Uitbreiden van een GAN met fuzzy logic voor doolhofgeneratie

Quinten Dewulf[1] en **Seth Jorissen**[1]
Begeleid door: Luc De Raedt en **Paolo Morettin**
[1]KULeuven
{quinten.dewulf, seth.jorissen}@student.kuleuven.be

Abstract

Een GAN (generative adversarial network) is een generatief model dat zich niet leent voor het maken van objecten die zekere constraints moeten naleven. Voor vele toepassingen is dit echter een noodzakelijke voorwaarde. In deze paper wordt een klassiek GAN vergeleken met een GAN uitgebreid met fuzzy logic omtrent de effectiviteit in het opleggen van gewenste constraints aan de geproduceerde uitvoer. Ook de diversiteit van geproduceerde objecten wordt in acht genomen. Dit alles werd getest a.d.h.v. het generen van doolhoven. De resultaten geven weer dat het uitgebreid GAN slaagt in het opleggen van tractable constraints (correct formatteren van een doolhof) met een aanvaardbaar verlies in diversiteit. Om de intractable constraints (oplosbaarheid) te verwerken in de fuzzy logic formule, poneert deze paper verschillende neurale netwerken. Oplosbaarheid is echter een complexe constraint en geen van de voorgestelde netwerken presteerde adequaat.

1 Inleiding

Een GAN (generative adversarial network) is een generatief model dat in theorie een onbekende kansverdeling kan vinden in zijn trainingsdata. In de praktijk is het echter niet gegarandeerd dat deze kansverdeling gelijk zal zijn aan de kansverdeling van de data [Goodfellow *et al.*, 2014]. Toch heeft het al veel successen gekend in het genereren van afbeeldingen, video, ...

Voor vele toepassingen wordt verwacht dat de gegenereerde data voldoet aan bepaalde constraints. Voorbeelden hiervan zijn het genereren van moleculen of video game levels [Di Liello *et al.*, 2020]. Hierbij volstaat het niet dat gegenereerde objecten enkel visueel lijken op de trainingsvoorbeelden, want dan wordt niet gegarandeerd dat de constraints voldaan zijn. Het voorbeeld dat in deze paper gebruikt wordt, is het genereren van doolhoven. Een traditioneel GAN die getraind wordt met correcte doolhoven, genereert doolhoven die visueel lijken op de trainingsdata. Het probleem is dat deze doolhoven vaak onoplosbaar of incorrect geformatteerd zijn. Een traditioneel GAN is hier dus onvoldoende.

Deze paper stelt een oplossing voor door een klassiek GAN uit te breiden met fuzzy logic. Deze uitbreiding zorgt ervoor dat de generator gestraft wordt voor het genereren van data die niet aan de constraints voldoet. Hierdoor zal de generator leren om enkel data te genereren die aan alle constraints voldoet. Zo wordt gegarandeerd dat de bekomen kansverdeling lijkt op die van de data en dat de constraints voldaan zijn. Een bijkomend probleem is echter dat complexe constriants niet geschreven kunnen worden in fuzzy logic. In deze paper worden deze complexe constraints voorgesteld door een predikaat en wordt er naar mogelijke implementaties gezocht.

2 Klassiek GAN

Een GAN bestaat uit twee grote onderdelen: de generator G en de discriminator D. De objecten die men wenst te verkrijgen behoren tot de data space X. Ook is er een latent space Z waartoe de input voor G behoord. De trainingsdata voor het GAN is afkomstig van de gewenste verdeling P_{data} over X. De generator $G : Z \to X : \mathbf{z} \mapsto G(\mathbf{z}; \theta_g)$ beeldt willekeurige noise vectoren uit Z af op elementen uit de data space. De invoer voor de generator volgt de verdeling $p_\mathbf{z}$, dit betekent dat de uitvoer van de generator verdeeld is volgens P_g, met $G(\mathbf{z}; \theta_g) \sim P_g$ als $\mathbf{z} \sim p_\mathbf{z}$. G is goed getraind wanneer de parameters θ_g zo gekozen zijn zodat P_g een accurate benadering is van P_{data}. De discriminator $D : X \to [0, 1] : \mathbf{x} \mapsto D(\mathbf{x}; \theta_d)$ beeldt objecten uit de data space af op elementen uit het interval $[0, 1]$. D is goed getraind wanneer het een lage score geeft aan objecten die met hoge waarschijnlijkheid niet afkomstig zijn van P_{data} en een hoge score geeft aan objecten die dat wel zijn. [Goodfellow *et al.*, 2014]

Het trainingsproces van een GAN bestaat uit het afwisselend trainen van de initieel ongetrainde generator en discriminator. Concreet zal afwisselend de discriminator de waardefunctie V proberen te maximaliseren en zal de generator deze proberen te minimaliseren door hun respectievelijke parameters te wijzigen.

$$V(G, D) =$$

$$\mathbb{E}_{\mathbf{x} \sim P_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

$$\min_G \max_D V(G, D)$$

3 Uitgebreid GAN

In deze sectie wordt er een uitgebreid GAN voorgesteld, gericht op het inrekening brengen van gewenste constraints tijdens de trainingsfase. De uitbreiding bestaat uit een toevoeging van de fuzzy loss term aan de waardefunctie van een GAN. Deze term gebruikt een fuzzy logic formule om te berekenen hoe sterk een door de generator geproduceerd object aan de gewenste constraints voldoet.

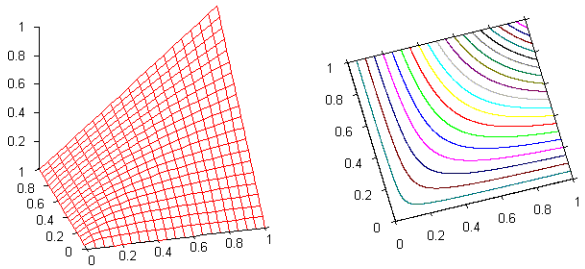
Er wordt aangenomen dat de data space X een discrete ruimte is. Laat X' de continue uitbreiding van X zijn, de generator G wordt dan getraind om continue benaderingen ($\in X'$) te maken van bepaalde objecten in X. Het domein van de discriminator zal dan ook X' zijn. Eenmaal het trainingsproces gedaan is zal er nog een additionele discretisatiestap worden uitgevoerd op de output van G, zodat er uiteindelijk toch objecten van X worden geproduceerd.

3.1 Fuzzy Logic

Fuzzy logic is een vorm van logica die niet met 'harde' waarden zoals waar en onwaar werkt. In de plaats daarvan hebben variabelen een waarde in het interval $[0, 1]$, hoe hoger de waarde van een variable hoe meer deze waar is, hoe dichter de waarde bij 0 komt hoe meer deze onwaar is. Er zijn verschillende fuzzy logic systemen, de meeste worden vastgelegd door een t-norm. Een t-norm is het fuzzy logic equivalent van de logische conjunctie. Alle andere logische operatoren worden geconstrueerd aan de hand van deze conjunctie en de negatie. [Diligenti *et al.*, 2017]

Een voorbeeld van zo'n t-norm is de product t-norm. Deze heeft voor de invoer van variabelen a en b met $a, b \in [0, 1]$ het product $a.b$ als uitvoer. De Hamacher t-norm is dan weer een familie van verschillende t-normen en brengt daardoor verschillende fuzzy logics voort.

$$T_p^H(x, y) = \begin{cases} 0 & als\ x = y = p = 0 \\ \frac{xy}{p+(1-p)(x+y-xy)} & overig \end{cases}$$



Figuur 1: De formule voor de Hamacher t-norm met de grafiek voor $p = 0$

3.2 Correctheidsformule fl

De correctheidsformule fl met $fl : X' \to [0, 1] : \mathbf{x} \mapsto fl(\mathbf{x})$ berekent hoe sterk een gegenereerd object aan alle gewenste constraints voldoet. fl is een fuzzy logic formule van de vorm: $constraint\_1(\mathbf{x}) \wedge constraint\_2(\mathbf{x}) \wedge constraint\_3(\mathbf{x}) \wedge \ldots$ Hierbij is $constraint\_i : X' \to [0, 1] : \mathbf{x} \mapsto constraint\_i(\mathbf{x})$ een functie die een geproduceerd object afbeeldt op een maat voor het naleven van constraint i. Een functiewaarde van 1 betekent dat object $\mathbf{x}$ volledig aan het constraint voldoet. fl bevat voor elk gewenst constraint dergelijke term.

De i^{de} constraint is een tractable constraint, wanneer $constraint\_i$ een fuzzy logic formule is over de elementen van vector $\mathbf{x}$. $constraint\_i$ is dan simpelweg de logische formule die deze beperking omschrijft. Als constraint i echter een complexe constraint is, die niet neergeschreven kan worden in een booleaanse formule van redelijke lengte, dan beschouwen we $constraint\_i$ als een fuzzy logic predikaat. Dit predikaat is eventueel geïmplementeerd door een vooraf getraind neuraal netwerk.

3.3 Fuzzy Loss

Fuzzy loss wordt geïntroduceerd als:

$$\mathbb{E}_{\mathbf{z}\sim p_z}[-\log fl(G(\mathbf{z}))]$$

De grootte van de fuzzy loss hangt af van de correctheid van de gegenereerde objecten en behoort steeds tot het interval $[0, +\infty[$. Als de verwachtte overeenkomst van de gegenereerde objecten met de constraints zeer hoog is zal fuzzy loss een kleine waarde in de buurt van 0 hebben. Als de verwachtte correctheidswaarde echter zeer laag is zal fuzzy loss een extreem hoge waarde hebben ($+\infty$ als de verwachtte correctheidswaarde 0 is).

3.4 Uitgebreide Waardefunctie

De waardefunctie van het uitgebreid GAN bestaat uit de waardefunctie van een klassiek GAN uitgebreid met de fuzzy loss. Aangezien de generator G de waarde van $V_{uitgebreid}$ zal willen minimaliseren, zal G ook de fuzzy loss willen minimaliseren. De discriminator kan door zijn parameters aan te passen niets wijzigen aan de waarde van de fuzzy loss. Omdat G de fuzzy loss zal minimaliseren, zal G de verwachtte correctheidswaarde van de gegenereerde objecten maximaliseren. De hyperparameter $k \in [0, +\infty[$ wordt ingevoerd om de grootte van de fuzzy loss te balanceren t.o.v. de grootte van de adversarial loss. Deze uitbreiding is geïnspireerd door CAN's. [Di Liello *et al.*, 2020]

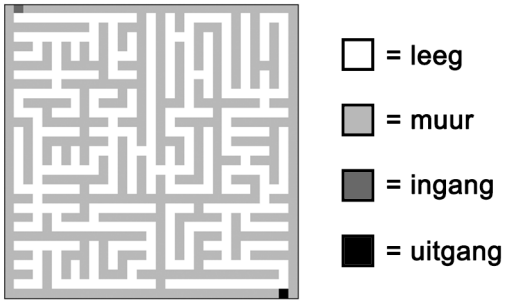
$$V_{uitgebreid}(G, D) = \mathbb{E}_{\mathbf{x}\sim P_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))] + k\,.\,\mathbb{E}_{\mathbf{z}\sim p_z}[-\log fl(G(\mathbf{z}))]$$

4 Probleemstelling

In deze paper wordt de geschiktheid van het voorgestelde uitgebreid GAN voor het genereren van doolhoven onderzocht. Er worden twee tractable constraints en één intractable constraint gedefinieerd.

T1: Het level is volledig ommuurd

T2: Er is exact één ingang en uitgang op de correcte positie



Figuur 2: Een doolhof afkomstig uit de trainingsdata met legende

Oplosbaarheid: Het level is oplosbaar

Volgende onderzoeksvragen worden behandeld:

OV1: Kan de kans dat een gegenereerd doolhof aan de gewenste tractable constraints voldoet aanzienlijk vergroot worden door het gebruik van het voorgestelde uitgebreid GAN i.p.v. een klassiek GAN?

Het uitgebreid GAN zal hier dus enkel actief proberen de tractable constraints T1 en T2 op te leggen aan zijn output. De hypothese is dat het uitgebreid GAN een hogere correctheidswaarde zal hebben dan een traditioneel GAN. Hierbij is de correctheid het aandeel van de gegenereerde levels die aan de constraints voldoen.

De oplosbaarhied is een zeer complexe constraint die niet tractable is. Er is dus nood aan een oplosbaarheidspredikaat om te gebruiken in de formule fl. Dit geeft de tweede onderzoeksvraag:

OV2: Bestaat er een accurate implementatie voor het oplosbaarheidspredikaat dat gebruikt kan worden in de fuzzy logic formule fl?

Dit betekent dat deze implementatie een afleidbare functie moet zijn die een gegenereerd level afbeeldt op het interval [0,1], waarbij de functiewaarde een accurate schatting is voor de oplosbaarheid van dat level. De hypothese is dat dergelijke implementatie bestaat en de paper zal dit evalueren door te proberen om het op te stellen.

Indien een accurate implementatie voor het oplosbaarheidspredikaat bestaat, kan dit toegevoegd worden aan fl. Hieruit volgt de finale onderzoeksvraag:

OV3: Kan de kans dat een gegenereerd doolhof aan de gewenste tractable en intractable constraints voldoet aanzienlijk vergroot worden door gebruikt te maken van het voorgestelde uitgebreid GAN?

5 Methode

Het klassiek en voorgesteld uitgebreid GAN worden beiden geïmplementeerd m.b.v. PyTorch. Het gebruik van fuzzy logic in combinatie met PyTorch wordt gefaciliteerd door het uitbreiden van torch.autograd (https://github.com/Quinten-D/WV_GAN_uitgebreid_met_fuzzy_logic). Voor de experimenten wordt de fuzzy logic op basis van de Hamacher t-norm gebruikt. De waarde p is de hyperparameter die de t-norm bepaald en kan makkelijk aangepast worden om verschillende t-normen te testen. Na het trainingsproces wordt de geschiktheid van beide modellen voor doolhofgeneratie nagegaan a.h.v. verschillende testen.

Trainingsdata

De trainingsdata is gegenereerd door een procedureel algoritme (verdeel en heers) en bestaat uit 10 000 levels. Ieder doolhof is een rooster van 31x31 tegels waarbij iedere tegel is gecodeerd a.d.h.v. one-hot codering. Aangezien er vier mogelijkheden zijn voor elke tegel (leeg, muur, ingang of uitgang), wordt ieder level voorgesteld als een vector met 3844 elementen. De eerste vier opeenvolgende elementen vormen de one-hot codering van de eerste tegel, de tweede vier opeenvolgende elementen vormen deze voor de tweede tegel, ...

Architectuur GAN

De generator en discriminator worden beiden geïmplementeerd door een multilayer perceptron (MLP). Zowel het klassiek als uitgebreid GAN maken gebruik van deze architectuur. Het enige verschil tussen de twee zal bestaan uit het feit dat het klassiek GAN getraind wordt met de traditionele waardefunctie en het uitgebreid GAN met de uitgebreide waardefunctie.

Netwerk	invoer→uitvoer	Operatie
G	100→600	nn.Linear met ReLu
	600→2000	nn.Linear met ReLu
	2000→1600	nn.Linear met ReLu
	1600→3844	nn.Linear
	3844→(961,4)	reshape()
	(961,4)→(961,4)	log_softmax()
	(961,4)→(961,4)	gumbel_softmax()
	(961,4)→3844	reshape()
D	3844→600	nn.Linear met LeakyReLu
	600→600	nn.Linear met LeakyReLu
	600→400	nn.Linear met LeakyReLu
	400→1	nn.Linear met Sigmoid

Tabel 1: Architectuur van de generator en de discriminator
Hierbij is nn.Linear de PyTorch functie voor een 'fully connected layer' in een neuraal netwerk

De Gumbel-Softmax stap van de generator G heeft als doel om een kansverdeling om te zetten in een continue benadering van een one-hot vector. Ook zal de 'straight-through trick' toegepast worden bij het genereren van doolhoven door G. Dit wil zeggen dat wanneer een gegenereerd object niet afleidbaar moet zijn naar de gewichten van G er nog een bijkomende discretisatiestap (argmax) wordt toegepast op het object. Wanneer dergelijk object wel afleidbaar moet zijn naar de gewichten van G wordt deze stap weggelaten. [Jang *et al.*, 2016] [Wong, 2020]

Constraints

De afbeelding die een geproduceerd doolhof afbeeldt op de maat van het naleven van de tractable constraint T1 is $form\_1$. $form\_1$ is de fuzzy logic formule die constraint T1 omschrijft. Op dezelfde manier is $form\_2$ de afbeelding

die doolhoven afbeeldt op de maat van het naleven van T2 en is het de fuzzy logic formule die deze constraint omschrijft.

Het oplosbaarheidspredikaat is de afbeelding die de correctheid van doolhoven volgens het constraint 'oplosbaarheid' nagaat. Strikt genomen zou zo'n predikaat dus oplosbare doolhoven afbeelden op 1 en onoplosbare op 0. Dit is echter niet geschikt voor gebruik in de correctheidsformule fl, wegens het feit dat dergelijk predikaat niet afleidbaar is naar de componenten van zijn inputvector. We beschouwen het oplosbaarheidspredikaat Opl als een afbeelding die oplosbare doolhoven een hoge score geeft (≈ 1), niet oplosbare doolhoven een lage score geeft (≈ 0) en afleidbaar is. Dit oplosbaarheidspredikaat wordt geïmplementeerd door een neuraal netwerk. In deze paper worden verschillen mogelijkheden getest.

De trainingsdata voor deze neurale netwerken bestaat uit een level met een target. De levels zijn gegenereerd door het controle GAN op verschillende momenten in het trainingsproces. De target hangt af van soort neuraal netwerk dat gebruikt wordt. Enerzijds is er een netwerk voor $classification$, waarbij de target één waarde is die aangeeft of het level al dan niet oplosbaar is. Anderzijds is er een netwerk voor $multi\text{–}label\_classification$, waarbij de target de bereikbaarheidsmap is van het level. De bereikbaarheidsmap is een vector die voor iedere tegel een waarde bijhoud. Deze waarde geeft aan of die tegel al dan niet bereikbaar is. De oplosbaarheid wordt dan gegeven door de bereikbaarheid van de uitgangtegel.

Trainen

Elk onderzocht GAN wordt getraind voor 15 epochs en 5 keer per epoch worden de gewichten van de generator opgeslagen. Het trainingsalgoritme van een klassiek GAN is identiek aan dat voor een uitgebreid GAN op het blauwe gedeeldte na. Als optimizer werd de Adam optimizer gekozen. [Pytorch, 2019]

Eerst werd het controle GAN getraind. Dit is een klassiek getraind GAN dat voor alle onderzochte waarden van de hyperparameters de beste resultaten gaf. Doorheen de experimenten wordt het controle GAN als een baseline gebruikt. Alle uitgebreide GAN's die gebruikt worden in de experimenten hebben dezelfde waarden als het controle GAN voor overeenkomstige hyperparameters. De gebruikte batch grootte voor het controle GAN en dus ook de overige GAN's is 40.

Testen

Het evalueren van een GAN gebeurt aan de hand van testen. Voor de verschillende gewichten van de generator worden 400 levels gegenereerd die vervolgens onderworpen worden aan twee soorten testen:

De constraints test geeft de correctheidswaarde weer. Dit is de verhouding van het aantal levels dat voldoet aan de opgelegde constraints t.o.v. het totaal aantal gegenereerde levels.

De L1-norm test is een maat voor de diversiteit van de gegenereerde doolhoven. Voor deze test wordt per 40 levels de L1-norm (Manhattan afstand) berekend van elk paar levels en wordt het gemiddelde genomen van alle bekomen L1-normen. Deze test wordt gebruikt om 'mode collapse' op te sporen. Dit is het verschijnsel waarbij de generator slechts een heel beperkt aantal verschillende doolhoven genereert. In dit geval is de diversiteit van de gegenereerde doolhoven extreem laag, dit wordt weerspiegeld in een lage L1-norm waarde.

Algorithm 1 Trainingsalgoritme voor het uitgebreid GAN

Input: Een discriminator D en generator G, de trainingsvoorbeelden training_data, de batch grootte batch_grootte, de correctheidsformule fl en het gekozen aantal epochs aantal_epochs

Output: De gewichten van de generator G op verschillende momenten in het trainingsproces

aantal_updates = 0
m = batch_grootte
for epoch in aantal_epochs **do**
  for batch in training_data **do**

    /* update de discriminator */
    Neem m noise samples $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ (batch = $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$)
    Bereken de stochastic gradient waarbij er wordt afgeleid naar de gewichten van de discriminator:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [-\log D(\mathbf{x}^{(i)})]$$
$$+ \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [-\log(1 - D(\mathbf{z}^{(i)}))]$$

    Laat de optimizer de gewichten van de discriminator θ_d updaten met behulp van de berekende gradient om te trachten de waarde van $V_{uitgebreid}$ te verhogen

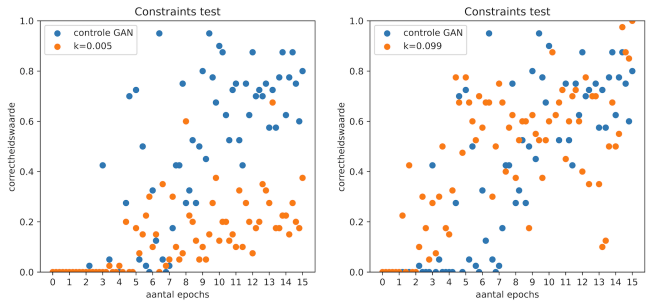
    /* update de generator */
    Neem m noise samples $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$
    Bereken de stochastic gradient waarbij er wordt afgeleid naar de gewichten van de generator:

$$\nabla_{\theta_g} [\frac{1}{m} \sum_{i=1}^{m} [-\log D(G(\mathbf{z}^{(i)}))]$$
$$+k. \frac{1}{m} \sum_{i=1}^{m} [-\log fl(G(\mathbf{z}^{(i)}))]]$$

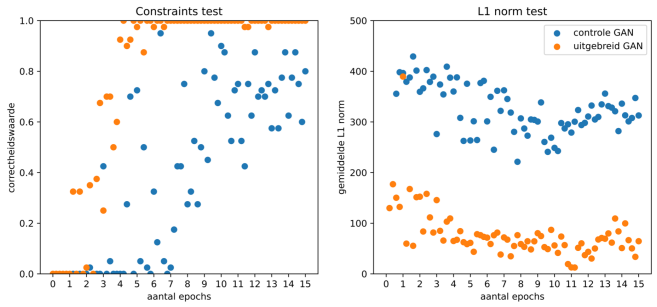
    Laat de optimizer de gewichten van de generator θ_g updaten met behulp van de berekende gradient om te trachten de waarde van $V_{uitgebreid}$ te verlagen

    /* sla elke 50 updates de gewichten van G op */
    aantal_updates += 1
    if aantal_updates $\% 50 == 0$ **then**
      Sla θ_g op
    end if

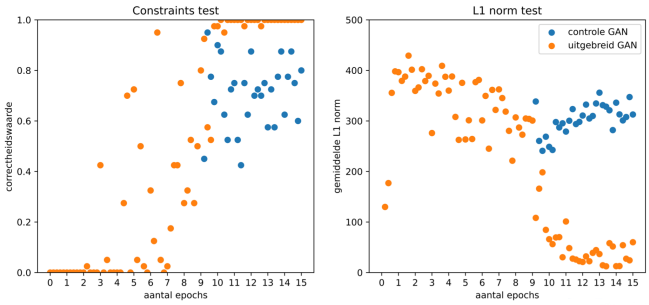
  end for
end for



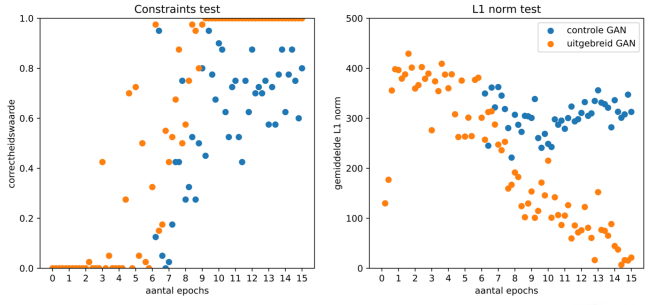
Figuur 3: De constraints test (voor constraints T1 en T2) toegepast op het controle GAN en het uitgebreid GAN voor $k = 0.05$ en $k = 0.099$



Figuur 4: Het uitgebreid GAN met $k = 0.1$ en het controle GAN geëvalueerd aan de hand van de constraints test (voor constraints T1 en T2) en de L1-norm test



Figuur 5: Het uitgebreid GAN met k als stapfunctie en het controle GAN geëvalueerd met de constraints test (voor constraints T1 en T2) en de L1-norm test



Figuur 6: Het uitgebreid GAN met linear toenemende k en het controle GAN geëvalueerd aan de hand van de constraints test (voor constraints T1 en T2) en de L1-norm test

6 Resultaten

6.1 Onderzoeksvraag 1

Deze resultaten evalueren de hypothese bij de eerste onderzoeksvraag. Omdat hier enkel wordt onderzocht of het uitgebreid GAN de tractable constraints kan opleggen wordt de correctheidsformule gedefinieerd als: $fl(\mathbf{x}) = form\_1(\mathbf{x}) \wedge form\_2(\mathbf{x})$.

Er werd geëxperimenteerd met de hyperparameter k die is ingevoerd in de uitgebreide waardefunctie. Alle andere hyperparameters zijn gelijk aan de hyperparameters van het controle GAN. De fuzzy logic die gebruikt is, is de Hamacher t-norm met $p = 0$.

Statische k

Statische k betekent dat gedurende het hele trainingsproces k gelijk is aan een constante. Er zijn veel waarden voor k getest en voor de meeste waarden is er geen substantiële verbetering in de correctheidswaarde t.o.v. het controle GAN. Soms waren de resultaten van het uitgebreid GAN, tegen de verwachingen in, slechter dan die van het controle GAN, bv. bij $k = 0,05$ (zie Figuur 2).

In een paar gevallen, bv $k = 0,1$, of bij hoge k, waren de resultaten voor de correctheidswaarden perfect, op een paar waarden na, vanaf een bepaalde epoch. Het nadeel is dat voor al deze waardes voor k de diversiteit ook zeer laag is (zie Figuur 3). Het lijkt puur toeval dat dit het geval is bij $k = 0,1$, want ook bij iets hogere waarden is er geen verbetering t.o.v. het controle GAN. Zelfs het GAN met $k = 0,099$ komt niet in de buurt van de goede correctheidswaarde van de GAN met $k = 0,1$.

De conclusie is dat bij een statische k de correctheidswaarde zeer volatiel is en indien de correctheidswaarde hoog is, is er zeer weinig diversiteit.

k als stapfunctie

Met k als stapfunctie wordt bedoeld dat k eerst gelijk is aan 0 en dat vanaf een bepaalde epoch, k wordt gelijkgesteld aan een bepaalde constante. Er is een opvallend resultaat bij

$$k = \begin{cases} 0 & epoch[0,9) \\ 0.05 & epoch[9,15] \end{cases}$$

Alhoewel een statische $k = 0.05$ een slecht resultaat gaf voor de correctheidswaarde, stijgt de correctheidswaarde bij de stapfunctie zeer snel t.o.v. het controle GAN, vanaf epoch 9 en is die zelfs perfect, op een paar gevallen na, vanaf epoch 10. Het lijkt dat de fuzzy loss beter informatie kan geven aan de generator indien de levels al beter op doolhoven lijken. De diversiteit is wel nog steeds een nadeel: vanaf epoch 9 zakt de diversiteit onmiddellijk naar een veel lager niveau (zie Figuur 4). Er zijn ook nog andere stapfuncties getest en deze gaven allemaal gelijkaardige resultaten.

De conclusie is dat bij een k als stapfunctie de correctheidswaarde in het algemeen perfect is kort na de stap, maar dat de diversiteit dan opnieuw zeer laag is.

Dynamische k

Dynamische k betekent dat k eerst gelijk is aan 0 en dat vanaf een bepaalde epoch, k lineair stijgt met het aantal epochs.

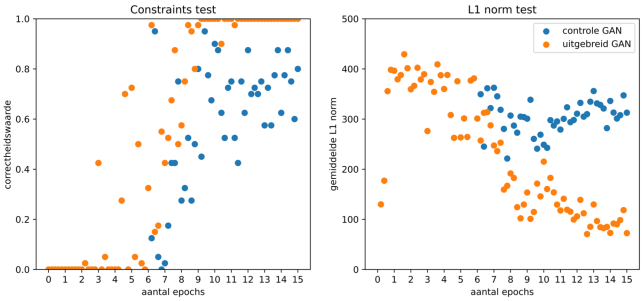
Een voorbeeld hiervan is

$$k = \begin{cases} 0 & epoch[0,6) \\ (epoch-5)*0.0075 & epoch[6,15] \end{cases}$$

De correctheidswaarde is hier zeer gelijkaardig als bij k als stapfunctie. Het verschil is dat de correctheidswaarde minder snel stijgt t.o.v. het controle GAN, maar zal wel nog altijd de perfecte waarden bereiken. Het grote voordeel is dat de diversiteit hier niet onmiddelijk daalt, maar lineair daalt met het aantal epochs. Dit zorgt voor een zone waar de diversiteit nog steeds vrij hoog is en er toch al een perfecte correctheidswaarde bereikt is (zie Figuur 5).

Een poging om er voor te zorgen dat deze zone zo groot mogelijk is, is door k constant te houden vanaf een bepaalde epoch. Een voorbeeld hiervan is

$$k = \begin{cases} 0 & epoch[0,6) \\ (epoch-5)*0.0075 & epoch[6,10) \\ 0.03 & epoch[10,15] \end{cases}$$



Figuur 7: Het uitgebreid GAN waarbij k eerst lineair toeneemt en daarna constant blijft en het controle GAN geëvalueerd aan de hand van de constraints test (voor constraints T1 en T2) en de L1-norm test

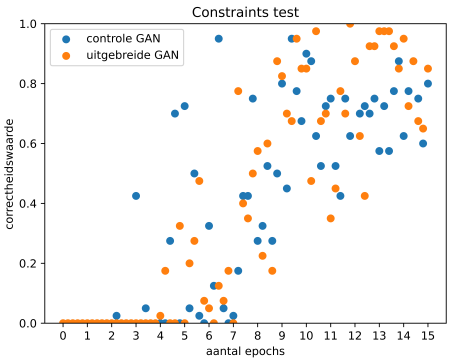
De resultaten gaven inderdaad aan dat vanaf k constant is dat daling in diversiteit begint af te vlakken. Voor de correctheidswaarde is er geen verschil tussen de twee functies (zie Figuur 6).

De conlusie is dat voor dynamische k er een zone is waar de diversiteit vrij hoog is en waar de correctheidswaarde perfect is en dat deze zone groter gemaakt kan worden door k constant te houden vanaf een bepaalde epoch.

Van de drie opties is de dynamisch k de beste.

$$k = \begin{cases} 0 & epoch[0,6) \\ (epoch-5)*0.0075 & epoch[6,10) \\ 0.03 & epoch[10,15] \end{cases}$$

wordt gebruikt voor de volgende experimenten. In deze experimenten wordt de hyperparameter p uit de Hamacher t-norm aangepast. Alle andere hyperparameters zijn gelijk aan de hyperparameters van het controle GAN.



Figuur 8: Het uitgebreid GAN waarbij $p = 1$ en het controle GAN geëvalueerd aan de hand van de constraints test (voor constraints T1 en T2) en de L1-norm test

Product t-norm

Uiteindelijk is er maar één experiment uitgevoerd met een andere waarde voor p en dit is met $p = 1$. Deze t-norm staat beter bekend als de product t-norm.

Dit experiment is buiten de t-norm, volledig hetzelfde als het experiment bij de dynamische k dat zeer goede resultaten had. Met $p = 1$ echter doet de corectheidswaarde het nauwelijks beter dan het controle GAN (zie Figuur 7). De conclusie die hieruit getrokken kan worden is dat indien een model goed werkt met een bepaalde t-norm, dat er dan geen garantie is dat het model nog altijd goed gaat werken met een ander t-norm.

Nuancering L1-norm

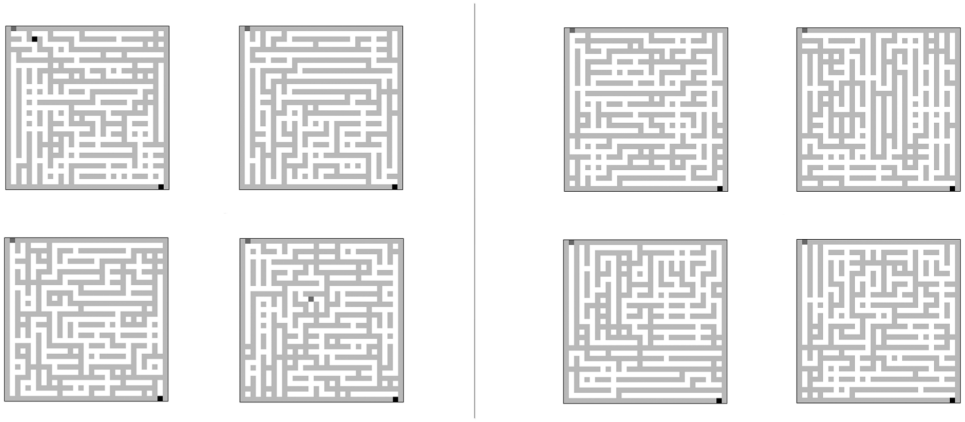
De diversiteit is zelfs voor de betere uitgebreide GAN's beduidend minder t.o.v. het controle GAN. Een mogelijke oorzaak hiervoor is dat het controle GAN meer fouten maakt en dat deze zorgen voor een hogere L1-norm. Vanaf epoch 9 is het gemiddeld aantal foute tegels van levels gegenereerd door het controle GAN echter nooit hoger dan 2,5. Bij het vergelijken van twee level komt de hoogst mogelijke L1-norm voor indien de fouten op verschillende tegels voorkomen. Indien dit het geval is zijn er gemiddeld 5 tegels waar één level correct is en het andere fout. Bij een discrete one-hot codering is de bijdrage aan de L1-norm van één tegel die verschilt altijd gelijk aan 2. De bijdrage van de fouten aan de gemiddelde L1-norm zal dus nooit hoger zijn dan 10. Vanaf epoch 9 ligt de gemiddelde L1-norm steeds rond de 300. De bijdrage van de foute tegels is dus rond de $3,3\%$. De foute tegels zullen er dus wel voor zorgen dat de L1-norm hoger is, maar deze bijdrage is zeer klein.

6.2 Onderzoeksvraag 2

Er werd geprobeerd een accurate implementatie te vinden voor het oplosbaarheidspredikaat *Opl* in de vorm van een getraind neuraal netwerk. Hiervoor werden verschillende architecturen getest.

MLP

Deze multilayer perceptron neemt een doolhof (in one-hot encoding) als invoer en eindigt in één neuron. De uitvoerwaarde ($\in [0,1]$) geeft weer hoe waarschijnlijk het is dat de



Figuur 9: Enkele willekeurig gegenereerde doolhoven door: links het 'visueel best getrainde' controle GAN (9 epochs getraind), rechts het 'visueel best getrainde' uitgebreid GAN (k neemt lineair toe en blijft dan constant, 10 epochs getraind)

invoer een oplosbaar doolhof is. De trainingsdata bestaat uit 20000 tupels $(doolhof, target)$ waarbij $target$ 1 is als het doolhof oplosbaar is en anders 0. Bij het trainen werd gebruik gemaakt van de stochastic gradient descent optimizer en $batch\_size = 40$.

invoer→uitvoer	Operatie
3844→600	nn.Linear met LeakyReLu
600→600	nn.Linear met LeakyReLu
600→400	nn.Linear met LeakyReLu
400→1	nn.Linear met Sigmoid

Tabel 2: Architectuur van de voorgestelde MLP voor classificatie van oplosbare doolhoven

Na 100 epochs trainen bereikte de loss een waarde van 0.0007. De gebruikte loss functie is bceloss. Na validatie met testdata (20000 ongeziene doolhof-target tuples) bleek echter dat de laagste gemiddelde loss van deze data bereikt was rond epoch 15 en een waarde had van 0.4964. Voor deze gewichten geldt er dat de gemiddelde afstand tussen de voorspelde waarde en de target $\frac{1}{20000}\sum_{i=1}^{20000}|MLP(\mathbf{x}^{(i)}) - target(\mathbf{x}^{(i)})|$ gelijk is aan 0.4951. Deze afstand is veel te groot om bruikbaar te zijn.

Na 100 epochs trainen bedraagt de gemiddelde loss van de testdata 1.6662. Een ongetraind netwerk heeft een gemiddeld verlies van 0.6934. Het feit dat de loss convergeert bij het trainen maar dat bij het valideren blijkt dat langer getrainde modellen slechter presteerden wijst op overfitting. Het netwerk leerde niet het algemene concept oplosbaarheid, maar eerder de individuele targets van de trainingsdoolhoven.

Variatie van hyperparameters zoals learning rate, momentum en batch size verhielpen dit probleem niet. Ook het gebruik van de Adam optimiser i.d.p.v. SGD was niet effectief.

MLP voor multi-label classificatie

Het tweede voorgestelde netwerk is een MLP voor multi-label classificatie. De laatste laag van dit neuraal netwerk bevat een neuron voor elke tegel van het doolhof. De waarde van zo'n neuron komt overeen met de waarschijnlijkheid dat deze tegel bereikbaar is vanaf de ingang van het doolhof dat als invoer diende. De trainingsdata bestaat uit 20000 $(doolhof, bereikbaarheidsmap)$ tupels. Er werd gebruik gemaakt van bceloss en de Adam optimizer.

invoer→uitvoer	Operatie
3844→7688	nn.Linear met LeakyReLu
7688→7688	nn.Linear met LeakyReLu
7688→7688	nn.Linear met LeakyReLu
7688→961	nn.Linear met Sigmoid

Tabel 3: Architectuur van de voorgestelde multi-label classificatie MLP voor classificatie van oplosbare doolhoven

Ondanks het proberen van vele verschillende waarden voor de hyperparameters slaagde dit model er niet in om te convergeren tijdens het trainingsproces. De laagst behaalde loss bedraagde 0.4207. Ook dit is te hoog om bruikbaar te zijn.

Neuraal netwerk met convolutionele lagen

Wegens de populariteit van Convolutionele neurale netwerken bij het classificeren van afbeeldingen werd er besloten te experimenteren met convolutionele lagen. [Brownlee, 2019] Dit netwerk eindigt in één artificieel neuron, die de waarschijnlijkheid geeft dat de invoer oplosbaar was. De trainingsdata voor dit netwerk is opnieuw 20000 $(doolhof, target)$ tupels. Er werd gebruik gemaakt van bceloss en de Adam optimizer.

Ook dit netwerk leed aan overfitting en haalde steeds zeer hoge gemiddelde loss waarden bij het valideren met testdata. De laagste gemiddelde loss van de testdata was 0.4377. Ook het introduceren van een max pooling laag met $kernel\_size = 2$ en $stride = 1$ na elke convolutionele laag bood geen verbetering.

Neuraal netwerk met convolutionele lagen voor multi-label classificatie

Dit netwerk is zeer gelijkaardig aan het vorige, maar is gemodificeerd voor multi-label classificatie. De laatste laag van

invoer→uitvoer	Operatie	in, out, k
(4,31,31)→(8,29,29)	Conv2d, ReLu	4, 8, 3
(8,29,29)→(16,25,25)	Conv2d, ReLu	8, 16, 5
(16,25,25)→(24,19,19)	Conv2d, ReLu	16, 24, 7
(24,19,19)→(32,11,11)	Conv2d, ReLu	24, 32, 9
(32,11,11)→3872	flatten	
3872→121	nn.Linear, LeakyReLu	
121→25	nn.Linear, LeakyReLu	
25→1	nn.Linear, Sigmoid	

Tabel 4: Architectuur van het voorgestelde netwerk met convolutionele lagen
in = input_layers, out = output_layers, k = kernel_size

het netwerk bevat weer voor elke tegel van het doolhof een overeenkomstig neuron die weergeeft hoe waarschijnlijk het is dat deze tegel bereikbaar is. De trainingsdata is opnieuw 20000 $(doolhof, bereikbaarheidsmap)$ tupels.

invoer→uitvoer	Operatie	in, out, k
(4,31,31)→(8,29,29)	Conv2d, ReLu	4, 8, 3
(8,29,29)→(16,25,25)	Conv2d, ReLu	8, 16, 5
(16,25,25)→(24,19,19)	Conv2d, ReLu	16, 24, 7
(24,19,19)→(32,11,11)	Conv2d, ReLu	24, 32, 9
(32,11,11)→3872	flatten	
3872→3000	nn.Linear, LeakyReLu	
3000→2000	nn.Linear, LeakyReLu	
2000→961	nn.Linear, Sigmoid	

Tabel 5: Architectuur van het voorgestelde netwerk met convolutionele lagen voor multi-label classificatie

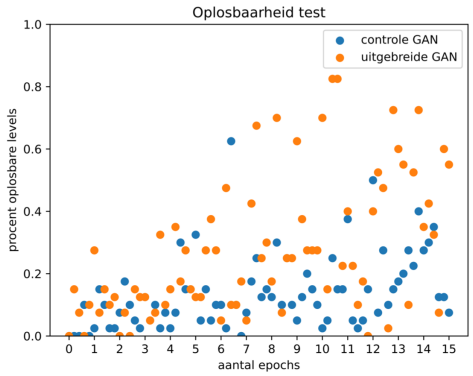
Ook dit netwerk slaagde er niet in om te convergeren tijdens het trainen. Er werden meerdere waarden voor de hyperparameters getest. De trainingsloss bleef schommelen rond 0.25-0.30. De laagste trainingsloss werd behaald rond epoch 5 en bedroeg 0.2467. Voor dit model is de gemiddelde afstand tussen de waarde van de uitgangtegel en de voorspelde waarde van de uitgangtegel gelijk aan 0.2139. Aangezien de waarde van de uitgangtegel 0 of 1 is is dit nog steeds te hoog.

6.3 Onderzoeksvraag 3

Om te onderzoeken of gebruik van het uitgebreid GAN gunstig is voor het genereren van doolhoven die aan de tractable constraints voldoen én oplosbaar zijn wordt de correctheidsformule gedefinieerd als: $fl(\mathbf{x}) = form\_1(\mathbf{x}) \wedge form\_2(\mathbf{x}) \wedge Opl(\mathbf{x})$. Wegens het feit dat er geen accurate implementatie is gevonden voor het oplosbaarheidspredikaat Opl was het echter niet mogelijk om betekenisvolle testen uit te voeren.

7 Conclusie

Fuzzy loss lijkt een bruikbare tool te zijn voor het opleggen van tractable constraints aan de uitvoer van een GAN. Het is wel een vereiste dat de relevante hyperparameters correct gekozen moeten zijn alvorens fuzzy loss daadwerkelijk een verbetering brengt. Verder onderzoek naar de effectiviteit van het voorgestelde uitgebreid GAN in het opleggen van complexe constraints en de diversiteit van gegenereerde objecten is nodig om hierover uitsluiting te brengen.



Figuur 10: Het uitgebreid GAN waarbij $Opl(\mathbf{x})$ geïmplementeerd wordt door het vierde voorgestelde (niet adequaat getrainde) netwerk en het controle GAN, getest op het maken van oplosbare doolhoven.

Onderzoeksvraag 1

De kans dat een doolhof gegenereerd door een getraind uitgebreid GAN aan de gewenste tractable constraints voldoet is aanzienlijk groter dan de kans dat een doolhof gegenereerd door een onder identieke omstandigheden getraind klassiek GAN aan deze constraints voldoet. Dit geldt echter enkel wanneer de modellen voldoende lang zijn getraind en er geschikte waarden zijn gekozen voor de hyperparameters van de fuzzy loss. Uit de experimenten is gebleken dat een dynamische waarde voor parameter k effectief is. Op deze manier wordt de fuzzy loss langzaam aan geïntroduceerd, op een moment dat de generator al in staat is om objecten te maken die ruwweg op doolhoven lijken. Voor de gebruikte implementatie van het uitgebreid GAN was er wel een beduidende vermindering in diversiteit van gegenereerde doolhoven in vergelijking met een klassiek GAN.

Onderzoeksvraag 2

Geen van de implementaties van het oplosbaarheidspredikaat Opl die in deze paper zijn voorgesteld waren in staat om accuraat oplosbare doolhoven af te beelden op een waarde ≈ 1 en onoplosbare op een waarde ≈ 0.

Onderzoeksvraag 3

Wegens het ontbreken van een accurate implementatie van Opl was het niet mogelijk om experimenten uit te voeren omtrent de effectiviteit van het uitgebreid GAN voor het opleggen van tractable én intractable constraints aan zijn output.

8 Verder Onderzoek

Diversiteit

Bij de experimenten is gebleken dat voor gebruik van het uitgebreid GAN een hoge correctheidswaarde samengaat met een lagere diversiteit van gegenereerde doolhoven. Verder onderzoek moet uitwijzen of een verlaagde diversiteit van gegenereerde objecten inherent is aan het gebruik van fuzzy loss. Hiervoor zouden implementaties van het uitgebreid GAN getest kunnen worden die gebruik maken van verschillende architecturen en waardefuncties. De Wasserstein loss waardefunctie zou wegens zijn gebruik bij het voorkomen van

mode collapse hier zeker aan bod moeten komen. [Google, 2020]

Oplosbaarheid

De voorgestelde netwerken voor het implementeren van Opl waren allemaal beperkt in grootte en complexiteit. Verdere studie omtrent een accurate implementatie van het oplosbaarheidspredikaat zou gericht moeten zijn op experimenten met substantiëlere neurale netwerken en grotere datasets.

Referenties

[Brownlee, 2019] Jason Brownlee. When to use mlp, cnn, and rnn neural networks, Aug 2019.

[Di Liello *et al.*, 2020] Luca Di Liello, Pierfrancesco Ardino, Jacopo Gobbi, Paolo Morettin, Stefano Teso, and Andrea Passerini. Efficient generation of structured objects with constrained adversarial networks, 2020.

[Diligenti *et al.*, 2017] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165, 2017. Combining Constraint Solving with Mining and Learning.

[Goodfellow *et al.*, 2014] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[Google, 2020] Google. Common problems nbsp;—nbsp; generative adversarial networks nbsp;—nbsp; google developers, Feb 2020.

[Jang *et al.*, 2016] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2016.

[Pytorch, 2019] Pytorch. Adam - pytorch 1.11.0 documentation, 2019.

[Wong, 2020] Wanshun Wong. What is gumbel-softmax?, May 2020.