \mathcal{E}$ duiden aan dat s, a resp. s' genomen worden volgens resp. de probabiliteitsdistributie ρ die het gedrag van de agent beschrijft en de toestandstransitiefunctie van het MDP $\mathcal{E}$. Merk op dat de parameters uit de vorige iteratie θ_{i-1} vastgehouden worden om de kostfunctie $L_i(\theta_i)$ te optimaliseren. Dit omdat

experimenten aantonen dat het trainen zo stabieler verloopt. Deze kostfunctie afleiden levert de volgende gradiënt:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)\right] \quad (1.13)$$

Experience replay

Een tweede modificatie die bij DQN gemaakt wordt ten opzichte van puur Q-learning is het gebruik van *experience replay*. Hierbij worden de ervaringen van de agent op tijdstip t, $e_t = (s_t, a_t, r_t, s_{t+1})$ opgeslagen in een dataset $D = e_1, \ldots, e_N$. Deze dataset wordt het *replay memory* genoemd. De Q-learning updates worden dan berekend op basis van samples $e \sim D$ die willekeurig uit D getrokken worden. Het volledige algoritme wordt gegeven in Algoritme 4.

Algoritme 4 DQN

Initialiseer replay memory D met grootte N, en Q-netwerk Q met willekeurige parameters.
for all episodes **do**
    Initialiseer toestand s
    while episode niet geëindigd **do**
        Kies actie a uit ϵ-greedy policy afgeleid uit Q
        Voer actie a uit en observeer r, s'
        Sla transitie (s, a, r, s') op in D
        $s \leftarrow s'$
        Trek een willekeurige minibatch van transities (s_j, a_j, r_j, s_{j+1}) uit D
        Definieer

$$y_j = \begin{cases} r_j & \text{indien } s_{j+1} \text{ terminaal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{indien } s_{j+1} \text{ non-terminaal} \end{cases}$$

        Voer een gradient descent-stap uit volgens de gradiënt in vergelijking 1.13.
    end while
end for

Hoofdstuk 2

Toepassing van SVCCA op DQN

Een mogelijke manier om inzicht te krijgen in de werking van neurale netwerken is door verschillende netwerken die getraind zijn op dezelfde data onderling proberen te vergelijken. Meer specifiek willen we weten of verschillende neurale netwerken gelijkaardige *representaties* hebben aangeleerd. Aangezien de aangeleerde representaties van neurale netwerken over het algemeen verspreid zijn over verschillende neuronen en zelfs verschillende lagen [31], is dit geen triviale taak. SVCCA [19] is een techniek die toegepast kan worden om dit probleem op te lossen.

2.1 SVCCA

Het centrale idee van SVCCA is dat ieder neuron beschouwd wordt als een *activatievector*. Ieder neuron produceert op iedere invoer een bepaalde scalaire uitvoerwaarde. Indien voor iedere mogelijke invoerwaarde die scalaire uitvoerwaarde gekend zou zijn, zou dit het neuron volledig karakteriseren. Zo'n karakterisatie is in de praktijk echter niet berekenbaar en ook niet erg nuttig: we zijn enkel geïnteresseerd in het gedrag van een neuron op een bepaalde dataset, niet op willekeurige ruis. Daarom wordt de activatievector opgebouwd door alle activaties van een neuron op een bepaalde trainings- of validatiedataset te verzamelen.

Concreet wordt de activatievector van een neuron als volgt gedefiniëerd: gegeven een da-

taset $X = \{x_1, \ldots, x_m\}$, definiëren we de activatievector van het i-de neuron in laag l, $\boldsymbol{z}_i^l$, als de vector van uitvoerwaarden op X:

$$\boldsymbol{z}_i^l = (z_i^l(x_1), \ldots, z_i^l(x_m))$$

Waarbij $z_i^l(x)$ de uitvoer weergeeft van het i-de neuron in laag l op invoersample x. Merk op dat dit de activaties zijn van *één enkel neuron* op *de volledige dataset*, in contrast met de vaak beschouwde uitvoer van een volledige laag op één enkel invoersample. Een neuron beschouwen we dus als een vector in $\mathbb{R}^m$. We kunnen nu een laag met n neuronen van een neuraal netwerk beschouwen als de verzameling van activatievectoren van de neuronen die de laag opmaken. Deze verzameling spant een deelruimte op van $\mathbb{R}^m$. Met deze definities van neuronen als vectoren en lagen als deelruimtes, kunnen we SVCCA uitvoeren als volgt:

1. **Invoer:** de invoer van SVCCA bestaat uit twee verzamelingen van neuronen $l_1 = \{\boldsymbol{z}_1^{l_1}, \ldots, \boldsymbol{z}_{n_1}^{l_1}\}$ en $l_2 = \{\boldsymbol{z}_1^{l_2}, \ldots, \boldsymbol{z}_{n_2}^{l_2}\}$. Merk op dat de hoeveelheid neuronen in beide lagen n_1, n_2 niet gelijk hoeven te zijn, wat vergelijkingen tussen verschillende lagen toelaat.

2. **SVD:** op beide deelruimtes l_1 en l_2 wordt een singuliere waardenontbinding (Eng: Singular Value Decomposition) uitgevoerd om de deelruimtes $l_1' \subset l_1$ en $l_2' \subset l_2$ te bekomen. Hiervoor worden genoeg singuliere vectoren gekozen om 99% van de variantie in de deelruimtes te verklaren. Op deze manier worden vectoren met zeer lage variantie uitgesloten, omdat deze vectoren vooral ruis geven in neurale netwerken.

3. **CCA:** Op de deelruimtes $l_1' = \{\boldsymbol{z}_1'^{l_1}, \ldots, \boldsymbol{z}_{n_1'}'^{l_1}\}$ en $l_2' = \{\boldsymbol{z}_1'^{l_2}, \ldots, \boldsymbol{z}_{n_2'}'^{l_2}\}$ wordt een canonische correlatieanalyse [45] uitgevoerd (Eng: Canonical Correlation Analysis). CCA voert een lineaire transformatie $\tilde{l_1} = W_X l_1', \tilde{l_2} = W_X l_2'$ uit op beide deelruimtes zodat de correlaties $corrs = \{\rho_1, \ldots, \rho_{\min(n_1', n_2')}\}$ gemaximaliseerd worden [45].

4. **Uitvoer:** De uitvoer van SVCCA bestaat uit koppels van richtingen $(\tilde{\boldsymbol{z}_i}^{l_1}, \tilde{\boldsymbol{z}_i}^{l_2})$ en hun correlatiecoëfficiënt, ρ_i. Deze richtingen zijn lineaire combinaties van de oorspronkelijke neuronen: $\tilde{\boldsymbol{z}}_j^{l_i} = \sum_{r=1}^{m} \alpha_{jr}^{(l_i)} \boldsymbol{z}_r^{l_i}$.

Om de representaties van twee verzamelingen van neuronen (bv. corresponderende lagen in verschillende neurale netwerken, of dezelfde laag in hetzelfde neurale netwerk op verschillende momenten tijdens training) te vergelijken, kunnen we nu kijken naar de hoogste k correlatiecoëfficiënten $\rho_1, \ldots, \rho_k$. Als deze significant hoger zijn dan de hoogste k correlatiecoëfficiënten voor twee onafhankelijke normaal verdeelde veranderlijken, kunnen we concluderen dat de neurale netwerken gelijkaardige representaties aangeleerd hebben. Deze techniek wordt gedemonstreerd in Jupyter Notebooks op de GitHub repository[1] van SVCCA (zie figuur 2.1a, genomen uit de repository). Hier worden twee gewone neurale netwerken vergeleken met 3 hidden layers van 500 nodes die getraind werden op MNIST. We zien dat de correlatiecoëfficiënten significant hoger zijn dan voor onafhankelijke data, wat erop wijst dat de twee neurale netwerken gelijkaardige representaties hebben aangeleerd.

Een tweede opvallende eigenschap van SVCCA is het feit dat de richtingen die geproduceerd worden door het algoritme significant belangrijker zijn dan de neuronen zelf. Het experiment dat deze eigenschap aantoont wordt weergegeven in figuur 2.1b. Voor dit experiment werden twee verschillende representaties l_1, l_2 voor eenzelfde laag l met n neuronen gevormd door twee verschillende initialisaties van hetzelfde convolutionele neurale netwerk te trainen op CIFAR-10. Daarna werd SVCCA toegepast op l_1 en l_2 om richtingen $\{\tilde{\boldsymbol{z}}_1^{l_1}, \ldots, \tilde{\boldsymbol{z}}_n^{l_1}\}$ en $\{\tilde{\boldsymbol{z}}_1^{l_2}, \ldots, \tilde{\boldsymbol{z}}_n^{l_2}\}$ te bekomen. Vervolgens werd de uitvoer van laag l_i geprojecteerd op de deelruimte $span(\tilde{\boldsymbol{z}}_1^{l_i}, \ldots, \tilde{\boldsymbol{z}}_k^{l_i})$, opgespannen door de richtingen met de k hoogste correlatiecoëfficiënten, voor verschillende waarden van $k < n$.

Op figuur 2.1b kunnen we zien dat het netwerk goed presteert voor een zeer klein aantal richtingen. Ter vergelijking werd hetzelfde experiment uitgevoerd waarbij geprojecteerd werd op willekeurige neuronen en op de k neuronen met grootste activaties. In beide gevallen zijn significant meer richtingen nodig om een gelijkaardige prestatie te leveren. Dit resultaat kan gebruikt worden voor modelcompressie: gegeven een laag l met n neuronen en gewichten $W^{(l)}$ en een vector $\boldsymbol{x}$, kunnen we de traditionele operatie $W^{(l)}\boldsymbol{x}$ vervangen door $(W^{(l)}P^T)(P\boldsymbol{x})$ waarbij P een $k \times n$ projectiematrix is die $\boldsymbol{x}$ projecteert op de k

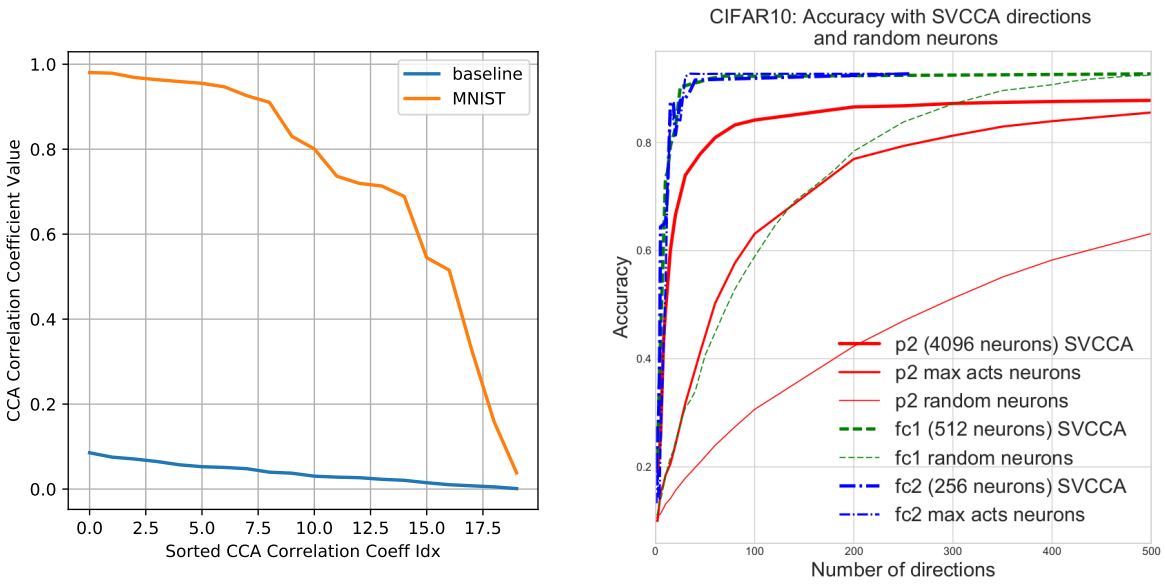
[1] `https://github.com/google/svcca`

belangrijkste SVCCA-richtingen. Op deze manier kan het aantal parameters en dus de computationele kost verminderd worden met een factor $\sim k/n$.

2.2 Reconstructie op DQN

Bovenstaande experimenten hebben we op analoge wijze uitgevoerd, maar ditmaal op twee initialisaties van een neuraal netwerk met één hidden layer van 128 neuronen dat getraind werd op OpenAI Gym Cartpole [46]. Dit is een simulatie van een fysisch systeem waar een stok vastgemaakt is aan een kar die in twee richtingen op een as kan bewegen. De bedoeling is om de stok op de kar te balanceren door de kar op de juiste manier te bewegen. De observatie die de agent krijgt is een viertal (p, v, a, t), met p de positie van de kar op de as, v de huidige snelheid van de kar, a de hoek van de stok en t de huidige bewegingssnelheid van de top van de stok. De agent kan kiezen tussen twee acties: de kar naar links of naar rechts duwen. Indien de hoek van de stok groter wordt dan 24 graden eindigt de episode. De agent krijgt een constante reward voor iedere tijdsstap, en wordt zo aangespoord om de episode zo lang mogelijk te laten duren. De maximale duur van een episode is 500 stappen. Activaties werden verzameld voor 5 episodes, voor een totaal van 2500 invoersamples. Figuur 2.2a toont het resultaat voor het eerste experiment. We zien zeer gelijkaardige resultaten als bij MNIST, wat er opnieuw op wijst dat de geleerde representaties van de neurale netwerken gelijkaardig zijn.

Voor het tweede experiment maken we een onderscheid tussen de beslissingsaccuraatheid en de score: de beslissingsaccuraatheid is het percentage beslissingen die de agent maakt na de projectie die overeenkomen met de beslissing die de originele agent zou maken. De score is het totaal aantal punten die de agent haalt op de environment (hoe lang hij erin slaagt om de paal omhoog te houden). De reden voor dit onderscheid is het feit dat beslissingen die een RL-agent maakt gevolgen kunnen hebben op de lange termijn. Een agent zou dus voor het merendeel van de inputs de correcte beslissing kunnen nemen, maar toch nog zeer slecht presteren indien enkele cruciale beslissingen fout genomen zouden zijn. Op de figuur

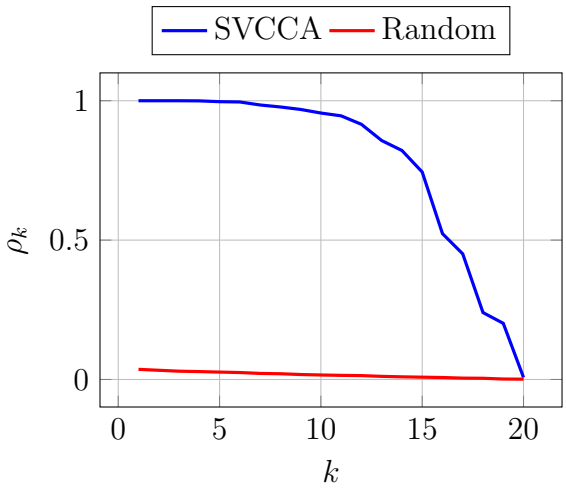


(a) **(b)**

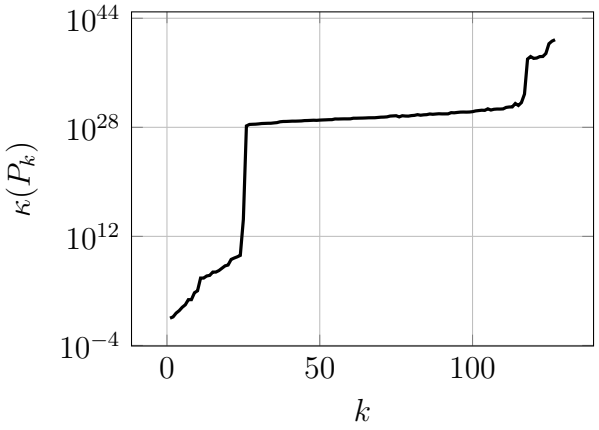
Figuur 2.1: Resultaten van SVCCA. (a): Vergelijking van twee neurale netwerken getraind op MNIST. De 20 hoogste correlatiecoëfficiënten zijn significant hoger dan voor onafhankelijke data. (b): Projectie van hidden layers op top SVCCA-richtingen. Accuraatheid van het model stijgt zeer snel voor een zeer klein aantal richtingen. (figuur genomen uit [19])

zien we echter dat de projectie op de top k SVCCA-richtingen volgens beide metrieken zeer slecht presteert.

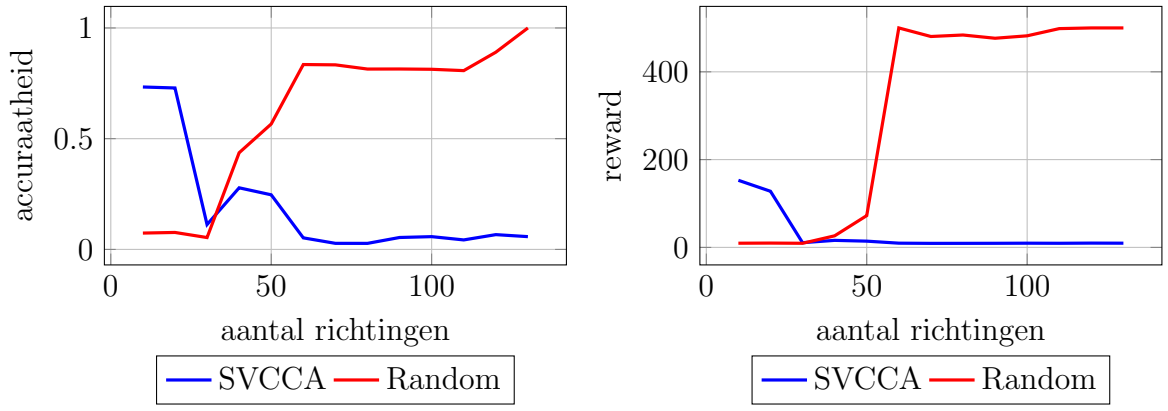
Opvallend is het feit dat de agent nog altijd slecht presteert indien geprojecteerd wordt op *alle* 128 richtingen. Indien de 128 richtingen lineair onafhankelijk zijn, zou de projectie van de 128-dimensionale output van de hidden layer op deze 128 richtingen identiek moeten zijn aan de output zelf, wat dus perfecte nauwkeurigheid zou opleveren (zoals we zien bij willekeurige richtingen). Om dit te verklaren, kijken we naar het conditiegetal van de projectiematrix P_k voor de top k SVCCA-richtingen $\kappa(P_k) = \frac{\sigma_{\max}(P_k)}{\sigma_{\min}(P_k)}$, waarbij $\sigma_{\max}(P_k)$ en $\sigma_{\min}(P_k)$ resp. de grootste en kleinste singuliere waarden zijn van P_k. Matrices met een zeer groot conditiegetal zijn *bijna* singulier, wat kan leiden tot grote numerieke fouten in matrixvermenigvuldigingen. Het conditiegetal van P_k voor variërende waarde van k wordt weergegeven in figuur 2.2b. We zien het conditiegetal inderdaad voor zeer lage waarden van k al zeer snel toenemen, wat suggereert dat de slechte prestaties te wijten zijn aan numerieke fouten.



(a) Vergelijking van twee neurale netwerken getraind met DQN op Cartpole. De 20 hoogste correlatiecoëfficiënten zijn opnieuw significant hoger dan voor onafhankelijke data.



(b) Conditiegetal van de projectiematrix voor top k SVCCA-richtingen, voor variërende waarde van k. We zien het conditiegetal enorm snel toenemen met toenemende k (merk op dat de y-as een logaritmische schaal heeft).



Figuur 2.3: Projectie van de hidden layer van DQN op top SVCCA-richtingen. Links: beslissingsaccuraatheid, rechts: score. Op beide grafieken presteert SVCCA bijzonder slecht.

Hoofdstuk 3

Knowledge en Policy Distillation

3.1 Knowledge Distillation

Knowledge distillation is een techniek die recent in populariteit gestegen is, mede door de succesvolle resultaten verkregen door Hinton et al. [1] die knowledge distillation gebruikten om een ensemble van modellen te distilleren in één enkel model. Het idee achter knowledge distillation is als volgt: eerst wordt een groot, krachtig model getraind op een bepaalde dataset of taak. Dit krachtig model wordt in veel gevallen de *teacher* genoemd. Vervolgens wordt een kleiner, minder krachtig model (de *student*) getraind op de output van die teacher (deze outputs worden ook *soft targets* genoemd). Aangezien de output van de teacher meer informatie bevat dan de labels in de oorspronkelijke dataset (zie verder), kan de student hieruit efficiënter leren. Zo kunnen we betere resultaten bereiken dan indien we de student rechtstreeks zouden trainen op de data.

Knowledge distillation kent uiteenlopende toepassingen, waaronder modelcompressie [47, 48], inferentieversnelling [49], joint co-training [50] en ook het verhogen van interpretabiliteit van modellen [51, 52]. Ook op reinforcement learning kan knowledge distillation toegepast worden [2]. In dit geval spreekt men eerder over *policy distillation*, aangezien reinforcement learning op zich een fundamenteel ander probleem is dan supervised learning,

wat voor extra complicaties zorgt bij het toepassen van knowledge distillation.

In classificatieproblemen wordt in de meeste gevallen gebruik gemaakt van een softmax-functie om waarschijnlijkheidswaarden te bekomen voor de verschillende outputklassen:

$$\sigma(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{3.1}$$

Deze waarschijnlijkheidswaarden geven de geschatte kans weer dat een bepaalde invoer bij de gegeven klassen behoort. Het belangrijkste verschil met de oorspronkelijke labels hier is dat *elke* klasse een waarschijnlijkheidswaarde toegekend wordt, terwijl de labels enkel een 1 bevatten voor de juiste klasse. Wanneer een model echter volledig getraind is en dus voor de invoerdata een hoge "confidence" bezit, zijn de waarschijnlijkheidswaarden voor de foute klassen meestal zeer klein. Een model dat getraind is op MNIST [5] kan bijvoorbeeld aan een afbeelding van een 2 een kans van 10^{-6} geven dat het een 7 is, en een kans van 10^{-9} dat het een 3 is. Deze waarden bevatten belangrijke informatie: namelijk welke afbeeldingen van een 2 eerder *lijken* op een 7, en welke eerder lijken op een 3. Wanneer de student echter rechtstreeks op deze uitvoerwaarden getraind wordt, zal dit onderscheid slechts een zeer kleine invloed hebben op de kostfunctie, aangezien de waarden zo dicht bij 0 liggen.

Om dit probleem op te lossen, wordt de softmax-functie uitgebreid met een parameter die *temperatuur* τ genoemd wordt:

$$\sigma_\tau(\boldsymbol{z})_i = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)} \tag{3.2}$$

Wanneer $\tau > 1$ zorgt dit voor een meer gelijkmatig verdeelde kansverdeling, waardoor de student meer informatie uit de soft targets kan halen. Tijdens de training van het studentmodel wordt dan dezelfde waarde voor τ gebruikt om de loss te berekenen. Na het trainen kunnen we dan gewoon weer de normale softmax gebruiken ($\tau = 1$).

3.2 Policy Distillation

Policy distillation is de natuurlijke extensie van knowledge distillation naar reinforcement learning. Het doel van de student is nu om een policy (zie sectie 1.2.1) te leren van de teacher. Deze extensie is niet triviaal: reinforcement learning verschilt van supervised learning in het feit dat acties gevolgen op lange termijn kunnen hebben, en daarnaast worden in veel RL-algoritmen geen kansverdelingen geleerd op acties. De Q-waarden in Q-learning vormen bijvoorbeeld geen geldige kansverdeling. Met enkele uitbreidingen kan distillatie echter ook op reinforcement learning toegepast worden. Zo werd policy distillation al gebruikt om modellen te trainen op taken die ze rechtstreeks niet kunnen beheersen [53], sterkere policies te leren [54] en multi-task learning te implementeren [2, 55]. In dit hoofdstuk behandelen we specifiek de wijzigingen die aan het oorpsronkelijke knowledge distillation-algoritme gebracht moeten worden om distillation toe te passen op Q-learning. Een overzicht van het policy distillation-algoritme wordt weergegeven in algoritme 5.

De belangrijkste extra complicatie bij policy distillation is het selecteren van de controle-policy. Dit is de policy die gebruikt wordt om te interageren met de environment. Deze interacties worden opgeslagen in een replay memory waaruit dan samples gehaald worden om de student op te trainen (zie figuur 3.2). We hebben twee opties voor de controle-policy (zie figuur 3.1): de policy van de teacher en de policy van de student. In het oorspronkelijke werk over policy distillation [2] wordt de teacher policy gebruikt, maar in recenter onderzoek werden betere resultaten behaald wanneer de student policy gebruikt wordt [56, 53]. Dit kan intuïtief ingezien worden als volgt: een volledig getrainde teacher volgt in veel gevallen een bijna deterministische policy. Tijdens de training heeft die teacher echter meestal een veel breder bereik van toestanden geobserveerd, en ook voor deze toestanden waar de teacher minder in terechtkomt heeft hij een policy geleerd. Als we dan een nieuwe agent laten leren uit de teacher policy, komt deze agent in aanraking met een veel kleinere fractie van de toestandsruimte van het probleem, waarop hij gaat overfitten [57].

Een ander verschil met supervised knowledge distillation is de keuze van kostfunctie $\mathcal{L}(\boldsymbol{q}^T, \boldsymbol{q}^S)$,

waarbij $\boldsymbol{q}^T$ en $\boldsymbol{q}^S$ resp. de Q-waarden van de teacher en de student zijn. Bij Q-learning wordt gewoonlijk gebruik gemaakt van de MSE loss:

$$\mathcal{L}_{MSE}(\boldsymbol{q}^T, \boldsymbol{q}^S) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{q}_i^T - \boldsymbol{q}_i^S)^2$$

Deze kostfunctie kan in principe ook gebruikt worden voor policy distillation, maar de beste resultaten worden in de praktijk gehaald door de temperature Kullback Leibler divergence [2]:

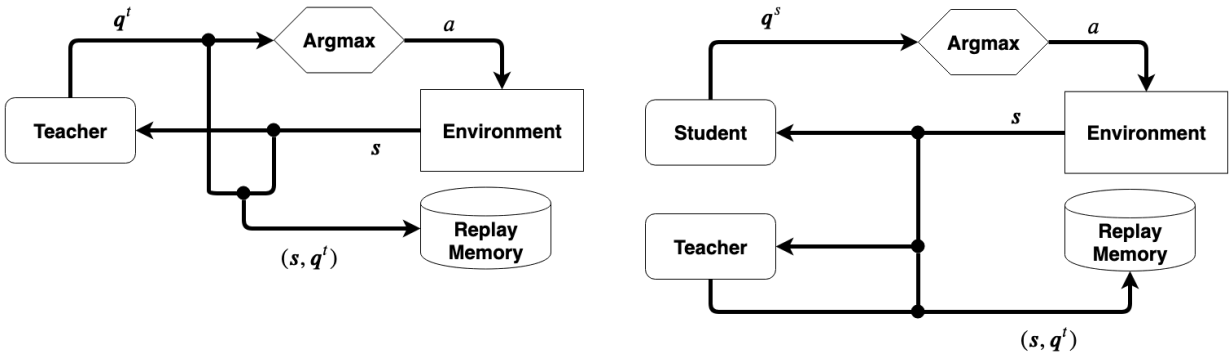
$$L_{KL}(\mathbf{q}^T, \mathbf{q}^S) = \sigma_\tau\left(\mathbf{q}^T\right) \ln \frac{\sigma_\tau\left(\mathbf{q}^T\right)}{\sigma\left(\mathbf{q}^S\right)} \tag{3.3}$$

$$= \sigma\left(\frac{\mathbf{q}^T}{\tau}\right) \ln \frac{\sigma\left(\frac{\mathbf{q}^T}{\tau}\right)}{\sigma\left(\mathbf{q}^S\right)} \tag{3.4}$$

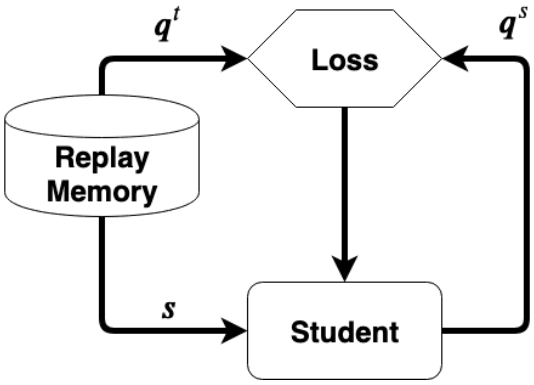
Waarbij τ de gebruikte temperatuur aanduidt. Een belangrijk verschil met de temperature softmax uit supervised knowledge distillation is hier de keuze van temperatuur. Bij supervised learning wordt deze temperatuur groter dan 1 gekozen, wat ervoor zorgt dat de kansverdeling meer gelijkmatig verdeeld wordt. Bij Q-learning zitten we echter meestal met het omgekeerde probleem: een zeer klein verschil in Q-waarden kan een volledig verschillende policy met zich meebrengen. Daarom kiezen we in policy distillation de temperatuur eerder kleiner dan 1.

Algoritme 5 Policy distillation voor Q-waarden

Initialiseer replay memory D met grootte N, teacher T en student S.
Definiëer controlepolicy $C \leftarrow T$ of $C \leftarrow S$.
Definiëer kostfunctie $\mathcal{L}(\boldsymbol{q}^t, \boldsymbol{q}^s)$
for all episodes **do**
    Initialiseer toestand s
    while episode niet geëindigd **do**
        Genereer Q-waarden $\boldsymbol{q}$ uit C.
        Kies actie a volgens ϵ-greedy policy afgeleid uit $\boldsymbol{q}$.
        Voer actie a uit en observeer r, s'.
        if $C = T$ **then**
            Sla $(s, \boldsymbol{q})$ op in D.
        else
            Genereer Q-waarden $\boldsymbol{q}^t$ uit T.
            Sla $(s, \boldsymbol{q}^t)$ op in D
        end if
        $s \leftarrow s'$
        Trek een willekeurige minibatch B van koppels $(s_i, \boldsymbol{q}_i^t)$ uit D
        for all $(s_i, \boldsymbol{q}_i^t) \in B$ **do**
            Genereer Q-waarden $\boldsymbol{q}_i^s$
            Bereken loss $\mathcal{L}(\boldsymbol{q}_i^t, \boldsymbol{q}_i^s)$
        end for
        Voer een gradient descent-stap uit volgens de losses $\mathcal{L}(\boldsymbol{q}_i^t, \boldsymbol{q}_i^s)$
    end while
end for



Figuur 3.1: Controle-policy voor policy distillation. Links: Teacher policy: de teacher interageert met de environment en slaat zijn interacties op in de replay memory. Rechts: Student policy: de student interageert met de environment en slaat de gekregen inputs op in de replay memory, samen met de Q-values die de teacher genereert op die inputs. In beide figuren staan $\boldsymbol{q}^t$ en $\boldsymbol{q}^s$ voor de Q-values gegenereerd door resp. de teacher en de student, a voor de actie, en $\boldsymbol{s}$ voor de toestandsvector.



Figuur 3.2: Policy Distillation: Tijdens de interactie met de environment (figuur 3.1) wordt op regelmatige tijdstippen een willekeurige sample uit de replay memory genomen. De toestand wordt als invoer aan de student gegeven, die hiervoor Q-values genereert. Deze Q-values worden vergeleken met de Q-values van de teacher in de kostfunctie, en deze loss wordt gebruikt om de student te trainen.

Hoofdstuk 4

Vaagcontrole

Vaagverzamelingen werden in 1965 ingevoerd door Lotfi Zadeh als een veralgemening van klassieke verzamelingen [58]. In een klassieke verzameling geldt voor ieder element in het universum dat het volledig wel of volledig niet tot de verzameling behoort. In een vaagverzameling, daarentegen, beschrijven we het lidmaatschap van een element tot een verzameling met een waarde uit het eenheidsinterval. Hierbij staat 0 voor perfect niet-lidmaatschap, 1 voor perfect lidmaatschap, en alle waarden uit $]0, 1[$ voor partieel lidmaatschap. We kunnen klassieke verzamelingen dus beschouwen als een speciaal geval van vaagverzamelingen waarvan de lidmaatschapsgraad voor alle elementen ofwel 0 ofwel 1 is.

In dit hoofdstuk geven we een overzicht van vaagverzamelingen en de bijhorende uitbreidingen van klassieke bewerkingen op verzamelingen. Hiervoor definiëren we eerst de uitbreiding van logische operatoren tot het eenheidsinterval. Met behulp van deze operatoren definiëren we de uitbreidingen van unie, doorsnede en complement, en vervolgens gebruiken we deze bewerkingen om een similariteitsmaat tussen vaagverzamelingen te definiëren. Ten slotte gebruiken we al deze constructies om systemen voor vaagcontrole te ontwikkelen.

4.1 Definities

4.1.1 Logische operatoren op het eenheidsinterval

We onderscheiden drie operatoren op het eenheidsinterval: de negator, de triangulaire norm en de triangulaire conorm [59]. Deze operatoren komen overeen met resp. negatie, conjunctie en disjunctie.

Een negator is een afbeelding $N : [0,1] \to [0,1]$ waarvoor geldt:

$$N(1) = 0, N(0) = 1$$
$$\forall (x_1, x_2) \in [0,1]^2 : x_1 \leq x_2 \Rightarrow N(x_1) \geq N(x_2)$$

Een negator N wordt ook *involutief* genoemd indien $\forall x \in [0,1] : N(N(x)) = x$. De meest bekende en simpelste negator is $N_s(x) = 1 - x$ en wordt ook de standaardnegator genoemd. De randvoorwaarden $N(1) = 0$ en $N(0) = 1$ verzekeren dat elke negator een uitbreiding is van de negatie uit klassieke logica: voor iedere "scherpe" waarde (d.w.z. 0 of 1) gedraagt een negator zich op dezelfde manier als de klassieke negatie.

Een *triangulaire norm* (kort: t-norm) is een afbeelding $T : [0,1]^2 \to [0,1]$ waarvoor geldt:

$$\forall x \in [0,1] : T(x,1) = x$$
$$\forall (x,y) \in [0,1]^2 : T(x,y) = T(y,x)$$
$$\forall (x,y,z) \in [0,1]^2 : T(T(x,y),z) = T(x,T(y,z))$$
$$\forall (x_1,y_1,x_2,y_2) \in [0,1]^4 : x_1 \leq y_1 \wedge x_2 \leq y_2 \Rightarrow T(x_1,x_2) \leq T(y_1,y_2)$$

Een *triangulaire conorm* (kort: t-conorm) is een afbeelding $S : [0,1]^2 \to [0,1]$ waarvoor

geldt:

$$\forall x \in [0,1] : S(x,0) = x$$
$$\forall (x,y) \in [0,1]^2 : S(x,y) = S(y,x)$$
$$\forall (x,y,z) \in [0,1]^2 : S(S(x,y),z) = S(x,S(y,z))$$
$$\forall (x_1,y_1,x_2,y_2) \in [0,1]^4 : x_1 \leq y_1 \wedge x_2 \leq y_2 \Rightarrow S(x_1,x_2) \leq S(y_1,y_2)$$

Men kan eenvoudig verifiëren dat voor iedere t-norm T geldt: $T(0,1) = T(1,0) = T(0,0) = 0$ en $T(1,1) = 1$. Dit betekent dat elke triangulaire norm over $\{0,1\}$ samenvalt met de klassieke conjunctie. Analoog valt iedere t-conorm samen met de klassieke disjunctie over $\{0,1\}$: $S(0,1) = S(1,0) = S(1,1) = 1$ en $S(0,0) = 0$.

Een *de Morgan triplet* is een drietal (T,S,N) bestaande uit een t-norm T, een t-conorm S en een involutieve negator N zodat:

$$\forall (x,y) \in [0,1]^2 : T(x,y) = N(S(N(x),N(y)) \tag{4.1}$$

Men noemt T dan de N-duale t-norm van S en S de N-duale conorm van T. Indien $N = N_s$, spreekt men gewoon over de duale t-norm van S en de duale t-conorm van T. Enkele voorbeelden van duale t-normen en t-conormen zijn $T_M = \min$ met $S_M = \max$, en het product $T_P(x,y) = xy$ met de probabilistische som $S_P(x,y) = x + y - xy$.

Merk op dat we uit eigenschap 4.1 en de involutiviteit van N de volgende relaties kunnen afleiden:

$$\forall (x,y) \in [0,1]^2 : N(T(x,y)) = S(N(x),N(y))$$
$$\forall (x,y) \in [0,1]^2 : N(S(x,y)) = T(N(x),N(y))$$

Dit zijn de continue uitbreidingen van de bekende wetten van De Morgan:

$$\neg(p \wedge q) = (\neg p \vee \neg q)$$
$$\neg(p \vee q) = (\neg p \wedge \neg q)$$

We kunnen dus intuïtief inzien dat het voor de meeste praktische toepassingen aan te raden is om een de Morgan triplet te gebruiken.

4.1.2 Vaagverzamelingen

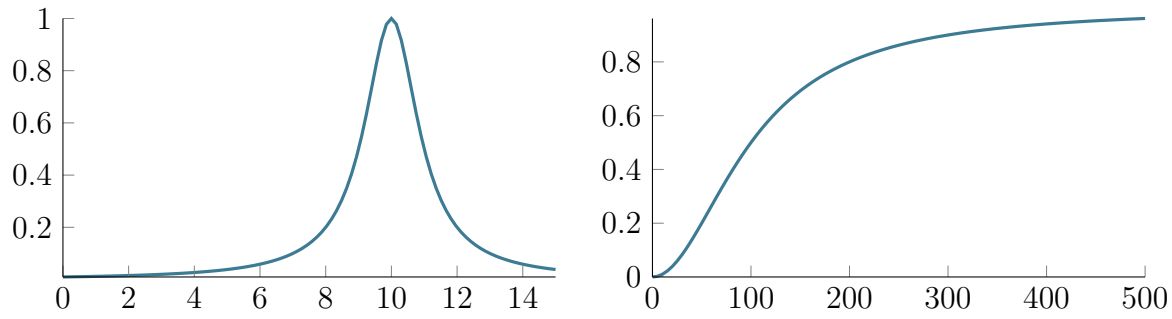
Formeel definiëren we een vaagverzameling en de bijhorende uitbreidingen van klassieke bewerkingen op verzamelingen als volgt:

- Een vaagverzameling is een afbeelding $A : \mathbb{R} \to [0, 1]$. Deze afbeelding wordt ook de *lidmaatschapsfunctie* genoemd. De verzameling van vaagverzamelingen op de reële getallen wordt aangeduid als $\mathcal{F}(\mathbb{R})$.
- Zij A, B twee vaagverzamelingen, T een t-norm, S een t-conorm en N een negator. We definiëren dan:
  - De S-unie van A en B: $A \cup_S B : \mathbb{R} \to [0, 1]; x \to S(A(x), B(x))$
  - De T-doorsnede van A en B: $A \cap_T B : \mathbb{R} \to [0, 1]; x \to T(A(x), B(x))$
  - het N-complement van A: $co_N A : \mathbb{R} \to [0, 1]; x \to N(A(x))$

Figuur 4.1 geeft twee voorbeelden van vaagverzamelingen. De linkerfiguur beeldt een mogelijke representatie af van alle reële getallen dicht gelegen bij 10, met lidmaatschapsfunctie $A(x) = \frac{1}{1+(x-10)^2}$. Het enige getal met lidmaatschap 1, wat dus betekent dat het perfect aan dit predicaat voldoet, is 10 zelf. de rechterfiguur beeldt een mogelijke representatie af van alle grote reële getallen, met lidmaatschapsfunctie $B(x) = \left(1 + \left(\frac{100}{x}\right)^2\right)^{-1}$. Merk op dat dit niet de enige mogelijke representaties zijn van die predicaten. De precieze lidmaatschapsfunctie is altijd context-afhankelijk.

4.1.3 Kardinaliteit en similariteit

We kunnen nu gebruik maken van de definities voor doorsnede en unie om kardinaliteits- en similariteitsmaten op te stellen voor vaagverzamelingen. Een similariteitsmaat geeft weer in welke mate verschillende vaagverzamelingen op elkaar "lijken". Dit zal van groot belang zijn bij het vereenvoudigen van vaagcontrolesystemen.



Figuur 4.1: Voorbeelden van vaagverzamelingen. Links: mogelijke voorstelling van "reële getallen dicht gelegen bij 10". Rechts: mogelijke voorstelling van "grote reële getallen".

Voor klassieke verzamelingen wordt de kardinaliteit gedefiniëerd als het "aantal elementen" in de verzameling, waarbij voor oneindige verzamelingen een onderscheid gemaakt wordt tussen aftelbare en overaftelbare verzamelingen. Voor eindige verzamelingen kan dit idee gemakkelijk uitgebreid worden naar vaagverzamelingen. De Luca en Termini stelden zo in 1972 de zogenaamde σ-count voor [60]:

$$|A| = \sigma(A) = \sum_{x \in supp(A)} A(x)$$

Waarbij *supp* staat voor *support*, de verzameling van elementen x zodat $A(x) > 0$. De σ-count kan op zijn beurt ook uitgebreid worden naar oneindige verzamelingen door de oneindige integraal over elementen te nemen in plaats van de som:

$$|A| = \sigma(A) = \int_{-\infty}^{+\infty} A(x)dx \tag{4.2}$$

Similariteitsmaten geven de mate weer waarin twee vaagverzamelingen overlappen of lijken op elkaar. Aangezien deze notie ook zinvol is voor klassieke verzamelingen, bestaan er ook hiervan al implementaties voor klassieke verzamelingen. Een mogelijke implementatie is de Jaccard index:

$$J(A,B) = \begin{cases} \frac{|A \cap B|}{|A \cup B|} & \text{indien } A \cup B \neq \emptyset \\ 1 & \text{anders} \end{cases}$$

Voor vaagverzamelingen kunnen we de Jaccard-index implementeren voor iedere mogelijke keuze van t-norm en (duale) t-conorm. Merk wel op dat niet iedere keuze een bruikbare similariteitsmaat oplevert. Om dit in te zien, definiëren we eerst de basisvoorwaarden waarvan we verwachten dat ze in een bruikbare similariteitsmaat voldaan zijn. We noemen een afbeelding $M : \mathcal{F}(\mathbb{R}) \times \mathcal{F}(\mathbb{R}) \to [0, 1]$ een geldige similariteitsmaat indien:

$$\forall (A, B) \in \mathcal{F}(\mathbb{R})^2 : M(A, B) = M(B, A)$$
$$\forall A \in \mathcal{F}(\mathbb{R}) : M(A, A) = 1$$

Hieruit kunnen we afleiden dat de Jaccard-index op basis van het minimum en het maximum een geldige similariteitsmaat is, maar de Jaccard-index op basis van het product en de probabilistische som is dat niet (deze similariteitsmaat is niet reflexief). We kunnen de formule wel aanpassen om een geldige similariteitsmaat te bekomen, de Jaccard index op basis van het product:

$$J(A, B) = \begin{cases} \frac{|A \cap_{T_P} B|}{|A \cap_{T_P} A| + |B \cap_{T_P} B| - |A \cap_{T_P} B|} & \text{indien } A \cup B \neq \emptyset \\ 1 & \text{anders} \end{cases} \tag{4.3}$$

4.2 Vaagcontrole

De principes van vaaglogica hierboven beschreven hebben toepassingen in de controletheorie onder de vorm van zogenaamde *vaagregelaars* (Eng. *fuzzy controller*). Dit zijn controlesystemen op basis van *vaagregels*. Zo'n regels zijn typisch van de vorm ALS V_X is A DAN V_Y is B, waarbij V_X en V_Y resp. de invoer- en uitvoervariabelen zijn en A en B "karakteristieken" waaraan de variabelen al dan niet aan kunnen voldoen. Vaagregels laten toe om expertkennis op een linguïstische manier uit te drukken als benaderende omschrijving van een complex, niet-lineair besturingsproces.

4.2.1 De Mamdani vaagregelaar

Het eerste type vaagregelaar is de Mamdani vaagregelaar [61]. Een Mamdani vaagregelaar kan voorgesteld worden als een verzameling van n vaagregels in m invoerveranderlijken:

$$\begin{gathered}\text{ALS } x_1 \text{ is } A_{11} \text{ EN } \ldots \text{ EN } x_m \text{ is } A_{m1} \text{ DAN } y \text{ is } B_1\\ \text{ALS } x_1 \text{ is } A_{12} \text{ EN } \ldots \text{ EN } x_m \text{ is } A_{m2} \text{ DAN } y \text{ is } B_2\\ \ldots\\ \text{ALS } x_1 \text{ is } A_{1n} \text{ EN } \ldots \text{ EN } x_m \text{ is } A_{mn} \text{ DAN } y \text{ is } B_n\end{gathered}$$

Hier zijn alle A_{ij} en B_i vaagverzamelingen in $\mathbb{R}$ die manueel met behulp van expertkennis worden opgesteld (bijvoorbeeld "zeer groot" of "gematigd klein"). x_i zijn de invoerveranderlijken en y de uitvoer. Gegeven een invoervector $(x_1, \ldots, x_m)$ en een de Morgan triplet (T, S, N), worden voor elke regel de activatiegraad ρ en de conclusie B'_i bepaald als:

$$\begin{aligned}\rho &= T(A_{1i}(x_1), \ldots, A_{mi}(x_m))\\ B'_i(y) &= T(\rho, B_i(y))\end{aligned}$$

Het globale inferentieresultaat B' wordt berekend door de disjunctie te nemen van alle conclusies met behulp van de T-conorm S:

$$B'(y) = \underset{i=1}{\overset{n}{S}}(B'_i(y))$$

Dit globale inferentieresultaat is een vaagverzameling, maar om een systeem aan te sturen hebben we een scherpe waarde nodig (een getal). Dit getal wordt bekomen door een zogenaamde *defuzzificatiemethode* D. In de praktijk wordt vaak gebruik gemaakt van de zwaartepuntmethode:

$$D(B') = \frac{\int_{-\infty}^{+\infty} yB'(y)dy}{\int_{-\infty}^{+\infty} B'(y)dy}$$

4.2.2 De Takagi-Sugeno vaagregelaar

Omdat de vorm van het inferentieresultaat B' in principe arbitrair kan zijn, moet de uiteindelijke scherpe waarde in de meeste gevallen numeriek benaderd worden. Dit heeft als gevolg dat de defuzzificatiestap de computationeel zwaarste stap is van de Mamdani vaagregelaar. Als optimalisatie kunnen we dus de vaagverzamelingen in de consequenten vervangen door scherpe waarden. Deze scherpe waarden kunnen constanten zijn, of functies van de invoerveranderlijken. We bekomen dan een Takagi-Sugeno vaagregelaar [62]. Algemeen heeft de Takagi-Sugeno vaagregelaar de volgende vorm:

$$\begin{gathered}
\text{ALS } x_1 \text{ is } A_{11} \text{ EN } \ldots \text{ EN } x_m \text{ is } A_{m1} \text{ DAN } y \text{ is } f_1(x_1, \ldots, x_m) \\
\text{ALS } x_1 \text{ is } A_{12} \text{ EN } \ldots \text{ EN } x_m \text{ is } A_{m2} \text{ DAN } y \text{ is } f_2(x_1, \ldots, x_m) \\
\ldots \\
\text{ALS } x_1 \text{ is } A_{1n} \text{ EN } \ldots \text{ EN } x_m \text{ is } A_{mn} \text{ DAN } y \text{ is } f_n(x_1, \ldots, x_m)
\end{gathered}$$

Het globale inferentieresultaat wordt berekend als een gewogen gemiddelde van de scherpe uitvoerwaarden, waarbij de activatiegraden als gewichten gebruikt worden. De activatiegraden zelf worden op dezelfde manier berekend als bij de Mamdani vaagregelaar.

4.2.3 De Neuro-Fuzzy vaagregelaar (ANFIS)

Een belangrijk nadeel aan Mamdani- en Takagi-Sugeno vaagregelaars is dat de lidmaatschapsfuncties en uitvoerfuncties van de vaagregels nog steeds manueel bepaald moeten worden met behulp van expertkennis. Om passende vaagregels te kunnen aanleren uit data, zijn daarom *neuro-fuzzy* vaagregelaars ontwikkeld. Het idee is om een vaagregelaar voor te stellen als een soort neuraal netwerk, en dat netwerk dan op analoge wijze te trainen als een gewoon neuraal netwerk. De meest gebruikte methode hiervoor heet ANFIS: *Adaptive-Network-based Fuzzy Inference System* [63]. Net als gewone neurale netwerken zijn ANFIS vaagregelaars *universele approximatoren*, wat betekent dat een systeem met genoeg vaagregels elke continue functie met arbitraire precisie kan benaderen. Figuur 4.2

geeft de structuur weer van een ANFIS vaagregelaar met twee invoerveranderlijken x en y en twee Takagi-Sugeno vaagregels. Het netwerk bestaat uit vijf lagen met vierkante of cirkelvormige nodes. Vierkante nodes maken gebruik van bepaalde parameters die uit de data aangeleerd worden, terwijl cirkels geen parameters bevatten. We overlopen nu de werking van de vijf lagen:

1. De waarden van de lidmaatschapsfuncties (A_i en B_i in figuur 4.2) worden berekend. Over het algemeen worden hiervoor lidmaatschapsfuncties van dezelfde vorm gebruikt, bv.
$$A_i(x) = e^{-\left(\frac{x-b_i}{a_i}\right)^2}$$
met als parameters a_i en b_i.

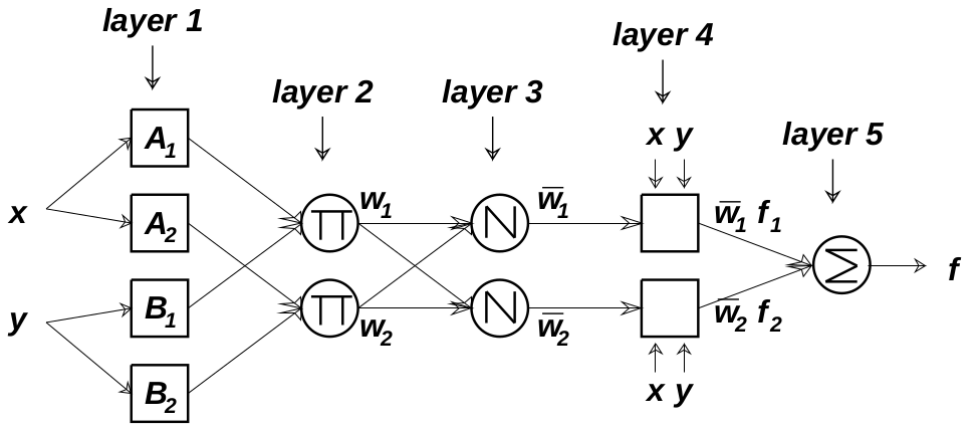
2. De activatiegraad w_i van elke regel wordt berekend als de T-norm van de verschillende lidmaatschapsgraden van de invoerwaarden.

3. De activatiegraden worden genormaliseerd:
$$\overline{w_i} = \frac{w_i}{\sum_j w_j}$$

4. De uitvoerwaarde voor iedere regel wordt berekend als $\overline{w_i} f_i(x, y)$. Hier kan $f_i(x, y)$ eender welke parametriseerbare functie zijn van de invoerwaarden. In de praktijk wordt meestal gebruik gemaakt van lineaire of constante functies.

5. De globale uitvoerwaarde wordt berekend als gewogen gemiddelde van de uitvoerwaarden:
$$y = \sum_i \overline{w_i} f_i(x, y)$$

Het aanleren van passende parameters voor ANFIS gebeurt op dezelfde manier als bij normale neurale netwerken: door de fout E van de voorspellingen van het netwerk op de data terug te laten propageren ("backpropagation") wordt voor iedere parameter a de partiële afgeleide $\frac{\partial E}{\partial a}$ berekend. Vervolgens worden de parameters aangepast met behulp van gradient descent.



Figuur 4.2: ANFIS vaagregelaar met twee Takagi-Sugeno vaagregels.

Hoofdstuk 5

Gebruik van vaagcontrole voor policy distillation

Policy distillation werd in het verleden al succesvol voor verschillende doeleinden toegepast. Enkele van deze toepassingen zijn het trainen van architecturen die rechtstreeks niet handelbaar zijn [53], het trainen van robuustere policies [54] en multitask learning [2, 55]. In dit hoofdstuk proberen we policy distillation toe te passen om de interpretabiliteit van een model te verhogen, door het idee te combineren met de neuro-fuzzy vaagregelaar besproken in hoofdstuk 4.2.3. Indien het aantal vaagverzamelingen in een vaagregelaar klein genoeg is, kunnen we intuïtieve beschrijvingen geven van de vaagregels die het systeem gebruikt. Als dit aantal vaagregels klein genoeg is, kunnen we de vaagregelaar in zijn geheel als interpreteerbaar beschouwen. Daarnaast laat de vorm van vaagregelaars ons toe om heuristische optimalisaties door te voeren, zoals het samenvoegen van vaagverzamelingen, identificeren van overbodige vaagverzamelingen, en intelligente initialisatie van parameters. Zo kunnen we het aantal vaagregels zo klein mogelijk houden.

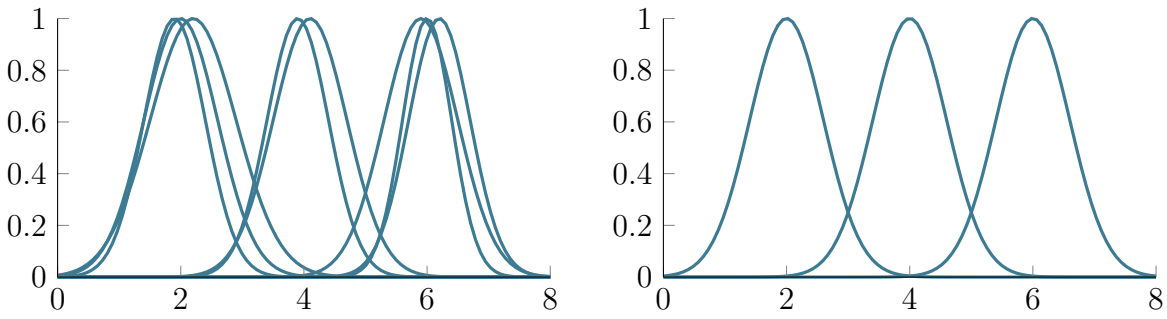
5.1 Constructie van het distillatie-algoritme

We beschrijven nu de concrete implementatie van Policy Distillation die we zullen gebruiken in de experimenten. We focussen hier op distillatie van een DQN-agent, wat betekent dat we de veronderstelling maken dat de output van de teacher een aantal Q-values is voor een eindig aantal acties. We gebruiken ook telkens ANFIS met constante uitvoerwaarden, omdat vaagregels met constante uitvoerwaarden makkelijker te interpreteren zijn dan met lineaire uitvoerfuncties. Vaaginferentiesystemen zoals ANFIS hebben als voordeel dat ze op een interpretabele manier gevormd *kunnen* worden, maar dit is geen garantie. Indien we ANFIS gewoon trainen door willekeurige initialisatie en gradient descent, lopen we het risico dat we te veel vaagregels gebruiken. In dat geval heeft ANFIS bijvoorbeeld de neiging om veel verschillende vaagverzamelingen te vormen die sterk op elkaar lijken, wat een weinig interpreteerbaar model oplevert [64] (zie figuur 5.1). Om de interpretabiliteit van ANFIS te maximaliseren, breiden we het algoritme uit hoofdstuk 3.2 uit naar drie stappen: initialisatie, distillatie en post-processing. We beschrijven deze drie stappen in volgorde.

5.1.1 Initialisatie

Om een goede initialisatie van vaagregels te vormen, wordt vaak gebruik gemaakt van clusteralgoritmes [65]. Dit omdat zulke clusteralgoritmes meestal significant minder rekenintensief zijn dan het trainen van het neuro-fuzzy systeem zelf, en een goede initialisatie het trainen veel kan versnellen. Daarnaast kan een goede initialisatie ook vermijden dat te veel gelijkaardige vaagregels of vaagverzamelingen gevormd worden. In dit werk maken we gebruik van subtractive clustering [66] (zie figuur 5.2).

Subtractive clustering neemt als input een verzameling punten $Z = \{\boldsymbol{z}_i\}$. Het resultaat is een verzameling centrumpunten $C = \{\boldsymbol{c}_i\} \subset Z$, waarbij ieder centrumpunt een cluster definiëert. Als we dus een verzameling van m-dimensionale invoervectoren I en een verzameling van n-dimensionale uitvoervectoren U hebben, kunnen we subtractive clustering

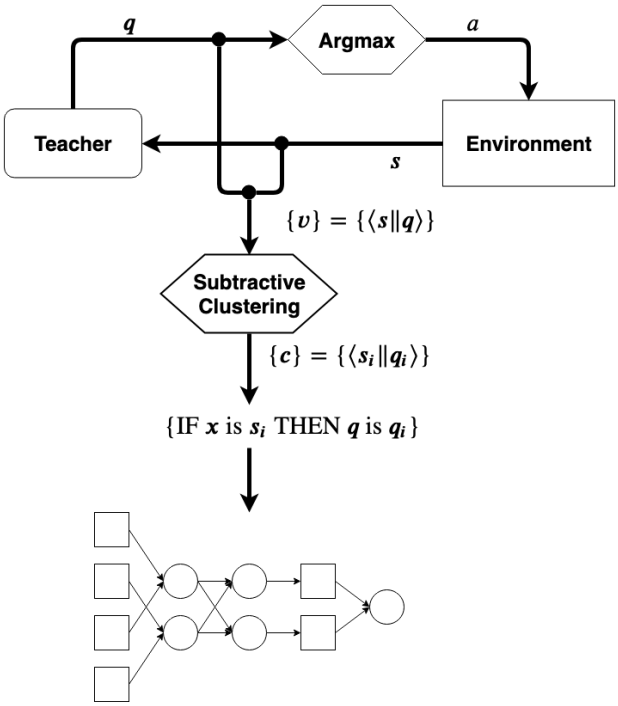


Figuur 5.1: Links: voorbeeld van slecht interpreteerbare vaagverzamelingen. Het interval wordt onderverdeeld in een groot aantal vaagverzamelingen die sterk op elkaar lijken, wat ervoor zorgt dat het geheel moeilijk interpreteerbaar is. Rechts: voorbeeld van interpreteerbare vaagverzamelingen. De opdeling van het interval kan hier gemakkelijk geïnterpreteerd worden als bv. "klein", "gemiddeld" en "groot".

als volgt gebruiken om een aantal vaagregels te definiëren waarmee we ANFIS kunnen initialiseren:

1. Stel de matrix M op door iedere invoervector $\boldsymbol{v}_i \in I$ te concateneren met de bijhorende uitvoervector $\boldsymbol{u}_i \in U$.
2. Gebruik subtractive clustering om een verzameling centrumpunten $C = \{\boldsymbol{c}_i\} \subset M$ te bekomen.
3. Splits ieder centrumpunt $\boldsymbol{c}_i$ terug op in een invoervector $\boldsymbol{v}_{\boldsymbol{c}_i}$ en een uitvoervector $\boldsymbol{u}_{\boldsymbol{c}_i}$.
4. Definiëer voor ieder koppel $(\boldsymbol{v}_{\boldsymbol{c}_i}, \boldsymbol{u}_{\boldsymbol{c}_i})$ de vaagregel: IF $\boldsymbol{x}$ is $\boldsymbol{v}_{\boldsymbol{c}_i}$ THEN $\boldsymbol{y}$ is $\boldsymbol{u}_{\boldsymbol{c}_i}$.

Aangezien we in de context van reinforcement learning geen dataset van invoer- en uitvoervectoren ter beschikking hebben, bouwen we deze dataset op door de teacher voor een aantal episodes met de environment te laten interageren. Bij iedere stap worden de toestand en de Q-values opgeslagen als invoer- en uitvoervectoren. Deze dataset wordt dan zoals hierboven gebruikt om ANFIS te initialiseren. Merk op dat niet elk clusteralgoritme gebruikt kan worden. Aangezien de vaagverzamelingen in de antecedenten van ANFIS



Figuur 5.2: Initialisatie van ANFIS via subtractive clustering: eerst wordt een dataset gegenereerd door de interactie tussen de teacher en de environment op te slaan, hierop wordt subtractive clustering uitgevoerd, en uit de resulterende cluster centra wordt ANFIS gegenereerd. $\boldsymbol{q}$ is de vector van Q-waarden, a is de actie, $\boldsymbol{s}$ is de toestandsvector en $\{\boldsymbol{c}\}$ is de verzameling van clustercentra gegenereerd door subtractive clustering.

gedefiniëerd worden door Gaussische curves en de consequenten constante vectoren zijn, moeten de gegenereerde clusters ook convexe vormen zijn met een duidelijk gedefiniëerd centrum. Algoritmes zoals DBSCAN, waar clusters arbitraire niet-convexe structuren kunnen bezitten en geen duidelijk centrum hebben, zijn niet geschikt voor ANFIS aangezien de resulterende clusters niet omgezet kunnen worden naar vaagregels.

Subtractive clustering werkt als volgt [66]: zij Z een verzameling van N data samples $z_1, ..., z_N$ in een n-dimensionale ruimte. In een eerste stap wordt de data genormaliseerd zodat het bereik van iedere dimensie gelijk is. Vervolgens krijgt ieder datapunt $\boldsymbol{z}_i$ een potentiaal p_i:

$$p_i = \sum_{j=1}^{N} e^{-\alpha||\boldsymbol{z}_i - \boldsymbol{z}_j||^2}$$

Waarbij $\alpha = \frac{4}{r_a^2}$ en $r_a > 0$. r_a is een parameter die de straal van de omgeving rond ieder punt definieert. Punten die buiten deze omgeving liggen, hebben een kleine invloed op de potentiaal, terwijl punten die dicht bij $\boldsymbol{z}_i$ liggen een grotere invloed hebben. Punten met een dichtbevolkte omgeving krijgen dus een hoge potentiaal toegewezen.

Vervolgens wordt het punt met de hoogste potentiaal gekozen als het centrum van de eerste cluster $\boldsymbol{z}_1^*$ met potentiaal p_1^*. Daarna wordt de potentiaal van alle andere punten gereduceerd in verhouding met de afstand tot $\boldsymbol{z}_1^*$. Punten dicht bij $\boldsymbol{z}_1^*$ krijgen een sterkere reductie dan verafgelegen punten:

$$p_i \leftarrow p_i - p_1^* e^{-\beta||\boldsymbol{z}_i - \boldsymbol{z}_1^*||^2}$$

Waarbij $\beta = \frac{4}{r_b^2}$ en $r_b > 0$. r_b is opnieuw een parameter die de straal van de omgeving voor reductie rond ieder punt definieert. Om te vermijden dat verschillende clusters te dicht bij elkaar gevormd worden, wordt r_b meestal iets hoger gekozen dan r_a.

Na het selecteren van de eerste cluster, wordt het tweede clustercentrum opnieuw gekozen als het punt met de hoogste potentiaal, en worden de potentialen opnieuw gereduceerd. Deze procedure herhaalt zich tot aan het stopcriterium is voldaan (zie algoritme 6). Voor

dit stopcriterium zijn opnieuw twee parameters nodig: ϵ^{up} en ϵ^{down}. Deze parameters bepalen wanneer een punt zeker geaccepteerd of zeker geweigerd wordt als clustercentrum. Merk op dat het aantal clusters niet op voorhand gegeven moet worden, maar de parameter r_a is recht evenredig met het resulterend aantal clusters en dus ook met het aantal vaagregels. We moeten dus zoeken naar de kleinste waarde voor r_a zodat het systeem nog expressief genoeg is om het probleem op te lossen.

Algoritme 6 Stopcriterium subtractive clustering

if $p_k^* > \epsilon^{up} p_1^*$ **then**
    Accepteer $\boldsymbol{z}_k^*$ als nieuwe cluster en doe verder
else if $p_k^* < \epsilon^{down} p_1^*$ **then**
    Weiger $\boldsymbol{z}_k^*$ als nieuwe cluster, eindig het algoritme
else
    $d_{min} \leftarrow \min_{i=1,\ldots,k-1} ||\boldsymbol{z}_k^* - \boldsymbol{z}_i^*||$
    if $d_{min}/r_a + p_k^*/p_1^* \geq 1$ **then**
        Accepteer $\boldsymbol{z}_k^*$ als nieuwe cluster
    else
        Weiger $\boldsymbol{z}_k^*$ als nieuwe cluster
        $p_k \leftarrow 0$
        doe verder
    end if
end if

5.1.2 Distillatie

Voor het genereren van trajectories door het MDP werd een epsilon-greedy policy gebruikt op basis van de output van de student (zie figuur 3.1, rechts). Dit om te verzekeren dat de bezochte states voldoende gevarieerd zijn, en omdat in de literatuur gekend is dat het volgen van de student meestal beter werkt dan het volgen van de teacher [56, 53].

Als kostfunctie werd de temperature KL-divergence 3.3 gebruikt met temperatuur $\tau = 0.1$. Na iedere stap worden de state en Q-values van de teacher opgeslagen in een memory buffer (zoals de experience replay buffer bij DQN [18]), en wordt een minibatch genomen waarop de student getraind wordt (zie figuur 3.2). Het distillatie-algoritme wordt gegeven in algoritme 7.

Algoritme 7 Basis distillatie-algoritme

Input Q: teacher netwerk

Output A: ANFIS

Initialiseer replay memory D met grootte N, en ANFIS A met willekeurige parameters.
for all episodes **do**
    Initialiseer toestand s
    while episode niet geëindigd **do**
        Kies actie a uit ϵ-greedy policy afgeleid uit A
        Voer actie a uit
        $\mathbf{q}^T \leftarrow Q(s)$
        Sla sample $(s, \mathbf{q}^T)$ op in D
        $s \leftarrow s'$
        Trek een willekeurige minibatch D' van samples $(s_j, \mathbf{q}_j^T)$ uit D
        Fit het model op D' met loss-functie L_{KL} (3.3)
    end while
end for

5.1.3 Post-processing

Groeperen van vaagverzamelingen

Om de similariteit van twee vaagverzamelingen A, B te berekenen, kunnen we gebruik maken van de Jaccard-index op basis van het product $J(A, B)$ [67] (vergelijking 4.3).

Om deze similariteitsmaat te implementeren moeten we de kardinaliteit van de doorsnede

$|A \cap_{T_P} B|$ berekenen. Hiervoor kunnen we gebruik maken van het feit dat iedere verzameling in onze implementatie van ANFIS gegeven wordt door een Gaussische functie:

$$A(x) = \exp\left(-\left(\frac{x-\mu_A}{\sigma_A}\right)^2\right)$$

Als we de continue σ-count (4.2) gebruiken als kardinaliteitsmaat kunnen we, rekening houdende met $(A \cap_{T_P} B)(x) = T_P(A(x), B(x)) = A(x)B(x)$, de kardinaliteit van $A \cap_{T_P} B$ afleiden als:

$$\begin{aligned}
|A \cap_{T_P} B| &= \int_{-\infty}^{+\infty} A(x)B(x)dx \\
&= \int_{-\infty}^{+\infty} \exp\left(-\left(\frac{x-\mu_A}{\sigma_A}\right)^2\right) \exp\left(-\left(\frac{x-\mu_B}{\sigma_B}\right)^2\right) dx \\
&= \int_{-\infty}^{+\infty} \exp\left(-\left(\frac{x-\mu_A}{\sigma_A}\right)^2 - \left(\frac{x-\mu_B}{\sigma_B}\right)^2\right) dx \\
&= \int_{-\infty}^{+\infty} \exp\left(-\left(\frac{1}{\sigma_A^2} + \frac{1}{\sigma_B^2}\right)x^2 + 2\left(\frac{\mu_A}{\sigma_A^2} + \frac{\mu_B}{\sigma_B^2}\right)x - \frac{\mu_A^2}{\sigma_A^2} - \frac{\mu_B^2}{\sigma_B^2}\right) dx
\end{aligned}$$

Nu weten we dat een oneindige integraal van een arbitraire Gaussische functie gelijk is aan [68]:

$$\int_{-\infty}^{+\infty} \exp(-ax^2 + bx + c)dx = \sqrt{\frac{\pi}{a}} \exp\left(\frac{b^2}{4a} + c\right)$$

Waaruit we dus direct de kardinaliteit van het product (en zo ook de Jaccard-index op basis van het product) van twee vaagverzamelingen direct kunnen berekenen.

Nu we gelijkaardige vaagverzamelingen kunnen identificeren na het trainen van ANFIS, kunnen we deze informatie gebruiken om het systeem simpeler te maken (zie algoritme 8). Voor iedere invoerdimensie voegen we alle koppels van vaagverzamelingen A, B met $J(A, B) > \alpha$ samen, voor een bepaalde threshold-parameter α. Dit herhalen we tot er geen koppels van vaagverzamelingen meer zijn met similariteit $> \alpha$.

Groeperen van consequenten

Consequenten van vaagregels kunnen op analoge wijze samengevoegd worden als de vaagverzamelingen in de antecedenten: gegeven twee consequentvectoren $\boldsymbol{c}_1$ en $\boldsymbol{c}_2$, kunnen we

Algoritme 8 Samenvoegen van vaagverzamelingen in antecedenten

Input A: ANFIS, α: threshold

Noteer A_{di}: de vaagverzameling in invoerdimensie d voor vaagregel i
Noteer μ_{di}, σ_{di}: μ resp. σ voor vaagverzameling A_{di}.
for all invoerdimensie d **do**
    $s \leftarrow \max_{\substack{i,j \\ i \neq j}}(J(A_{di}, A_{dj}))$
    $(i_m, j_m) \leftarrow \arg\max_{\substack{i,j \\ i \neq j}}(J(A_{di}, A_{dj}))$
    while $s > \alpha$ **do**
        $\mu^* \leftarrow \frac{\mu_{di}+\mu_{dj}}{2}$; $\sigma^* \leftarrow \frac{\sigma_{di}+\sigma_{dj}}{2}$
        Vervang A_{di}, A_{dj} door A^*: $A^*(x) = \exp\left(-\left(\frac{x-\mu^*}{\sigma^*}\right)^2\right)$
    end while
end for

deze vervangen door $\boldsymbol{c}_3 = \frac{\boldsymbol{c_1}+\boldsymbol{c_2}}{2}$. Dit leidt tot het tweede deel van de post-processing-fase: we voegen telkens de twee meest gelijkaardige consequenten samen tot er n unieke consequenten meer overschieten, met n een gekozen parameter (zie algoritme 9).

Aangezien de consequentvectoren gewone vectoren zijn in de n-dimensionale euclidische ruimte, zou men geneigd kunnen zijn om de euclidische afstand te gebruiken om de similariteit tussen consequenten te meten. Dit is echter geen bruikbare techniek, aangezien de consequentvectoren Q-waarden voorstellen, en deze Q-waarden een arbitraire schaal kunnen hebben. Daarnaast kunnen Q-waarden met een zeer kleine euclidische afstand toch een drastisch ander gedrag vertonen: de euclidische afstand tussen $\langle 0.9, 1.1\rangle$ en $\langle 0.99, 0.95\rangle$ is bijvoorbeeld relatief klein, ook al representeren deze twee vectoren verschillende acties als ze als Q-waarden geïnterpreteerd worden. Omgekeerd is de euclidische afstand tussen $\langle 1.0, 0.5\rangle$ en $\langle 5.0, 2.5\rangle$ veel groter, terwijl deze twee vectoren dezelfde actie voorstellen. Daarom wordt de "afstand" tussen twee consequentvectoren berekend als de cross-entropie van de softmax van de vectoren:

$$d(\boldsymbol{c}_1, \boldsymbol{c}_2) = -\sum_i \sigma(\boldsymbol{c}_1)_i * \log(\sigma(\boldsymbol{c}_2)_i)$$

waarbij $\boldsymbol{c}_1$ en $\boldsymbol{c}_2$ consequentvectoren zijn, en σ de softmaxfunctie 3.1 is. Zo beschouwen we de vectoren als probabiliteitsdistributies over acties en worden twee vectoren als "gelijkaardig" beschouwd als ze dezelfde actie representeren.

Algoritme 9 Samenvoegen van consequenten

Input A: ANFIS, n: gewenst aantal unieke consequenten

1: Noteer $\boldsymbol{c}_i$: de consequentvector voor vaagregel i
2: **for** n keer **do**
3: $(i_m, j_m) \leftarrow \arg\max_{\substack{i,j \\ i \neq j}}(d(\boldsymbol{c}_i, \boldsymbol{c}_j))$
4: Vervang $\boldsymbol{c}_i, \boldsymbol{c}_j$ door $\boldsymbol{c}^*$: $\boldsymbol{c}^* = \frac{\boldsymbol{c}_i + \boldsymbol{c}_j}{2}$
5: **end for**

5.2 Resultaten

We testen het distillatie-algoritme met de initialisatiestap uit op drie soorten MDP's: willekeurige gridworlds, een gestructureerde gridworld en de OpenAI Gym Cartpole environment [46]. Hierna passen we de post-processing stap toe om de resulterende ANFIS vaagregelaars zo veel mogelijk te simplificeren.

5.2.1 Willekeurige gridworlds

Voor het eerste experiment genereren we willekeurige gridworld-MDP's. Zo'n gridworlds bestaan uit een discreet rooster van cellen, waarbij elke cel een toestand is waar de agent zich in kan bevinden. Het voordeel van dit discreet rooster is dat een optimale policy gevonden kan worden via dynamisch programmeren, op voorwaarde dat het rooster klein genoeg is. De Q-values die we hieruit verkrijgen kunnen we dan gebruiken om te distilleren.

Constructie

De gridworlds in dit experiment bestaan uit een rooster van 20 op 20 cellen (zie figuur 5.3). De observatie van de agent bestaat uit de x- en y-coördinaten van de cel waar hij zich bevindt. De agent start in de initiële cel met coördinaten $(10, 10)$. Op ieder moment heeft de agent de keuze tussen 4 acties: boven, onder, links of rechts. Bij iedere actie is er 90% kans dat ze correct wordt uitgevoerd en 10% kans dat een willekeurige andere actie genomen wordt. Iedere cel is ofwel een muur, wat betekent dat ze niet begaanbaar is, ofwel bevat ze een (positieve of negatieve) reward. Sommige cellen zijn terminaal, wat betekent dat de episode eindigt wanneer de agent in die cel terechtkomt. Als een agent een actie neemt die hem op een muur zou plaatsen, wordt de actie simpelweg niet uitgevoerd en blijft de agent staan. Ten slotte is er op ieder moment ook 1% kans dat de episode willekeurig getermineerd wordt, om te garanderen dat de episodes eindig zijn.

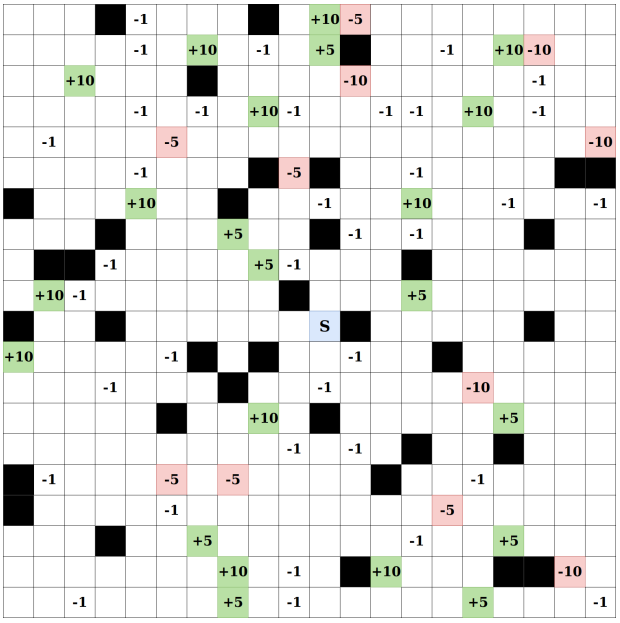
Het indelen van de gridworlds gebeurt volgens algoritme 10. Merk op dat alle cellen met positieve rewards ook terminaal zijn. Dit zorgt ervoor dat de agent gemotiveerd wordt om naar een terminale state te bewegen. Indien er positieve rewards beschikbaar zouden zijn die de episode niet termineren, zou de agent gemotiveerd worden om naar deze rewards te bewegen en dan constant dezelfde cel binnen en buiten te stappen tot de episode toevallig getermineerd wordt. Dit zou enorm veel variatie geven in de verkregen rewards per episode, wat analyse moeilijk maakt.

Resultaten

Figuur 5.4 toont de resultaten van het distillatie-algoritme op willekeurige gridworlds. De grafiek werd opgesteld door 50 willekeurige gridworlds te genereren en voor elke gridworld via dynamisch programmeren een optimale policy te berekenen. Daarna werd elke policy voor 500 episodes gedistilleerd naar ANFIS. De blauwe curve geeft voor iedere episode de mediaan weer van de score (relatief aan de score van de optimale policy) over de 50 willekeurige gridworlds. De twee zwarte curves tonen het 20e en 80e percentiel.

Algoritme 10 Constructie willekeurige gridworld

```
1: r() : genereert willekeurig getal tussen 0 en 1
2: for all cel c do
3:     if c ≠ c_init then
4:         if r() < 1/10 then
5:             Plaats een muur
6:         else if r() < 1/50 then
7:             Plaats een terminale reward +10
8:         else if r() < 1/50 then
9:             Plaats een terminale reward +5
10:        else if r() < 1/10 then
11:            Plaats een niet-terminale reward −1
12:        else if r() < 1/50 then
13:            Plaats een terminale reward −5
14:        else if r() < 1/50 then
15:            Plaats een terminale reward −10
16:        end if
17:    end if
18: end for
```



Figuur 5.3: Voorbeeld van een willekeurige gridworld. Groene en rode cellen zijn terminale toestanden, de blauwe cel is de initiële toestand.

Wat meteen opvalt aan de resultaten van dit experiment is de aanwezigheid van sterke outliers (vanwege het grote verschil tussen de mediaan en het 20e percentiel). Doordat er veel willekeur gebruikt wordt bij het genereren van de MDP's, heeft dit ook een grote invloed op de performantie van het model. We zien echter wel dat het distillatieproces in de meerderheid van de gevallen goed verloopt.

Figuur 5.5 geeft op analoge wijze de resultaten van hetzelfde experiment weer, maar waarbij een lineair model gebruikt werd als student in plaats van ANFIS (hier werd de initialisatiestap via subtractive clustering van het distillatie-algoritme uiteraard overgeslagen). We zien dat het lineair model even goed of beter presteert dan ANFIS. Dit betekent dus dat in veel gevallen de optimale policy een simpele lineaire policy is. De oorzaak hiervan is het feit dat een terminale toestand met reward +10 soms zeer dicht bij de begintoestand geplaatst wordt. Dit heeft als gevolg dat de meerderheid van de gridworld nooit gebruikt of geëxploreerd wordt, wat het probleem triviaal maakt. De resultaten van dit experiment zijn dus niet representatief voor echte problemen, waar de optimale policy meestal

niet-lineair is.

5.2.2 Gestructureerde gridworld

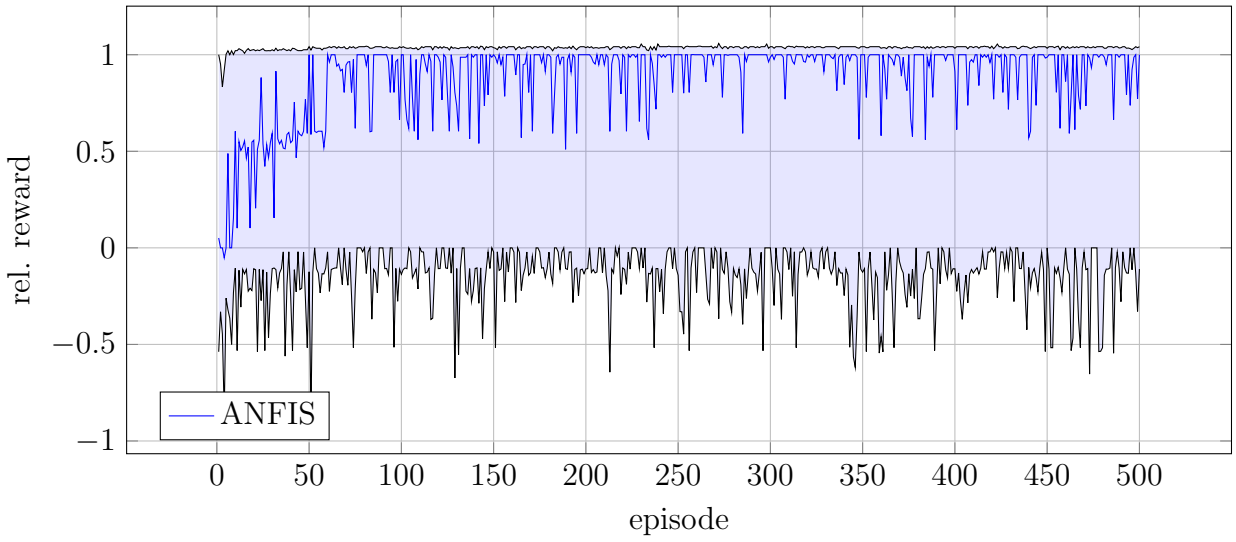
In het volgende experiment wordt de gridworld op een meer gestructureerde manier opgesteld. Dit zorgt ervoor dat de optimale policy gegarandeerd niet-lineair is, en dat de resultaten minder sterke variaties vertonen dan bij willekeurige gridworlds.

Constructie gridworld

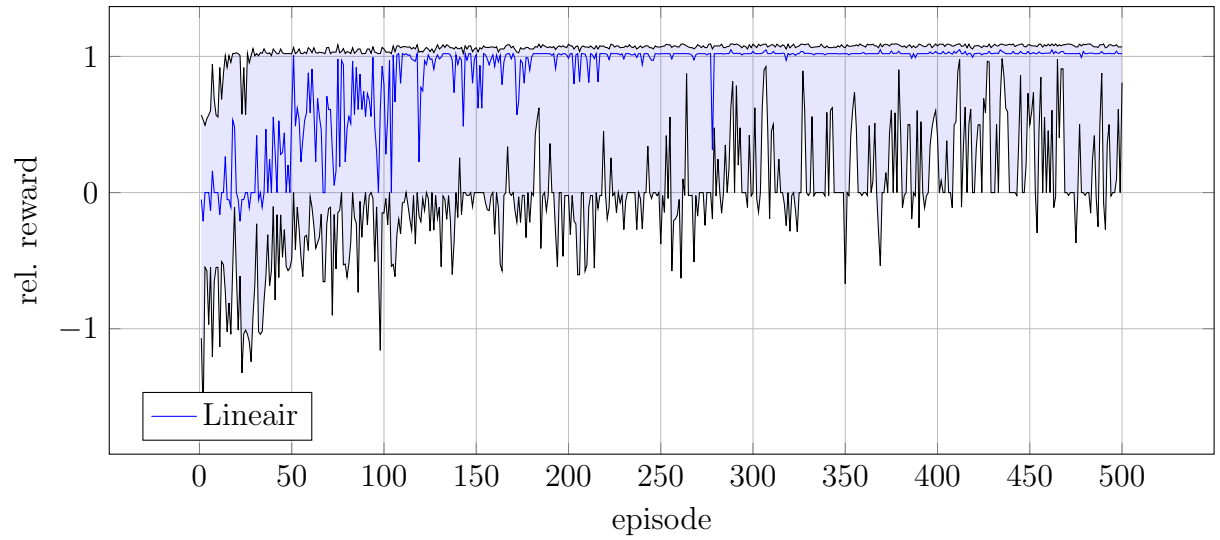
De gestructureerde gridworld wordt weergegeven in figuur 5.6. Het proces bestaat nu uit vier fasen. In iedere fase is er op één locatie een maximale reward geplaatst, de reward op andere locaties is omgekeerd evenredig met de afstand tot die maximale reward. Dit zorgt ervoor dat de rewards frequent genoeg beschikbaar zijn, zodat de exploratie van het probleem niet het grootste obstakel vormt. Wanneer de agent de maximale reward bereikt, wordt de maximale reward verplaatst en wordt de verdeling van de rewards op analoge wijze aangepast. Dit zorgt ervoor dat de agent zich in een andere richting moet verplaatsen en dat de optimale policy dus niet-lineair is. Daarnaast is er ook een *vermoeidheids*-factor: naast de vier bewegingsgrichtingen kan de agent nu ook kiezen om stil te staan. Bij iedere actie waarbij de agent beweegt, stijgt de vermoeidheid met 1. Wanneer de vermoeidheid 8 is, eindigt de episode en krijgt de agent een penalty. Als de agent stilstaat, daalt de vermoeidheid terug naar 0. Dit zorgt voor extra complexiteit in de policy.

De observatie van de agent bestaat uit een viertal (f, s, x, y), waarbij f de huidige vermoeidheid is (0-8), s de huidige fase (0-3) en x en y de coördinaten zijn van de agent op het rooster. De agent kan kiezen tussen 5 acties: bewegen naar links, rechts, boven of onder, of blijven staan.

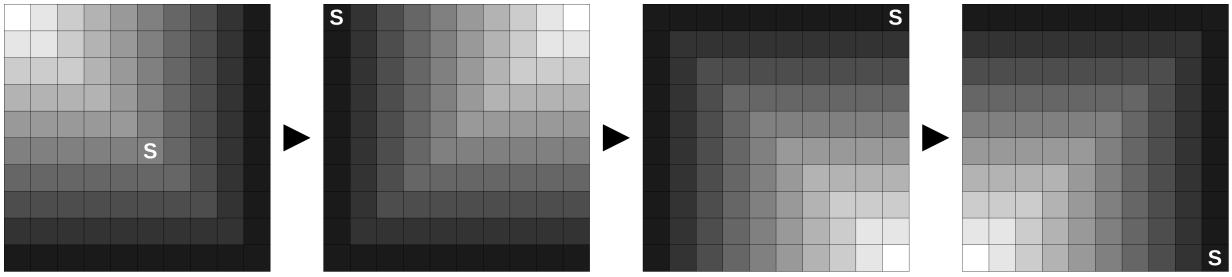
Op figuur 5.7 wordt distillatie van Dynamisch Programmeren naar ANFIS vergeleken met ANFIS Q-learning (hetzelfde algoritme als DQN waarbij het neuraal netwerk vervangen is door een ANFIS vaagregelaar met 20 vaagregels). De rode curve werd op analoge wijze



Figuur 5.4: Distillatie van Dynamisch Programmeren naar ANFIS voor 50 willekeurige gridworlds. De grafiek toont het 20e, 50e en 80e percentiel van de rewards. Rewards zijn in verhouding met de gemiddelde reward van een DP policy. De grote verschillen wijzen op sterke outliers in de data.



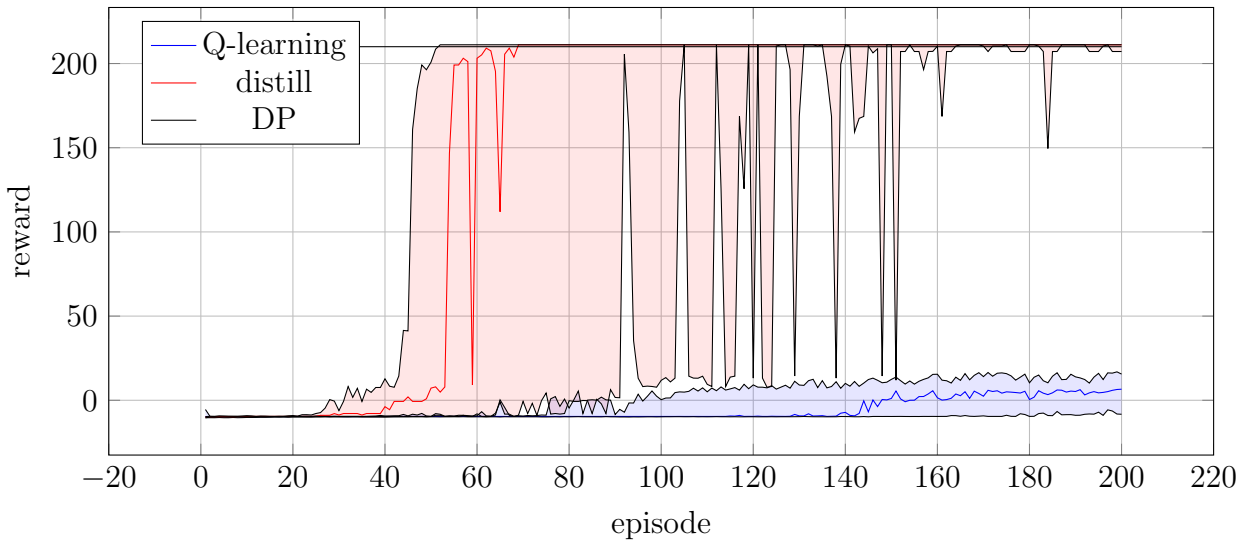
Figuur 5.5: Distillatie van Dynamisch Programmeren naar lineaire policy voor 50 willekeurige gridworlds.De grafiek toont het 20e, 50e en 80e percentiel van de rewards. Rewards zijn in verhouding met de gemiddelde reward van een DP policy. We zien dat een lineair model in de meeste gevallen reeds expressief genoeg is om de optimale policy te beschrijven, in veel gevallen zelfs beter dan ANFIS.



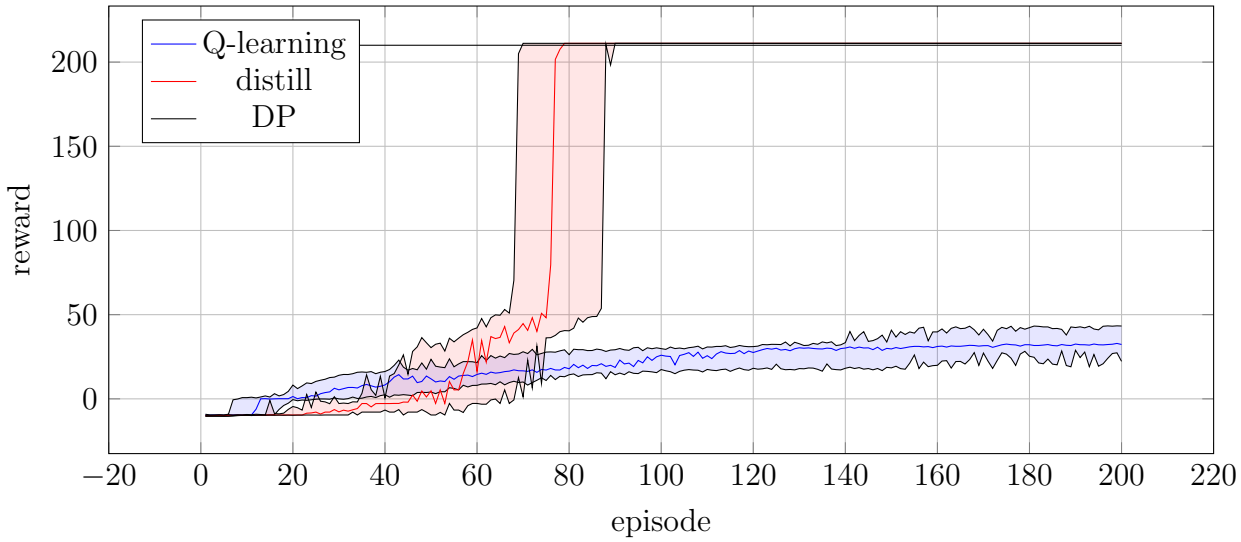
Figuur 5.6: Gestructureerde gridworld. Het proces bestaat uit vier fasen. De toestand aangeduid met "S" is de begintoestand van iedere fase. Wanneer de agent de hoogste reward in een bepaalde fase bereikt (de lichtst gekleurde cel), gaat het proces over naar de volgende fase. De episode eindigt wanneer 100 stappen genomen zijn, of wanneer de agent de laatste maximale reward bereikt.

opgesteld als bij de willekeurige gridworlds, door de optimale policy 50 keer te distilleren naar ANFIS en de mediaan, het 20e en het 80e percentiel van de reward per episode te plotten. Merk op dat het hier 50 keer om dezelfde optimale policy gaat, aangezien de gridworld telkens gelijk is, in tegenstelling tot de willekeurige griworlds waar 50 verschillende policies voor 50 verschillende gridworlds werden gedistilleerd. Daarnaast werd de blauwe curve opgesteld door ANFIS via Q-learning een policy te laten leren, en opnieuw de mediaan, het 20e en het 80e percentiel van de reward te plotten.

We zien duidelijk dat ANFIS via Q-learning niet capabel is om de optimale reward van 210 te bereiken (aangeduid door de lijn "DP" op de grafiek). Wanneer ANFIS via distillatie uit een Dynamisch Programmeren policy getraind wordt, lukt dit wel, en met slechts 13 regels. Figuur 5.8 geeft hetzelfde experiment weer voor een neuraal netwerk met 2 hidden layers van 64 nodes. We zien hier hetzelfde gedrag voorkomen. Aangezien beide modellen wel expressief genoeg zijn om de policy te encoderen, maar toch niet slagen via Q-learning, lijkt de beperkende factor van het probleem niet de representatie van de policy te zijn, maar eerder de exploratie van de gridworld. De kans dat de agent, wanneer hij de tweede fase bereikt, per toeval door de epsilon-greedy policy de juiste acties onderneemt is redelijk klein, waardoor de optimale policy niet ontdekt wordt.



Figuur 5.7: ANFIS Q-learning vs. distillatie van Dynamisch Programmeren naar ANFIS voor gestructureerde gridworld. Het experiment werd 50 keer uitgevoerd, de grafiek toont het 20e, 50e en 80e percentiel van de rewards.



Figuur 5.8: DQN vs. distillatie van Dynamisch Programmeren naar neuraal netwerk (2 hidden layers van 64 nodes) voor gestructureerde gridworld. Het experiment werd 50 keer uitgevoerd, de grafiek toont het 20e, 50e en 80e percentiel van de rewards.

OpenAI Gym Cartpole

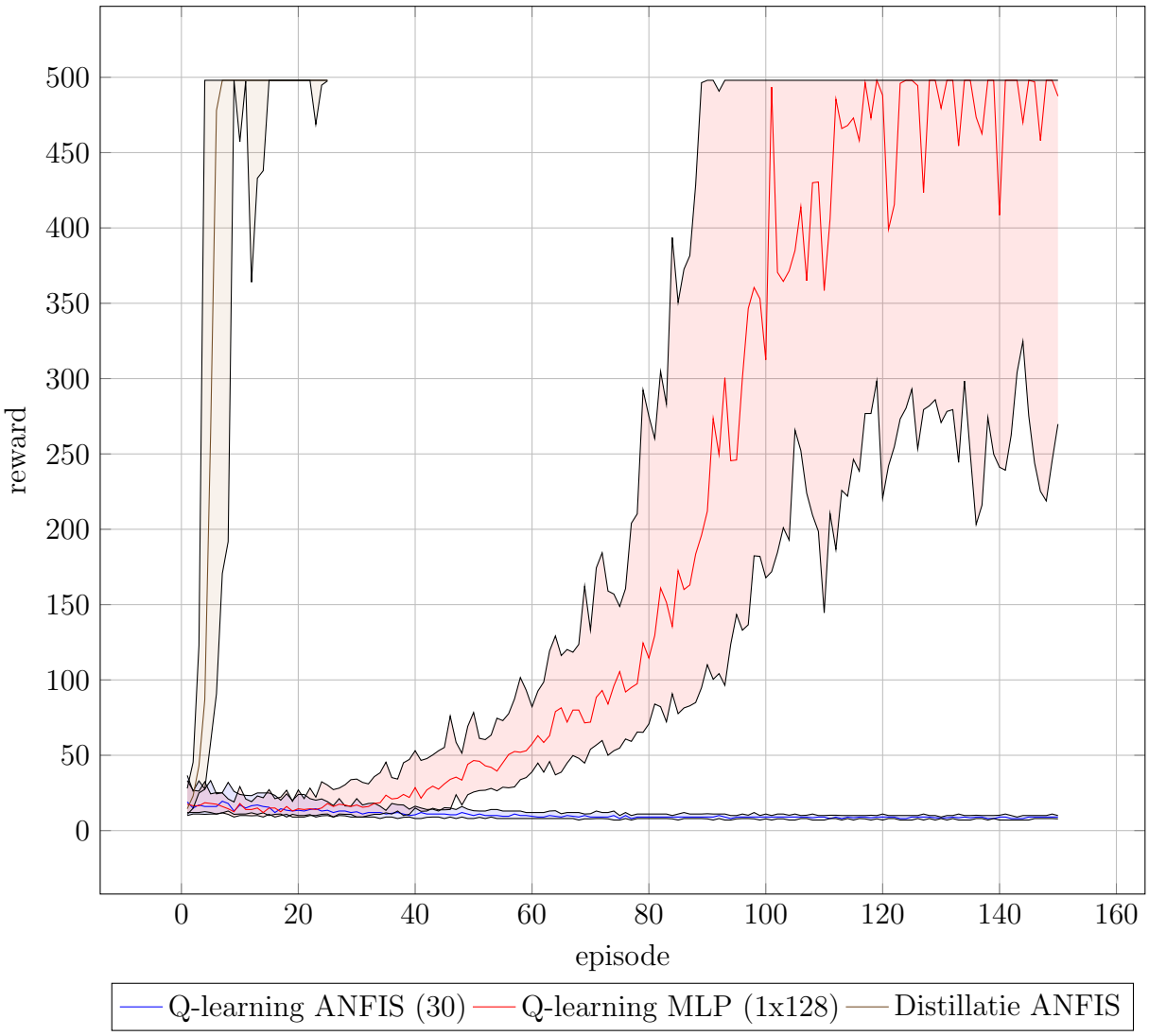
Ten slotte testen we het distillatie-algoritme op OpenAI Gym Cartpole (zie hoofdstuk 2.2 voor details). In tegenstelling tot de vorige experimenten is het hier niet haalbaar om dynamisch programmeren toe te passen, aangezien de invoerwaarden continu zijn. Daarom hebben we eerst DQN getraind met één hidden layer van 128 nodes, waarvan we de uitvoer gebruiken om te distilleren. We zien op figuur 5.9 dat ANFIS met 30 regels er via Q-learning niet in slaagt om het probleem op te lossen. DQN met een neuraal netwerk met één hidden layer van 128 neuronen daarentegen slaagt er wel in. Als we vervolgens DQN distilleren naar ANFIS, slaagt ook ANFIS erin. Wat hier merkwaardig is, is het kleine aantal regels dat ANFIS nodig heeft om de policy te encoderen: over de 50 runs van het experiment fluctueert het aantal gebruikte vaagregels tussen 3 en 6, terwijl er zeer weinig variatie is in de prestaties van het model. Daarnaast valt ook op dat het distillatie-algoritme zeer weinig episodes nodig heeft om te convergeren. In slechts 20 episodes slaagt ANFIS erin om de optimale policy aan te leren, in contrast met de 150 episodes DQN daarvoor nodig heeft.

5.2.3 Post-processing

Bij de vorige experimenten werd gebruik gemaakt van de ANFIS initialisatie via clustering, maar nog niet van de post-processing technieken. We passen nu beide technieken toe op de gestructureerde gridworld en op de OpenAI Gym Cartpole-environment.

Gestructureerde gridworld

De resultaten van beide post-processing-stappen voor de gestructureerde gridworld worden weergegeven in figuur 5.10. Op de linkerfiguur wordt de reward van de agent weergegeven voor een variërende thresholdwaarde α bij het samenvoegen van vaagverzamelingen in de antecedenten. Indien $\alpha = 1$, worden geen vaagverzamelingen samengevoegd, en verkrijgen we dus het oorspronkelijke model. We zien dat een zeer kleine wijziging in de vaagverzamelingen een zeer sterke invloed heeft op de performantie, aangezien voor $0.9 < \alpha < 1.0$ de



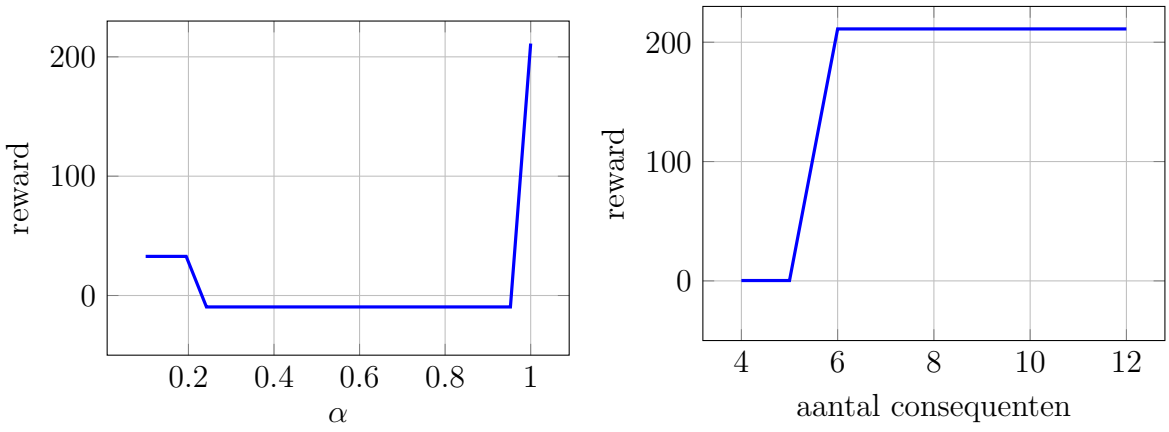
Figuur 5.9: DQN vs. ANFIS Q-learning vs. distillatie van DQN naar ANFIS voor OpenAI Gym Cartpole. Het experiment werd 50 keer uitgevoerd, de grafiek toont het 20e, 50e en 80e percentiel van de rewards.

reward al onder nul gezakt is. Dit betekent dat vaagverzamelingen die zeer sterk op mekaar lijken, toch een cruciaal onderscheid maken in de uitvoer, wat slechte gevolgen heeft voor de interpretabiliteit.

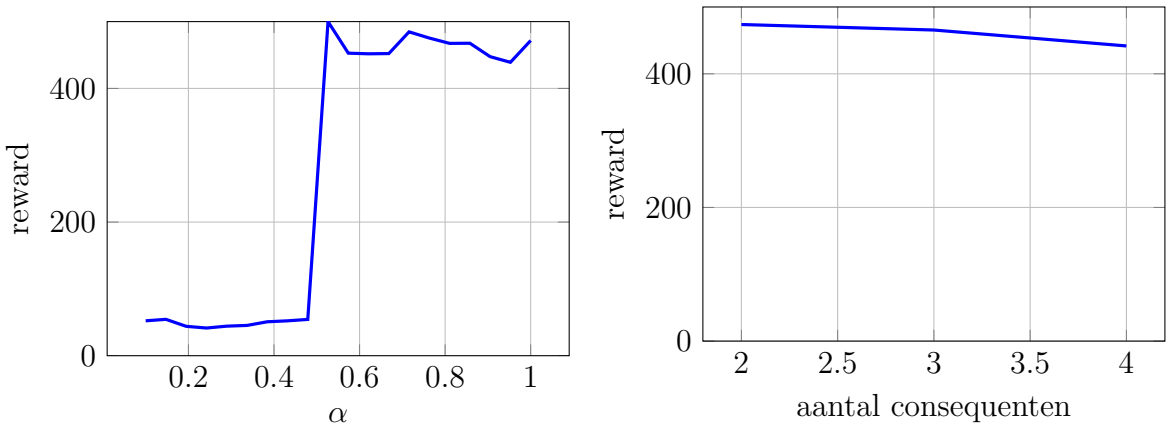
De rechterfiguur geeft de reward van de agent weer voor variërend aantal unieke consequenten. Het oorpsronkelijk model heeft 13 consequenten en het MDP heeft 4 acties, waardoor het niet zinvol is om boven de 12 of onder de 4 consequenten te meten. We zien dat we consequenten kunnen samenvoegen tot we er 6 overhouden zonder dat de performantie eronder lijdt. Onder de 6 consequenten neemt de performantie echter sterk af.

OpenAI Gym

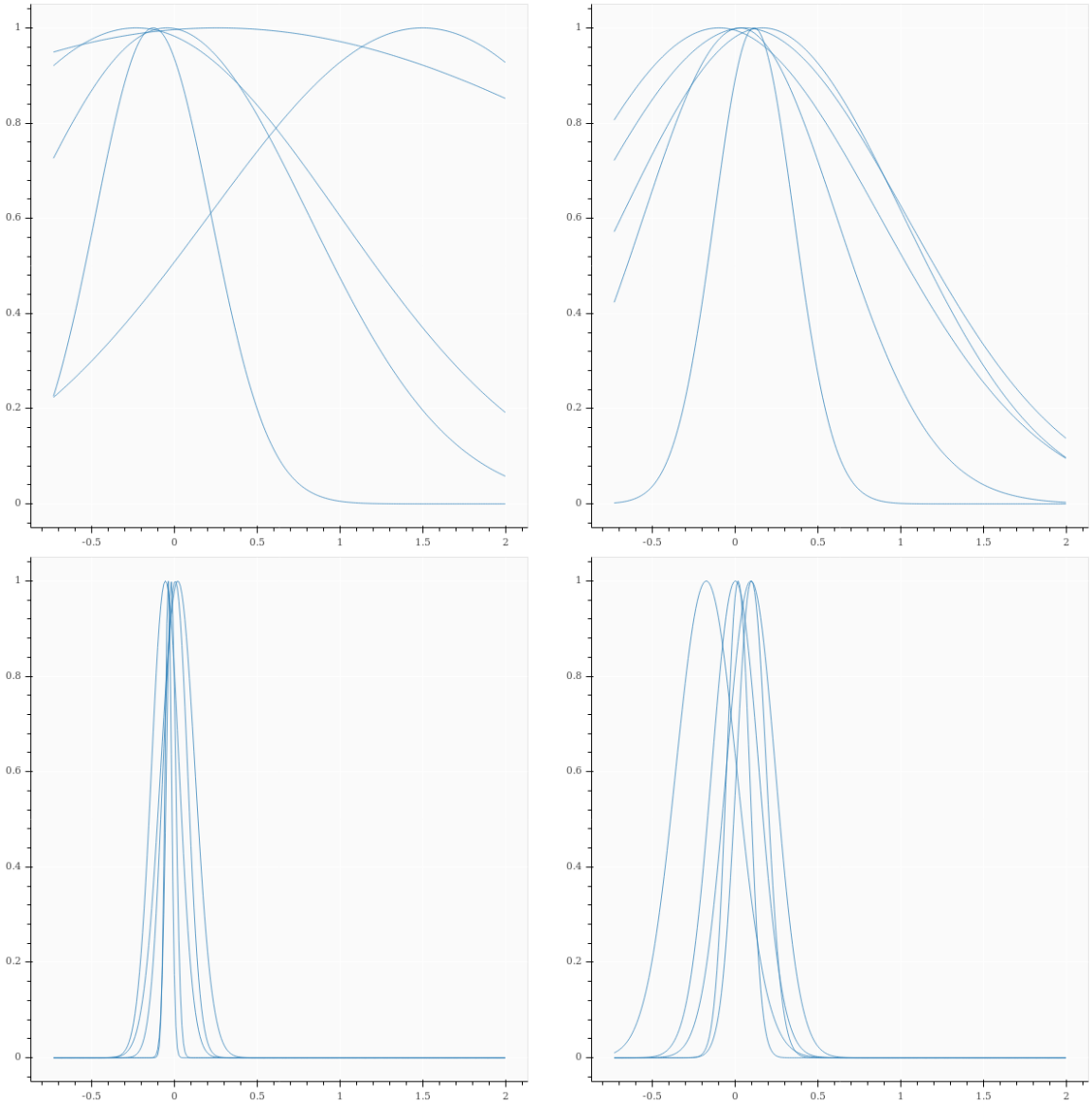
Figuur 5.11 geeft de resultaten weer van de post-processing-stappen voor Cartpole (analoog aan figuur 5.10). We vertrekken hierbij van een gedistilleerde ANFIS vaagregelaar met 5 regels. We zien hier een opmerkelijk verschil, vooral in de linkergrafiek die de invloed van het samenvoegen van vaagverzamelingen in de antecedenten weergeeft. Voor waarden van α groter dan 0.55 is de performantie amper aangetast. De rechterfiguur toont aan dat het aantal consequenten gereduceerd kan worden naar 2, wat het kleinste aantal consequenten is dat we redelijk kunnen verwachten aangezien de environment 2 acties heeft. Figuren 5.12 en 5.13 geven de vaagverzamelingen in de antecedenten weer van ANFIS resp. voor en na het samenvoegen van vaagverzamelingen met $\alpha = 0.55$. We zien een duidelijk verschil in complexiteit, met een reductie naar slechts 2 vaagverzamelingen voor de eerste drie invoerdimensies. De laatste invoerdimensie wordt minder gereduceerd, wat suggereert dat de uitvoer gevoeliger is aan veranderingen in deze dimensie dan in de andere.



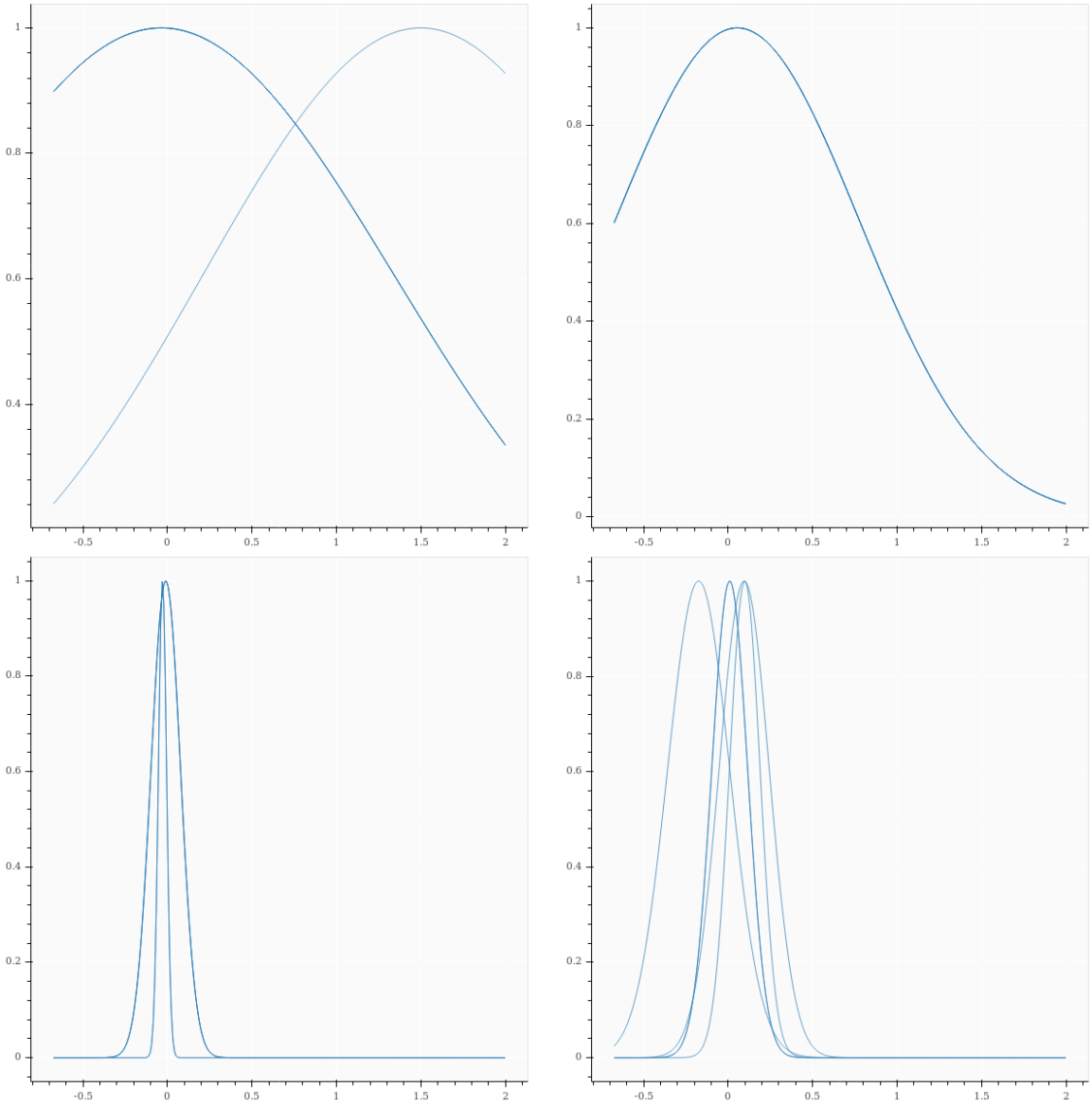
Figuur 5.10: Post-processing bij gestructureerde gridworld. Links: samenvoegen van vaagverzamelingen in de antecedenten. De grafiek toont de behaalde reward voor variërende waarden voor α. Rechts: samenvoegen van consequenten. De grafiek toont de behaalde reward voor een variërend aantal unieke consequenten.



Figuur 5.11: Post-processing bij Cartpole environment. Links: samenvoegen van antecedenten. De grafiek toont de behaalde reward voor variërende waarden voor α. Rechts: samenvoegen van consequenten. De grafiek toont de behaalde reward voor een variërend aantal unieke consequenten.



Figuur 5.12: Vaagverzamelingen in de antecedenten van ANFIS voor post-processing. Elke grafiek komt overeen met een invoerdimensie, elke curve met een vaagverzameling in die invoerdimensie. Hoewel het systeem slechts 5 vaagregels bevat, zien we veel complexiteit in de vaagverzamelingen, wat interpretatie moeilijk maakt.



Figuur 5.13: Vaagverzamelingen in de antecedenten van ANFIS na post-processing met $\alpha = 0.55$. Elke grafiek komt overeen met een invoerdimensie, elke curve met een vaagverzameling in die invoerdimensie. De eerste drie invoerdimensies zijn dramatisch afgenomen in complexiteit, wat interpretatie haalbaarder maakt. De laatste invoerdimensie blijft relatief complex.

Hoofdstuk 6

Conclusie en toekomstig werk

In hoofdstuk 2 van deze thesis hebben we een SVCCA [19], een bestaande techniek voor post-hoc interpretatie van neurale netwerken, toegepast op een multilayer perceptron getraind op de bekende Cartpole-environment. Het belangrijkste verschil hier was dat dit een reinforcement learning-context is, in tegenstelling tot de oorpsronkelijke resultaten die met SVCCA behaald werden, waar supervised learning gebruikt werd. Uit onze experimenten hebben we geconcludeerd dat bepaalde technieken die bij supervised learning wel werken, meer bepaald technieken voor compressie, bij reinforcement learning numeriek instabiel blijken te zijn. Dit wijst erop dat de representaties die door reinforcement learning agents aangeleerd worden fundamenteel verschillen van representaties in supervised learning, en dat post-hoc interpretatie van reinforcement learning agents dus een aangepaste aanpak vereist. Als toekomstig werk in deze richting kunnen we andere bestaande technieken voor post-hoc interpretatie toepassen op en/of uitbreiden naar reinforcement learning, om te zien of zich daar gelijkaardige problemen voordoen en hoe zulke problemen eventueel opgelost kunnen raken.

Als poging om het probleem van interpretabiliteit van reinforcement learning agents aan te pakken, hebben we in hoofdstuk 5 een techniek ontwikkeld om de kennis uit een neuraal netwerk over te brengen naar een meer interpreteerbaar ANFIS-model. Deze techniek is geïnspireerd op bestaand werk in knowledge distillation en policy distillation, in combinatie

met bestaande technieken om ANFIS zo interpreteerbaar mogelijk te houden. We kunnen uit deze experimenten concluderen dat ANFIS, in tegenstelling tot neurale netwerken, op zich niet flexibel genoeg is om rechtstreeks via Q-learning een policy aan te leren die tegelijk realistisch interpreteerbaar is, maar dat dit wel mogelijk is via distillatie uit een bestaande policy. Deze resultaten openen enkele nieuwe richtingen voor verder onderzoek:

- De bestaande literatuur over vaaglogica en vaaginferentiesystemen bevat veel heuristieke benaderingen om de performantie en interpretabiliteit van vaaginferentiesystemen verder te optimaliseren [69, 70] (bv. intelligentere manieren om vaagverzamelingen en vaagregels samen te voegen). Een groot deel van deze technieken werd in dit onderzoek nog niet geëxploreerd. Daarnaast kan er nog geëxperimenteerd worden met andere clusteralgoritmes voor de initialisatie van ANFIS, alternatieve vormen van lidmaatschapsfuncties, verschillenden t-(co-)normen, etc.

- De interpretabiliteit van ANFIS in dit werk werd vooral gemeten door het aantal unieke vaagverzamelingen en het aantal unieke consequenten in de vaagregels. Dit is echter geen perfecte metriek van interpretabiliteit. Zo kan een klein aantal vaagverzamelingen dat sterk overlapt bijvoorbeeld veel minder interpreteerbaar zijn dan een groter aantal vaagverzamelingen die onderling disjunct zijn. Er kan nog veel onderzoek verricht worden naar het kwantificeren van de interpretabiliteit van vaaginferentiesystemen, wat ook gepaard gaat met het beschrijven van zulke systemen in natuurlijke taal.

- Ten slotte is er verder onderzoek mogelijk naar uitbreidingen van dit distillatiealgoritme naar andere reinforcement learning-algoritmes, naast het DQN-algoritme dat hier gebruikt werd. Zo zou ook gedistilleerd kunnen worden vanuit DDPG [44], de uitbreiding van DQN naar continue actieruimtes. Het kan ook interessant zijn om te onderzoeken of ANFIS capabel is om beter te worden dan zijn teacher, door bv. het trainen van DQN stop te zetten nog voor de optimale policy bereikt is, en ANFIS na het distillatieproces via Q-learning te laten verder leren.

Bibliografie

[1] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [cs, stat]*, March 2015.

[2] Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy Distillation. *arXiv:1511.06295 [cs]*, November 2015.

[3] L. A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, 30(3):407–428, September 1975.

[4] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[5] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[6] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. page 60.

[7] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech Recognition with Deep Recurrent Neural Networks. *arXiv:1303.5778 [cs]*, March 2013.

[8] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A. Efros. Everybody Dance Now. *arXiv:1808.07371 [cs]*, August 2018.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*, September 2014.

[10] Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Scale-Aware Trident Networks for Object Detection. *arXiv preprint arXiv:1901.01892*, 2019.

[11] Seonghyeon Nam, Yunji Kim, and Seon Joo Kim. Text-Adaptive Generative Adversarial Networks: Manipulating Images with Natural Language. *arXiv:1810.11919 [cs]*, October 2018.

[12] Shauharda Khadka and Kagan Tumer. Evolution-Guided Policy Gradient in Reinforcement Learning. *arXiv:1805.07917 [cs, stat]*, May 2018.

[13] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer Publishing Company, Incorporated, 1st edition, 2017.

[14] Philippe Leray and Patrick Gallinari. Feature Selection with Neural Networks. *Behaviormetrika*, 26(1):145–166, 1999.

[15] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605, 1990.

[16] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, pages 293–299. IEEE, 1993.

[17] Mark Craven and Jude W Shavlik. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems*, pages 24–30, 1996.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013.

[19] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. page 17.

[20] Bin Liu, Ruiming Tang, Yingzhi Chen, Jinkai Yu, Huifeng Guo, and Yuzhou Zhang. Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction. *The World Wide Web Conference on - WWW '19*, pages 1119–1129, 2019.

[21] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep Interest Network for Click-Through Rate Prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1059–1068, New York, NY, USA, 2018. ACM.

[22] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. *arXiv:1610.03295 [cs, stat]*, October 2016.

[23] M. Kuderer, S. Gulati, and W. Burgard. Learning driving styles for autonomous vehicles from demonstration. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2641–2646, May 2015.

[24] Craig Saunders, Alexander Gammerman, and Volodya Vovk. Ridge regression learning algorithm in dual variables. 1998.

[25] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[26] S. P. K. Spielberg, R. B. Gopaluni, and P. D. Loewen. Deep reinforcement learning approaches for process control. In *2017 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP)*, pages 201–206, May 2017.

[27] Roger J Lewis. An introduction to classification and regression tree (CART) analysis. In *Annual Meeting of the Society for Academic Emergency Medicine in San Francisco, California*, volume 14, 2000.

[28] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomForest. *R news*, 2(3):18–22, 2002.

[29] R Bracewell. Heaviside's Unit Step Function. *The Fourier Transform and Its Applications*, pages 61–65, 2000.

[30] I Stephen. Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, 50(2):179, 1990.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[32] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the Expressive Power of Deep Neural Networks. *arXiv:1606.05336 [cs, stat]*, June 2016.

[33] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the Number of Linear Regions of Deep Neural Networks. page 9.

[34] Moshe Sniedovich. *Dynamic Programming: Foundations and Principles*. CRC press, 2010.

[35] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[36] Lawrence Davis. Handbook of genetic algorithms. 1991.

[37] Emile Aarts and Jan Korst. Simulated annealing and Boltzmann machines. 1988.

[38] Matthew D Zeiler. ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[39] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[40] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[41] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass, 1998.

[42] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[43] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[44] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, September 2015.

[45] David R. Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical Correlation Analysis: An Overview with Application to Learning Methods. *Neural Computation*, 16(12):2639–2664, December 2004.

[46] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. 2016.

[47] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv:1802.05668 [cs]*, February 2018.

[48] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, April 2017.

[49] Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. *arXiv:1711.10433 [cs]*, November 2017.

[50] Ying Zhang, Tao Xiang, Timothy M. Hospedales, and Huchuan Lu. Deep Mutual Learning. *arXiv:1706.00384 [cs]*, June 2017.

[51] Nicholas Frosst and Geoffrey Hinton. Distilling a Neural Network Into a Soft Decision Tree. *arXiv:1711.09784 [cs, stat]*, November 2017.

[52] Xuan Liu, Xiaoguang Wang, and Stan Matwin. Improving the Interpretability of Deep Neural Networks with Knowledge Distillation. *arXiv:1812.10924 [cs, stat]*, December 2018.

[53] Wojciech Marian Czarnecki, Siddhant M. Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Simon Osindero, Nicolas Heess, and Razvan Pascanu. Mix&Match - Agent Curricula for Reinforcement Learning. *arXiv:1806.01780 [cs, stat]*, June 2018.

[54] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *arXiv:1011.0686 [cs, stat]*, November 2010.

[55] Himani Arora, Rajath Kumar, Jason Krone, and Chong Li. Multi-task Learning for Continuous Control. *arXiv:1802.01034 [cs, stat]*, February 2018.

[56] Simon Schmitt, Jonathan J. Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, and S. M. Ali Eslami. Kickstarting Deep Reinforcement Learning. *arXiv:1803.03835 [cs]*, March 2018.

[57] Wojciech Marian Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant M. Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling Policy Distillation. *arXiv:1902.02186 [cs, stat]*, February 2019.

[58] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965.

[59] Chris Cornelis. Wiskundige Modellering van Artificiële Intelligentie, 2018.

[60] Aldo De Luca and Settimo Termini. A definition of a nonprobabilistic entropy in the setting of fuzzy sets theory. *Information and control*, 20(4):301–312, 1972.

[61] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, January 1975.

[62] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. In *Readings in Fuzzy Sets for Intelligent Systems*, pages 387–403. Elsevier, 1993.

[63] J.-S.R. Jang. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, May-June/1993.

[64] Hirofumi Miyajima, Noritaka Shigei, and Hiromi Miyajima. Approximation Capabilities of Interpretable Fuzzy Inference Systems. page 8, 2015.

[65] Rui Pedro Paiva and António Dourado. Interpretability and learning in neuro-fuzzy systems. 2004.

[66] Stephen L Chiu. Fuzzy model identification based on cluster estimation. *Journal of Intelligent & fuzzy systems*, 2(3):267–278, 1994.

[67] M. Setnes, R. Babuska, U. Kaymak, and H. R. van Nauta Lemke. Similarity measures in fuzzy rule base simplification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(3):376–386, June 1998.

[68] Saul Stahl. The evolution of the normal distribution. *Mathematics magazine*, 79(2):96–113, 2006.

[69] S. Guillaume. Designing fuzzy inference systems from data: An interpretability-oriented review. *IEEE Transactions on Fuzzy Systems*, 9(3):426–443, June 2001.

[70] Daniel Hein, Steffen Udluft, and Thomas A. Runkler. Generating Interpretable Fuzzy Controllers using Particle Swarm Optimization and Genetic Programming. *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '18*, pages 1268–1275, 2018.