
Methoden en Technieken: Sequence Learning

Unsupervised Learning

HOGESCHOOL VAN AMSTERDAM - MASTER APPLIED ARTIFICIAL INTELLIGENCE

Kristo Wind



Amsterdam
Januari 2023

Methoden en Technieken: Sequence Learning & Unsupervised Learning

Contents

1 Tijdreeksen	3
1.1 Wat voor neurale netwerk?	3
1.1.1 Fully connected netwerk	3
1.1.2 Conv1D netwerk	4
1.1.3 Recurrent netwerk	4
1.2 Enkele knoppen om aan te draaien	5
2 NLP	6
2.1 Preprocessing	6
2.1.1 Vectorizing tekst	6
3 Attention & Transformers	8
3.1 Toevoegen van temperatuur	10
4 Unsupervised Learning: ML	11
4.1 Dimensie-reductie	11
4.1.1 Lineair	12
4.1.2 Manifold Learning	12
4.2 Clustering	12
4.2.1 k -means clustering	12
4.2.2 Hiërarchische clustering	12
4.2.3 DBSCAN	13
5 Auto-encoders	13
6 Variational Auto-Encoders	15
7 Boltzmann Machines	17
7.1 KL Divergence	17
7.2 Graphical models	17
7.3 Restricted Boltzmann Machines	18
7.3.1 Voorbeeld: Movie Recommender Systems	19
7.3.2 Het model	19
7.3.3 Contrastive Divergence	19
7.3.4 De Monte Carlo benadering	19
7.3.5 De Markov Chain Monte Carlo benadering	20
8 Deep Belief Networks	21
9 Generative Adversial Networks (GANs)	22
9.1 Doelfunctie van een GAN	23
9.2 Loss-functies	24
10 Clustering op tijdreeksen	24
10.1 Shape-based approach	25
11 ROUGE	26
11.1 Precision and Recall	26
11.2 Using a package to calculate the ROUGE score	26
12 BERTScore	27
12.1 Small-scale experiment	28
13 Human Judgement	28

CONTENTS

14 Summary	28
15 Spraak	29
15.1 Klinkers	29
15.2 Co-articulatie	29
15.3 Socio-lingüistische variatie	30
16 Audio	30

Week 1

1 Tijdreeksen

Een **tijdreeks** X_t is een verzameling van waarnemingen, elk gedaan op een specifiek tijdstip t . De verzameling T van tijdstippen zou continu kunnen zijn, maar wij zullen uitgaan van een discrete verzameling T . Sterker nog, we zullen uitgaan van tijdreeksen met waarnemingen op een vaste afstand van elkaar.

Dit vak gaat over concentreren op het voorspellen van tijdreeksen. Andere doelen zijn:

- **Classificatie:** ken 1 of meer nominale labels toe aan een tijdreeks. Aan de hand van het gedrag van een motor bepalen of het binnenkort zal uitvallen.
- **Event detection:** specifiek signaal in een tijdreeks detecteren (hotword detection).
- **Anomaly detection:** detecteren van ongebruikelijk gedrag. Maakt vaak gebruik van unsupervised learning.

Tijdreeksanalyse zonder machine learning:

- Tijdreeksdecompositie
- Exponential smoothing (Holt Winters)
- S-ARIMA-modellen
- Fouries-analyse
- Dynamic regression

Deze technieken proberen de dynamiek van een systeem te begrijpen. Domeinkennis zal daarom waardevol zijn.

Stappenplan:

1. Data analyse
2. Splitsen (volgorde is essentieel (traindata is eerste stuk data, val volgende stuk en test laatste data))
3. Normaliseren traindata (μ en σ van traindata)
4. Pas eenheden zo nodig aan op het vraagstuk (timeseries_dataset_from_array())
5. Deze waarnemingen in batches verdelen en dat kan invoer zijn voor een neurale netwerk
6. Maatstaf MAE. Als model beter scoort dan nulmodel is het al goed. (nulmodel bijvoorbeeld temperatuur morgen 12:10 is zelfde temperatuur als vandaag 12:10).

1.1 Wat voor neurale netwerk?

1.1.1 Fully connected netwerk

MSE is beter om afgeleides mee te bepalen, dus als loss gebruiken. (plot x^2 en $|x|$ grafieken maar om te zien). MAE springt van negatieve afgeleide ineens naar positief. MSE gaat geleidelijk. RMSE heeft zelfde schaal als MAE, maar is niet hetzelfde. MAE is makkelijker te interpreteren. Zorg dus voor lossfuncties waarvan de afgeleides bekend zijn.

1.1.2 Conv1D netwerk

In plaats van een fully connected netwerk (die vaak niet een betere MAE dan nulmodel heeft), probeer model te helpen gerichter te zoeken, want alleen de data invoeren is vaak niet genoeg voor het netwerk. Data op een andere manier presenteren, zoals seizoenen, etc. Om dat te doen kunnen bijvoorbeeld Conv1D en MaxPooling toegevoegd worden, om relaties tussen datapunten te vinden en niet weg te gooien.

Tijdreeks in niet translatie-invariant, hoewel er wel een dagelijkse cyclus is, zal die cyclus veranderen. Volgorde is ook van belang in een tijdreeks. De recente geschiedenis heeft meer invloed dan de verre geschiedenis. Daar wordt nu geen rekening mee gehouden. Fully connected en CNN modellen zijn feed forward netwerken (geen geheugen).

1.1.3 Recurrent netwerk

Recurrente netwerken met geheugen, uitvoer van vroeger kan je gebruiken als invoer van nu.

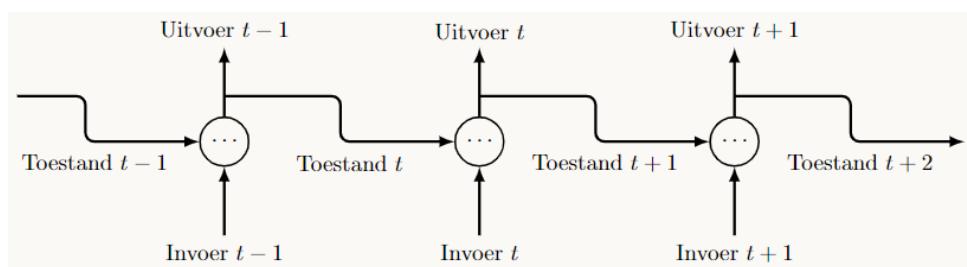


Figure 1: Recurrent Neural Network

Wat zou de uitvoer van een RNN moeten zijn? **Keras** houdt rekening met beide opties: alle uitvoer of alleen de uitvoer van de laatste stap. Dit wordt bepaald met `return_sequences`. Als je `SimpleRNN`-lagen wilt stapelen wil je de gehele uitvoer hebben per tijdstap. Dan moet `return_sequences=True` zijn. Het voordeel hiervan is dat je informatie uit enkele tijdstappen terug kan verkrijgen, maar het nadeel is dat je last hebt van een *vanishing gradient*. Als alternatief kunnen LSTM en GRU modellen gebruikt worden.

Long Short-Term Memory (LSTM)

Een LSTM lijkt erg op een simpele RNN. Er is alleen iets toegevoegd: een *carry* c^t .

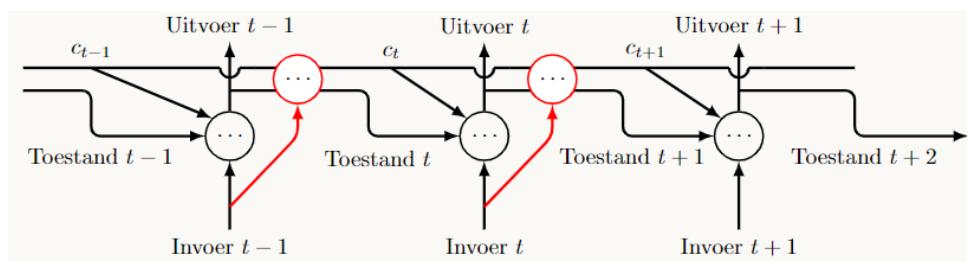


Figure 2: Long Short-Term Memory

De c^t hangen af van de invoer ($t - 1$), uitvoer ($t - 1$) en c^{t-1} . Als activatie-functies kiest men sigmoid-functies. Een interpretatie is dat deze activaties als poorten werken. Een LSTM is een voorbeeld van een *gated RNN*.

- De werking van de gates wordt bepaald door de gewichten.
- De gewichten worden gevonden middels een leerproces.

- De specificatie van de cellen bepaalt de hypotheseruimte, maar wat cellen doen wordt bepaald door de gewichten.
- De combinatie van de gates zou men moeten interpreteren als een voorwaarde op het zoeken (geen design).
- De schrijver is ervan overtuigd dat optimalisatie-algoritmes dit beter kunnen dan mensen.

Wees ervan bewust wat de LSTM-cel doet:

- Het kan informatie van een eerder tijdstip gebruiken op een later moment.
- Bestrijdt daarmee het effect van de verdwijnde gradient (vanishing gradient).

Dropout

In plaats van elke tijdstap willekeurig enkele eenheden weglaten, zou men elke tijdstap dezelfde willekeurige eenheden moeten weglaten. Dit kan op twee manieren:

- **dropout**: fractie om weg te laten van de invoer.
- **recurrent_dropout**: fractie om weg te laten van de recurrente toestand.

Recurrent_dropout zit standaard verweven in de LSTM laag, waarbij **dropout** een aparte laag is.

Omdat er wordt geregulariseerd met dropout kan men voor meer capaciteit kiezen. Wegens dropout duurt het langer voordat het zoeken convergeert. Daarom neemt men 5 keer zoveel epochs.

Probeer altijd te overfitten!

1. Als je overfit, gebruik je regularisatie (zonder capaciteit te verkleinen).
2. Als je door regularisatie niet meer overfit, probeer je de capaciteit weer te vergroten.
3. Ga net zo lang door tot je weer overfit.
4. Overfit je niet, dan kan je waarschijnlijk een beter model vinden.

Bidirectional RNNs (BRNNs)

Een bidirectional RNN combineert twee modellen. Een normale RNN en een RNN waarin je de tijd achterstevoren invoert. BRNNs werken goed op problemen waar de volgorde belangrijk kan zijn, maar wat de volgorde is hoeft niet belangrijk te zijn (zoals taal). Het is mogelijk om zinnen achterstevoren te lezen en te begrijpen (en dus te voorspellen).

1.2 Enkele knoppen om aan te draaien

- Aantal eenheden in elke recurrente laag van een stapeling.
- De hoeveelheid dropout.
- De learning rate en/of de optimizer.
- Probeer een stapel van dense-lagen ipv 1 dense-laag.
- Kortere of langere input.
- Andere sampling rate.
- Feature engineering.

Week 2

2 NLP

Natuurlijke taal...

- ...maakt gebruik van uitzonderingen
- ...kan dubbelzinnig zijn
- ...kan veranderen

Hoe wordt NLP gebruikt?

- Classificatie: wat is het onderwerp van de tekst?
- Context filtering: Bevat deze tekst scheldwoorden of ander misbruik?
- Sentiment analysis: is de tekst positief of negatief?
-
-
-

2.1 Preprocessing

2.1.1 Vectorizing tekst

- Standardize (vereenvoudigen van de tekst)
- Tokenization (splitsen van symbolen waarmee je wilt werken)
- Indexing (elke token krijgt een getal toegewezen)

Standardize: vermijd overbodigheid (denk na of het invloed heeft/belangrijk is):

- 'Duits' en 'duits'
- 'enquête' of 'enquete'
- 'ik loop, omdat' en 'ik loop omdat'
- 'hoe heet jij?' en 'hoe heet jij'
- 'hij liep', 'hij loopt' en 'hij [lopen]' (stemming)

Tokenization: Symbolen

- Woordniveau: splits op de spaties
- N-gram: splits in groepen van Noopeenvolgende woorden
- Karakterniveau: wordt niet vaak gebruikt bij text-generation

Bag-of-2-grams:

"ik ben blij" → {"ik", "ben", "blij", "ik ben", "ben blij"}

Een bag staat voor een multi-set (een verzameling met herhaling). Door middel van N-grams wordt er een heel lokale volgorde in het model ingevoerd. N-grams worden gebruikt bij niet-diepe modellen (vorm van feature engineering).

Indexing

Relateer een uniek geheel getal aan elk symbool uit de trainset (vocabulary, woordenlijst, symbolenlijst). Reserveer een index voor woorden die niet in de woordenlijst voorkomen (out of vocabulary-index of OOV-index). Gebruik als index waarde 1. De 0 ...

TextVectorization werkt alleen op CPU en kan zowel in datapreparatie-stap als in model gedeclareerd worden. Als je GPU gebruikt, is het sneller als je het in de data-preparatie declareert. Als je het ergens implementeert, wil je liever een model leveren waar de data-preparatie al in zit.

Stappenplan voor trainen:

1. Dataset splitsen
2. Definieer nulmodel (bijv. altijd positief (50% accuraatheid))
3. Kies de 20.000 meest voorkomende woorden voor in de woordenlijst.
 - Met 1-grams: accuraatheid (validatieset) van 88%.
 - Met 2-grams: accuraatheid (validatieset) van 89%.
 - Met 2-grams en TF-IDF accuraatheid (validatieset) van 89%.
 - TF-IDF: Hoe vaker een woord voorkomt in een tekst hoe belangrijker, hoe meer teksten dat woord bevatten hoe minder belangrijk.
4. Opnieuw een woordenlijst van 20.000 woorden. Gebruik alleen eerste 600 woorden (en padding). 600×20.000 matrix, dus heel veel parameters met veel nulletjes.
 - Accuraatheid (validatieset) van 87% en 3 uur per epoch.
5. Met eigen gedefinieerde word embedding (als genoeg data)
 - Accuraatheid (validatieset) van 90% en 7 minuten per epoch.
6. masking
 - Accuraatheid (validatieset) van 90% en 7 minuten per epoch.
7. Pretrained word embedding (**word2vec**)
 - Accuraatheid (validatieset) van 88% en 4 minuten per epoch.

Word embeddings

Word embeddings zijn vectorrepresentaties van woorden waar de semantische afstand tussen woorden overeenkomt met de meetkundige afstand van de vectoren. Dit maakt vectoren lager dimensionaal en minder ijl (minder sparse) vergeleken met one-hot-encoding. Typische voorbeelden van dimensies in een word embedding zijn 'geslacht' en 'meervoud'.

- Begin met willekeurige vectoren in de inbedding en leer de juiste vectoren zoals je de gewichten van het netwerk leert
- Maak gebruik van al bestaande embedding. Dit zijn pretrained embeddings.

Padding en masking

Padding is het eisen van een lengte van het aantal woorden. De informatie in het geheugen van de RNN zal verdwijnen als het aan het eind steeds nullen ziet. Een manier om dit tegen te gaan is *masking*.

3 Attention & Transformers

Sommige variabelen zullen we meer aandacht (attention) willen geven, omdat niet elk onderdeel even belangrijk is. Dit doen we door scores per variabele te berekenen.

De scores die we per variabele geven kunnen context-afhankelijk zijn. De context bepaald namelijk soms de betekenis van het woord. De score van een symbool is afhankelijk van de symbolen eromheen.

Self-attention

Inproduct van een word-embedding (hoek van de woorden tot van elkaar). Geeft een projectie het woord tot van de vectoren van de overige woorden.

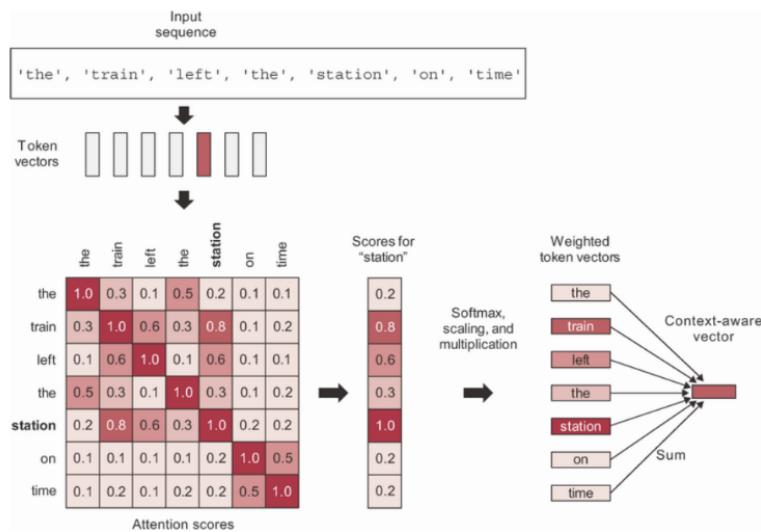


Figure 3: Self-attention (encoder)

Idee achter de transformer

De transformer was van origine een vertaler. De encoder geeft het woord bank (in de zin: 'Ik ga zitten op mijn bank') veel aandacht/attention, waardoor in de decoder niet *bank* gekozen wordt, maar *couch*.

Het query-key-value-model

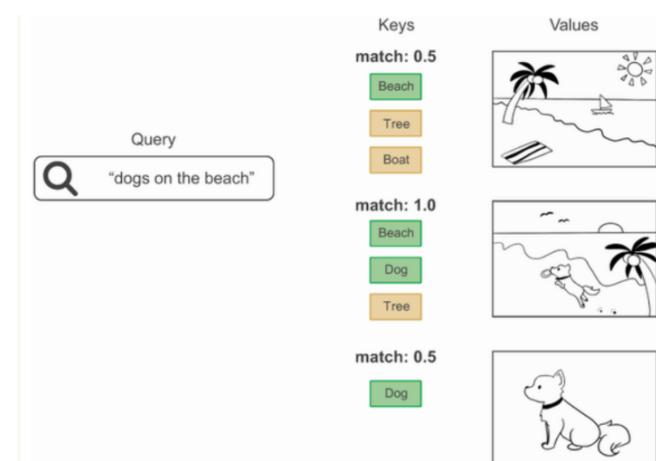


Figure 4: Query-key-value

Full transformer encoder

De transformer maakt verder gebruik van sequences door elk woord een getal mee te geven om de plek in de zin aan te duiden. De code van de volledige transformer encoder ziet er als volgt uit.

The code defines a Keras model named 'model_1'. It starts by setting parameters: vocab size (20000), sequence length (600), embed dim (768), num heads (2), and dense dim (32). It then creates inputs (shape (None, 600)), performs position embedding, and uses a TransformerEncoder layer. The output is then pooled using GlobalMaxPooling1D, followed by dropout and a dense layer with sigmoid activation. The model is compiled with binary_crossentropy loss and accuracy metric. Finally, a summary of the model's architecture and parameters is printed.

```
vocab_size = 20000
sequence_length = 600
embed_dim = 768
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int32")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.1)(x)
output = layers.Dense(32, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=output)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
```

Model: "model_1"		
Layer (Type)	Output Shape	Param #
=====	=====	=====
input_2 (InputLayer)	(None, None)	0
positional_embedding (Posit	(None, None, 256)	5273600
ionalEmbedding)		
transformer_encoder_1 (Tran	(None, None, 256)	543776
sformerEncoder)		
global_max_pooling1d_1 (Glo	(None, 256)	0
balMaxPooling1D)		
dropout_1 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 1)	257
=====	=====	=====
Total params:	5,817,633	
Trainable params:	5,817,633	
Non-trainable params:	0	

Figure 5: Full transformer encoder (met sequence)

Bag-of-words vs. Sequence models

De vuistregel om een modeltype te kiezen op prestatie (accuracy) voor tekstclassificatie luidt:

$$\frac{\text{aantal waarnemingen}}{\text{gemiddelde lengte van waarnemingen}} \begin{cases} < 1500 \text{ bag-of-bigrams} \\ > 1500 \text{ sequence model} \end{cases}$$

Week 3

Het genereren van een rij middels een model met conditional data noemen we *sampling*. De strategieën voor sampling zijn:

- **Greedy sampling.**

Kies steeds het woord met de hoogste kans. Nadeel: herhalingen en voorspelbare zinnen die niet echt lijken.

- **Stochastic sampling.**

Kies een woord middels een kansproces.

- uniforme verdeling (elk symbool even veel kans)
- de verdeling van het model

Nadeel: hoe weten we welk kansproces geschikt is voor het probleem?

3.1 Toevoegen van temperatuur

Temperatuur T : hyperparameter om de hoeveelheid stochasticiteit in het kansproces te beheersen. Als het model de kans op een symbool inschat met $P(\text{symbool}|X) = C_{\text{norm}}P(\text{symbool}|X)^{\frac{1}{T}}$

- Hoe lager T hoe kleiner de variatie
- Hoe hoger T hoe groter de variatie

Enkele overwegingen:

- Resultaten worden beter met meer data en grotere modellen.
- GPT-3: gestapeld transformer decoders op een enorme dataset.
- Een taalmodel gebruikt de statistische structuur van de teksten die het gevoerd krijgt.

Week 4

4 Unsupervised Learning: ML

Supervised learning voor- en nadelen:

- Uitstekend voor goed-gedefinieerde taken met voldoende labels
- Bijvoorbeeld classificeren van afbeeldingen
- Kosten van labelen kunnen hoog zijn en veel tijd kosten
- Supervised model kan beperkt generaliseren. Afhankelijk van de traindata, terwijl de wereld zoveel groter is.

Unsupervised Learning voor- en nadelen:

- Uitstekend voor problemen waar de structuren onbekend zijn of veranderen en er weinig gelabelde data is.
- Bijvoorbeeld het groeperen van afbeeldingen die op elkaar lijken.
- Minder handig in het beantwoorden van specifieke vragen.

Enkele unsupervised opdrachten:

- Geautomatiseerd labelen data
- Regulariseren
- Dimensie-reductie
- Feature engineering
- Detecteren van uitschieters
- Omgaan met drift van de data

Unsupervised-concepten:

- **Dimensie-reductie**
 - Het verminderen van het aantal variabelen.
Wat is de vorm van de datawolk?
- **Clustering**
 - Het groeperen van de waarnemingen.
Uit hoeveel wolken bestaat de datawolk?

4.1 Dimensie-reductie

- **Lineair**
 - Principal Component Analysis (PCA)
 - Random projection
- **Manifold learning (niet-lineair)**
 - Isometric mapping (Isomap)
 - Multidimensional scaling (MDS)
 - Locally linear embedding (LLE)
 - t-distributed stochastic neighbor embedding (t-SNE)

4.1.1 Lineair

Principal component analysis

We gaan ervan uit dat de vorm van de puntenwolk te benaderen is als een hoger-dimensionale matras (lineair) die in de variabele-ruimte ligt. In enkele dimensies is de matras lang, in andere dimensies is de matras dun. PCA detecteert de verschillende richtingen en de bijbehorende variantie (van de wolk). Vervolgens kan men de datawolk op een lager dimensionale ruimte projecteren.

Model kan leren om lijnen zo te trekken dat fouten worden geminimaliseerd.

Random projection

Dit is een projectie naar een lager dimensionale ruimte zonder eerst de wolk te bestuderen. De projectie maakt gebruik van willekeurig gekozen richtingen. De projectie is zo gekozen dat het afstanden tussen punten (zo goed mogelijk) bewaard. Maakt gebruik van een foutmaat ϵ . Hoe lager ϵ hoe beter de afstanden behouden blijven. Hoe lager ϵ hoe groter de inbeddingsruimte moet zijn.

PCA vs Random projection

PCA: adaptieve dimensie-reductie. Eerst de data bestuderen en dan projecteren. Random projection: niet-adaptieve dimensie-reductie. Projectie zonder de data van tevoren te bestuderen.

- Voordeel PCA: beter inzicht in de data.
- Voordeel Random Projection: bij veel variabelen praktischer.

4.1.2 Manifold Learning

De datawolk hoeft geen lineaire structuur in de ruimte te hebben. In dat geval kan men manifold learning gebruiken om de puntenwolk in te bedden in een lager dimensionale ruimte. Men probeert nog steeds afstanden te behouden. Dat kan over de hele ruimte, maar ook lokaal. Zie [Manifold Learning](#) voor een bespreking van verschillende algoritmes en voor- en nadelen.

4.2 Clustering

- k -means clustering
- Hiërarchische clustering
- DBSCAN

4.2.1 k -means clustering

Kies het aantal clusters k . Elk punt hoort bij het dichtsbijzijnde cluster. Kies de plek van de clusters zo dat de variantie binnen de clusters minimaal is.

- Voordeel: snel algoritme te gebruiken bij veel data.
- Nadeel: aantal clusters k moeten aangenomen worden.

4.2.2 Hiërarchische clustering

1. Begin met N clusters (elke waarneming)
2. Bereken alle afstanden (tussen punten en clusters) en voeg de twee dichtsbijzijnde samen (tot een nieuw cluster).
3. Herhaal bovenstaande stap tot er 1 cluster overblijft.

Uitvoer is een dendrogram waaraan men kan aflezen hoeveel clusters er in de puntenwolk zijn.

- Voordeel: het aantal clusters is een resultaat.
- Nadeel: kost veel rekenkracht.

4.2.3 DBSCAN

density-based spatial clustering of applications with noise. Maakt niet alleen gebruik van afstand, maar ook van dichtheid. Niet elk punt hoeft in een cluster te vallen. Bovenstaande maakt het geschikt voor het werken met uitschieters. Het algoritme detecteert 'ongewone' punten.

1. Een verzameling $S_{m,p}$ van minstens m punten die elk op afstand ϵ van elkaar staan vormt een cluster. Dit zijn de kernpunten.
2. Een punt dat op afstand ϵ van elk kernpunt is, wordt zelf ook een kernpunt.
3. Een punt dat op afstand ϵ van een kernpunt is, wordt aan het cluster toegevoegd (als grenspunt).
4. Stap 1 met overgebleven punten. Dit zijn de kernpunten van een nieuw cluster.
 - Als gevonden: stap 2, 3 met overgebleven punten en nieuw cluster.
 - Als niet gevonden: overgebleven punten zijn uitschieters.

5 Auto-encoders

Een PCA van p variabelen naar q componenten is te formuleren als een optimalisatieprobleem. Als \mathbf{X} een $n \times p$ data matrix is, vind een $p \times q$ matrix \mathbb{R} zodat

$$\mathbf{P} = \mathbf{X}\mathbb{R}$$

$$\mathbf{X}_{new} = \mathbb{R}^\top \mathbf{P}$$

en

$$\text{Reconstruction Error}^2 = \frac{1}{n} \sum_{i=1}^n |\vec{x}_i - \vec{x}_{new,i}|^2$$

De PCA projectie op \mathbb{R}^2 op de MNIST-dataset ziet er als volgt uit:

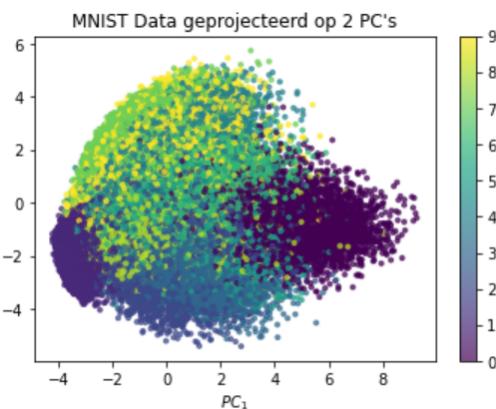


Figure 6: Architectuur gezicht genereren

Neuraal netwerk kan een PCA benaderen met lineaire activatiefunctie. Je verwacht geen lineaire deelruimte. Een auto-encoder kan de PCA op een niet lineaire manier benaderen (gebruik

niet lineaire activatiefunctie).

Een auto-encoder bestaat uit twee delen:

- Een encoder, die de invoer transformeert naar een midden laag.
- Een decoder, die de midden laag terug transformeert naar de invoer.

Wiskundig geformuleerd is de encoder een functie $\vec{h} = f(\vec{x})$ die de invoer $\vec{x} \in \mathbb{R}^p$ transformeert naar \mathbb{R}^q . De decoder probeert de encoder te inverteren, en is een functie $\vec{x}_{new} = g(\vec{h})$ die de middenlaag transformeert van \mathbb{R}^q naar \mathbb{R}^p .

We noemen de compositie $\vec{x}_{new} = g(f(\vec{x}))$ de reconstructie-functie.

- Als $q \ll p$ dan noemen we de autoencoder *undercomplete*. Er zal zeker weten informatie verloren gaan in de reconstructie (en dit willen we vaak juist).
- Als $q >> p$ dan noemen we de autoencoder *overcomplete*. In combinatie met regularisatie (om te zorgen dat het model daadwerkelijk blijf versimpelen) zijn deze modellen vaak krachtiger, maar moeilijker te trainen.

Door een willekeurige vector in de midden laag te kiezen en hier de decoder op los te laten kunnen kunstmatige datapunten gegenereerd worden.

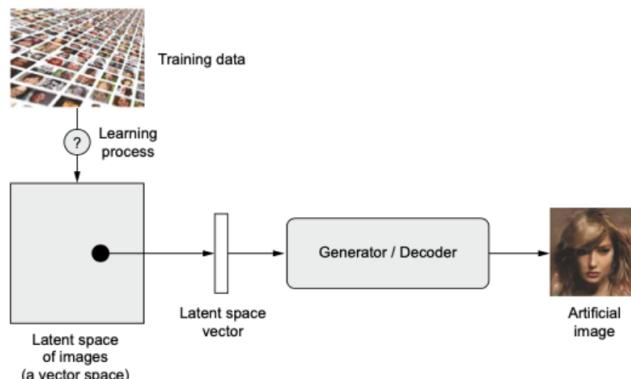


Figure 7: Architectuur gezicht genereren

Bepaalde richtingen in de latent space (niet-lineaire PCA plot) komen vaak overeen met bepaalde kenmerken.



Figure 8: Continu vlak van gegenereerde gezichten

Auto-encoder gebruik: je maakt je data kleiner, maar op een betekenisvolle manier.

Week 5

6 Variational Auto-Encoders

De auto-encoder probeert het kromme vlak te leren, waar de data op leeft. De PCA kan dat niet, die leeft op een lineair vlak.

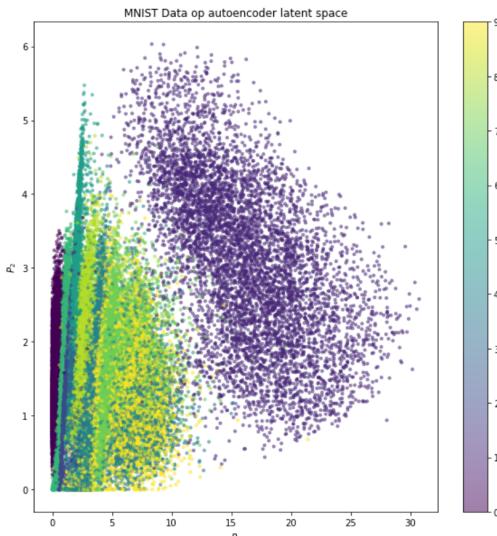


Figure 9: \mathbb{R}^2 Auto-encoder

- Waarnemingen worden gegenereerd via een kansproces
- De latent space beschrijft niet de exacte data maar een kansverdeling van dit proces
- We nemen aan dat deze kansverdeling een normaal verdeling is
- De encoder leert gemiddelde (μ) en variantie (σ^2) van de verdeling te schatten
- Vervolgens wordt er een trekking gedaan uit deze verdeling
- De decoder probeert deze trekking terug te transformeren naar de oorspronkelijke data.

Omdat de decoder een variantie moet schatten is een aangepaste loss-functie nodig:

$$\text{loss} = \text{Reconstruction loss} + KL\text{-loss}$$

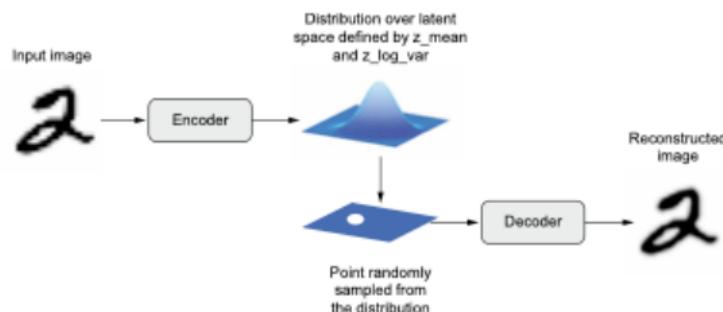


Figure 10: Variational auto-encoder architectuur

Variational auto-encoders geven niet alleen 2 coördinaten, maar ook 2 onzekerheden om het punt heen. Sampling zit alleen in het trainingsproces. Hij leert welke gemiddelde (μ) en

standaardafwijking (σ) goed zijn om de reconstructie-fout laag te hebben. Eigenlijk is het een soort verborgen data-augmentatie midden in je model. Nieuwe data genereren in je model om beter te trainen.

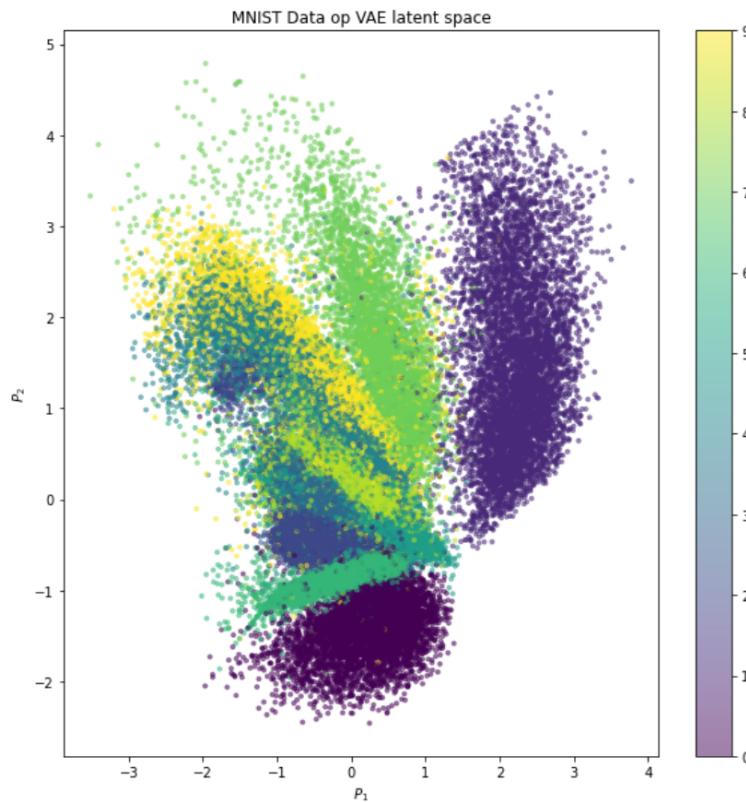


Figure 11: \mathbb{R}^2 Variational auto-encoder

Week 6

7 Boltzmann Machines

7.1 KL Divergence

De Kullback-Leiber (KL) divergentie is een begrip uit de informatie theorie. Informatie theorie gaat over het efficient comprimeren van data (denk aan zip).

Theorem 7.1: Self-attention

Als $p(x) = P(X = x)$ de kansverdeling is van een kansvariabele X , dan is

$$I(x) = -\log_2 p(x)$$

de self-information, gemeten in bits, van de gebeurtenis $X = x$.

De self-information meet (een ondergrens voor) het aantal bits dat nodig is voor symbool x bij een optimaal compressie algoritme.

Theorem 7.2: Shannon entropy

De verwachtingswaarde van $I(x)$ onder de kansverdeling $p(x)$

$$H(p) = \mathbb{E}[I(x)] = -\mathbb{E}[\log_2 p(x)] = -\sum_x p(x) \log_2 p(x)$$

noemen we de Shannon entropy van de kansverdeling p .

Dit is het gemiddeld aantal bits dat we nodig hebben om een symbool x te versturen.

Merk op: als $p(x)$ bestaat uit twee gebeurtenissen (binaire classificatie), dan is dit hetzelfde als de binary cross entropy die we al kennen vanuit classificatieproblemen.

Theorem 7.3: KL-divergence

Als $p(x)$ en $q(x)$ twee kansverdelingen zijn over dezelfde kansvariabele X , dan is de Kullback-Leibner divergentie de verwachtingswaarde over $p(x)$ van het verschil in self-information

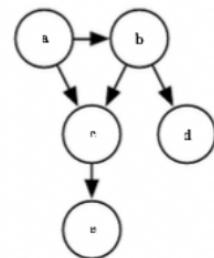
$$D_{KL}(p||q) = \mathbb{E}[\log_2 p(x) - \log_2 q(x)]$$

Als we een symbool x zouden comprimeren volgens een optimaal compressie algoritme over $q(x)$ terwijl x eigenlijk de verdeling $p(x)$ heeft, dan is de *KL-divergence* het gemiddeld aantal extra bits dat je nodig hebt.

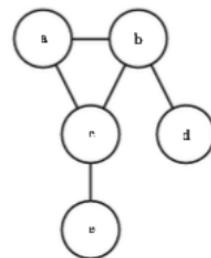
7.2 Graphical models

Graphical models komen in twee smaken:

- Directed models
- Undirected models



Directed Models



Undirected Models

In een *undirected model* beïnvloeden A en B elkaar wederzijds. De factorisatie van de kansverdeling is nu iets ingewikkelder, het kan geschreven worden als een product over iedere *clique* in de graaf.

$$p(a, b, c, d, e) = \frac{1}{Z} \phi_{a,b,c}(a, b, c) \phi_{b,d}(b, d) \phi_{c,e}(c, e)$$

Theorem 7.4: Boltzmann machine

Een Boltzmann machine is een undirected graphical model waarvan de kansverdeling de volgende vorm heeft:

$$p(\vec{x}) = \exp(-E(\vec{x}))$$

De functie $E(\vec{x})$ word ook wel de *energy function* genoemd.

Vaak word er vervolgens nog een opsplitsing gemaakt tussen

- *visible* nodes, deze zijn gemeten en zitten in je (trainings) data
- *hidden* ofwel *latent* nodes

Patel schrijft: "These unrestricted Boltzmann machines use neural networks with neurons that are connected not only to other neurons in other layers but also to neurons within the same layer. That, coupled with the presence of many hidden layers, makes training an unrestricted Boltzmann machine very inefficient. Unrestricted Boltzmann machines had little commercial success during the 1980s and 1990s as a result."

De oplossing hiervoor is de ***Restricted Boltzmann Machine***

7.3 Restricted Boltzmann Machines

Eerder definieerden we een Boltzmann Machine als een undirected graphical model met visible (invoer) en latent (hidden) nodes. We merkten op dat de onderlinge verbindingen tussen de latent nodes, en tussen de visible nodes het trainen heel moeilijk maken. Een Restrictive Boltzmann Machine lost dit op door deze verbindingen te verbieden.

Dit correspondeert met de volgende kansverdeling:

$$p(\vec{v}, \vec{h}) = \exp(-E(\vec{v}, \vec{h}))$$

met als energie

$$E(\vec{v}, \vec{h}) = -\vec{a} \cdot \vec{v} - \vec{b} \cdot \vec{h} - \vec{v} \cdot W \vec{h}$$

Hier zijn \vec{a} en \vec{b} bias vectoren en W een matrix die de overgang van de *visible* van en naar de *hidden* laag definieerd.

7.3.1 Voorbeeld: Movie Recommender Systems

Als voorbeeld gaan we een RBM gebruiken om een aanbevelingsmodel voor films te maken. We gebruiken een selectie van de MovieLens 20M Dataset, deze bevat 90213 film recensies gemaakt door 1000 personen over 1000 films.

- Om dit een binair classificatieprobleem te maken negeren we de recensies zelf, en kijken we alleen of een persoon film i wel ($v_i = 1$) of niet ($v_i = 0$) gezien heeft.
- Iedere rij in onze data bestaat dus uit een vector $\vec{v} = (v_1, v_2, \dots, v_{1000}) \in [0, 1]^{10}$

Doel is om de kans $P(v_i = 1)$ te voorspellen dat iemand een film wilt zien.

- De invoer en uitvoer van ons model zijn dus eigenlijk hetzelfde.
- We doen dit met een RBM met 1000 *visible* nodes (onze in-/uitvoer) en 10 *latent* nodes die alleen de waardes 0 of 1 aannemen.
- De *latent* nodes zijn dus een vector $\vec{h} = (h_1, h_2, \dots, h_{10}) \in [0, 1]^{10}$
- De intuïtie is dat de *latent* nodes een soort van genrevoorkeur (horror, rom-com, documentaires, ...) coderen.

7.3.2 Het model

Het model geeft ons twee belangrijke voorwaardelijke kansverdelingen:

- De forward pass

$$p(h_j = 1 | \vec{v}) = \sigma(b_j + \sum v_i W_{ij})$$

- De backward pass

$$p(v_j = 1 | \vec{h}) = \sigma(a_j + \sum W_{ij} h_j)$$

met $\sigma(x) = \frac{1}{1+e^{-x}}$ de gebruikelijke sigmoid functie.

7.3.3 Contrastive Divergence

De parameters (W_{ij} , a_i en b_j), worden gevonden door de *log-likelihood* te maximaliseren met gradient ascend. Dit komt neer op de marginale kans $p(\vec{v})$ te maximaliseren. Dit geeft onder andere de volgende update regel voor de gewichten W_{ij} :

$$\Delta W_{ij} = r \left(\underbrace{\mathbb{E}[v_i h_j | \vec{v}]}_{\text{data}} - \underbrace{\mathbb{E}[v_i h_j]}_{\text{model}} \right)$$

met r de *learning rate*. Volgens Hinton is dit verschil bijna gelijk aan het verschil in twee KL-divergences.

Probleem: om $\mathbb{E}[v_i h_j]$ exact uit te rekenen is een sommatie over alle mogelijkheden voor \vec{v} en \vec{h} nodig. Dit zijn 2^{10000} termen.

7.3.4 De Monte Carlo benadering

Een Monte-Carlo benadering is een manier om verwachtingswaarden $\mathbb{E}[f(x)]$ waar x een kansverdeling $p(x)$ heeft, te benaderen

- Neem een steekproef van x uit $p(x)$
- Bereken $f(x)$
- Bereken het steekproefgemiddelde van $f(x)$

Probleem: de kansverdeling de kansverdeling $p(\vec{v}, \vec{h})$ heeft 2^{10000} mogelijke uitkomsten...

7.3.5 De Markov Chain Monte Carlo benadering

De manier om $\mathbb{E}[v_i h_j]$ te berekenen zonder een trekking uit een onmogelijke kansverdeling te hoeven nemen gaat als volgt:

- Neem een aantal (een *batch*) van echte waarnemingen $\vec{v}^{(0)}$
- Bereken de voorwaardelijke kansverdeling $P(h_j = 1 | \vec{v}^{(0)}) = \sigma(b_j + \vec{v}_i^{(0)} W_{ij})$
- Trek $\vec{h}^{(0)}$ uit deze verdeling
- Bereken de voorwaardelijke kansverdeling $P(v_i = 1 | \vec{h}^{(0)}) = \sigma(a_i + W_{ij} \vec{h}_j^{(0)})$
- Trek $\vec{v}^{(1)}$ uit deze verdeling
- Bereken de voorwaardelijke kansverdeling $P(h_j = 1 | \vec{v}^{(1)}) = \sigma(b_j + \vec{v}_i^{(1)} W_{ij})$
- Trek $\vec{h}^{(1)}$ uit deze verdeling
- Ga zo door, totdat de voorwaardelijke kansverdelingen lijken te convergeren, dit geeft een trekking $\vec{v}^{(n)}, \vec{h}^{(n)}$
- Een gemiddelde hierover geeft onze benadering

Week 7

8 Deep Belief Networks

Het idee van een DBN is dat de *features* van een RBM gebruikt worden als invoer voor een volgende RBM, en zo een paar lagen diep.

- In feite is dit een Boltzmann Machine met een zwakkere restrictie dan een RBM, er mogen nu wel correlaties zijn tussen *hidden nodes*, maar niet in dezelfde laag.
- De diepere lagen leren meer en meer abstracte features.

Voordeel is dat deze lagen 1 voor 1 getraind kunnen worden. Dit maakt het een van de historisch eerste *deep learning* modellen die efficiënt getraind kon worden.

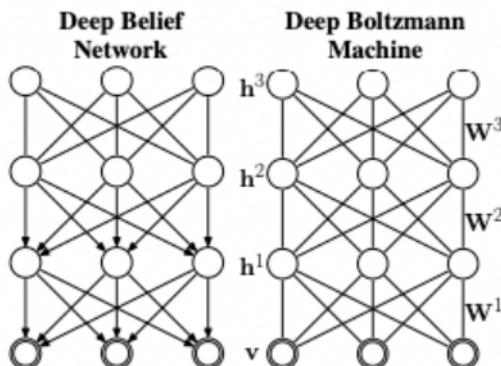


Figure 12: Deep Belief Network vs. Deep Boltzmann Machine

We kunnen nieuwe afbeeldingen genereren door een oorspronkelijk beeld nu aan de *visible* laag aan te bieden, de *hidden* lagen een voor een te samplen, en dan terug te gaan en de *visible* lagen te samplen.

We kunnen de gewichten en bias van een DBN gebruiken om een neuraal netwerk te initialiseren. Dit is een vorm van *transfer learning*.

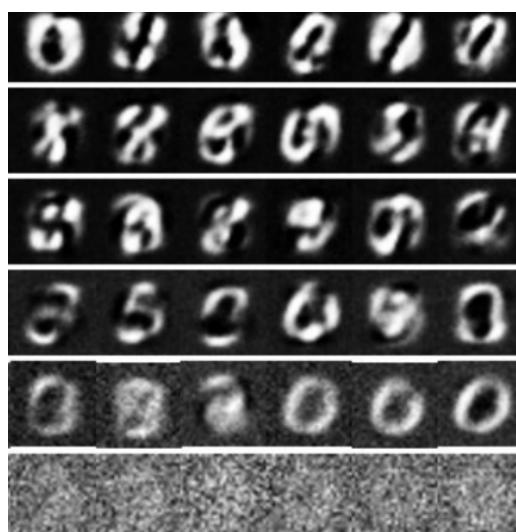


Figure 13: Deep Belief Network on MNIST-dataset

Week 8

9 Generative Adversial Networks (GANs)

GANs (zie [hier](#)) zijn een alternatief voor andere generatieve modellen zoals:

- Deep directed graphical models
- Deep undirected graphical models
- Generative autoencoders

En is een alternatief voor het leren van *latent spaces* van afbeeldingen.

GANs bestaan uit twee componenten:

- **Generator network**

Heeft als invoer een willekeurig punt uit de latent space en genereert daar een afbeelding van.

- *Discriminator network (adversary, tegenstander)*

Heeft als invoer een afbeelding en bepaalt of dat een afbeelding uit de trainset is of een gegenereerde afbeelding.

Een GAN wordt zo getraind dat:

- de generator goed wordt in het misleiden van de discriminator.
- de discriminator goed wordt in het herkennen van afbeeldingen van de generator.

Men zou de generator kunnen zien als de vervalser en de discriminator als de expert die vervalsingen probeert te herkennen.

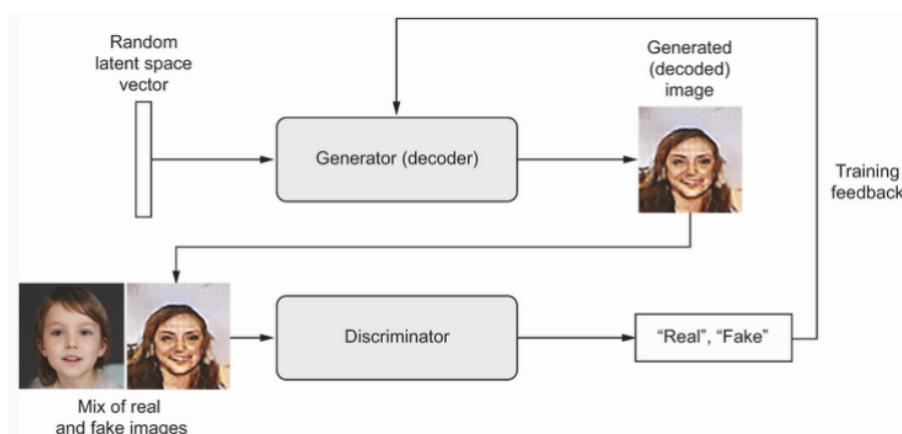


Figure 14: GAN visualisatie (hoog over)

Voorbeeld:

1. Trek m (minibatch) vectoren z uit de latent space
2. De generator G transformeert deze z naar afbeeldingen $G(z)$ van de vorm $(64, 64, 3)$
3. Combineer de afbeeldingen met m foto's uit de dataset
4. De discriminator D kan nu supervised getraind worden

5. Herhaal k (hyperparameter) keer
6. Trek opnieuw m vectoren z
7. De generator wordt getraind door de $GAN(z) = D(G(z))$ zo te trainen dat het de $G(Z)$ als echt classificeert, terwijl D niet verandert

Enkele beslissingen wat betreft dit voorbeeld:

1. We trekken uit de latent space volgens een normaalverdeling (ipv uniform)
2. GANs blijven vaak 'hangen'. Introduceren van toeval werkt hiertegen
 - Aan de labels wordt ruis toegevoegd
 - Dropout in de discriminator
3. Sparse gradients werken GANs tegen. Daarom gebruiken we strides en leaky-relu's ipv pooling en relu's
4. Kies een kernel size die deelbaar is door de stride size (tegen dambordpatroon)

Model: "generator"		
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 8192)	1056768
reshape (Reshape)	(None, 8, 8, 128)	0
conv2d_transpose (Conv2DTra nspose)	(None, 16, 16, 128)	262272
leaky_re_lu_3 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_transpose_1 (Conv2DT ranspose)	(None, 32, 32, 256)	524544
leaky_re_lu_4 (LeakyReLU)	(None, 32, 32, 256)	0
conv2d_transpose_2 (Conv2DT ranspose)	(None, 64, 64, 512)	2097664
leaky_re_lu_5 (LeakyReLU)	(None, 64, 64, 512)	0
conv2d_3 (Conv2D)	(None, 64, 64, 3)	38403
<hr/>		
Total params:	3,979,651	
Trainable params:	3,979,651	
Non-trainable params:	0	

Figure 15: Generator

9.1 Doelfunctie van een GAN

$$\min_G \max_D [E_x(\ln(D(x))) + E_z(\ln(1 - D(G(z)))]$$

- x uit dataset
- z gegenereerd
- $D(x)$ de kans (volgens D) dat x uit data komt
- $D(G(z))$ de kans (volgens D) dat $G(z)$ uit data komt

Model: "discriminator"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	3136
leaky_re_lu (LeakyReLU)	(None, 32, 32, 64)	0
conv2d_1 (Conv2D)	(None, 16, 16, 128)	131200
leaky_re_lu_1 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	262272
leaky_re_lu_2 (LeakyReLU)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dropout (Dropout)	(None, 8192)	0
dense (Dense)	(None, 1)	8193

Total params: 404,801
Trainable params: 404,801
Non-trainable params: 0

Figure 16: Discriminator

9.2 Loss-functies

De loss-functies voor de generator zijn als volgt te definiëren:

$$\min_G [E_x(\ln(D(x))) + E_z(\ln(1 - D(G(z)))]$$

is equivalent aan

$$\min_G [-E_z(\ln(D(G(z)))]$$

Dit komt overeen met het minimaliseren van de binary cross-entropy $-\hat{y}\ln(\hat{y}) - (1 - \hat{y})\ln(1 - \hat{y})$, met $y = 1$ als afbeelding uit data en $y = 0$ als afbeelding is gegenereerd.

De loss-functies voor de discriminator zijn als volgt te definiëren:

$$\max_D [E_x(\ln(D(x))) + E_z(\ln(1 - D(G(z)))]$$

is equivalent aan

$$\min_D [-E_x(\ln(D(x))) - E_z(\ln(1 - D(G(z)))]$$

Dit komt ook overeen met het minimaliseren van de binary cross-entropy, die hierboven beschreven is.

Bij een GAN zal het optimum niet vast liggen. Het trainen van de discriminator beïnvloedt het optimum van de generator (en andersom). Het algoritme zal op zoek gaan naar een balans.

10 Clustering op tijdreeksen

Rekening houden met:

- Schaling en translatie
- Verschuiving in de tijd
- Verlenging of verkorting
- Missende of verkeerde waarden

- Complexiteit (ruis)

Soorten tijdreeks clustering:

- **Whole time-series clustering** (veel tijdreeksen waar clusters op moeten komen)
 - **Time-series conversion** (domeinafhankelijk, past de data aan)
 - * **Model-based approach** (transformeren: zoals fourier-analyse, r.c. bepalen)
 - * **Feature-based approach** (nieuwe variabele maken, hopen op meer informatie daarin en op clusteren)
 - **Shape-based approach/raw-data-based approach** (minder domeinafhankelijk, wat lijkt wel en wat niet: hele data erin gooien)
- **Subsequence clustering** (binnen tijdreeks labelen)

10.1 Shape-based approach

Je wilt dat de afstand anders gedefinieerd wordt, waardoor alle waves met dezelfde vorm in hetzelfde kader wordt geplaatst. Dus dat alleen de shape/vorm van belang is.

- Normaliseer de data
- Kies afstandsmaat
 - **Euclidische afstand** (snel: geen verschuivingen oplossen, 'redelijke resultaten')
 - **Dynamic Time Warping** (accurater: met andere snelheden de vormen benaderen)
 - **Shape based distance measure** (snel en accuraat: y vasthouden, x verschuiven.
Dan mis je iets, maar zet je een 0 voor in de plaats)
- Kies clusteringsalgoritme
 - **Partitional** (k -means, k -medioids, k -shape)
 - **Density based** (DBSCAN of HDBSCAN)

Bonus: Generative Text Metrics

The scope of this workshop is about automatic metrics used for abstractive text summarization.

11 ROUGE

The ROUGE metric (used for comparing exact words), developed by Lin in 2004, is used to determine the lexical similarity between a generated text and a set of reference texts, typically created by humans. This metric is often employed in conjunction with evaluations from a crowd-sourced group of human annotators.

```
generated_summary = "The MAAI2022 team is the best team ever"  
reference_summary = "The team MAAI2022 is the best"
```

Figure 17: Text summary

11.1 Precision and Recall

A way to see how similar two texts are is by counting the number of tokens they have in common (6 in this case). ROUGE proposes a more clever approach computing the precision and recall scores for the overlap. ROUGE uses recall to measure how much of the reference summary is included in the generated summary. In other words, it tells you how many important pieces of information were included in the generated summary, by comparing it to the reference one. The formula to calculate recall when comparing tokens is as follows:

$$\text{recall} = \frac{\text{number of overlapping tokens}}{\text{total number of tokens in reference}}$$

This formula results in a recall of $6/6 = 1$ with unigrams, which means that all words in the reference summary were also included in the generated summary. If the generated summary is verbose and contains unnecessary repetition, this also would have a high recall score, but not necessarily concise.

To address this issue, ROUGE also calculates precision, which measures the proportion of relevant information in the generated summary as follows:

$$\text{precision} = \frac{\text{number of overlapping tokens}}{\text{total number of tokens in generated text}}$$

When applying this formula to the example of verbose summary, it would give a precision of $6/8 = 0.75$, which is lower than recall of $6/6 = 1$ obtained by the summary. To get a more complete picture of how good a summary is, ROUGE generally calculates both precision and recall and then finds the F1-score, which is the harmonic mean of these two values. In practice, the F1-score is commonly used to report the summary's performance.

11.2 Using a package to calculate the ROUGE score

The snippet below uses the `rouge-score` package to calculate the score. ROUGE has many variations and the simplest one called rouge-1 calculates de overlaps of unigrams - more specifically computing the overlap of words, as we discussed above.

Other known variations of ROUGE scores: For example, ROUGE-2 looks at the overlap of two-word phrases (bigrams). ROUGE-L is based on the longest common subsequence (LCS)

```
from rouge_score import rouge_scorer

rouge_scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)

rouge_scores = rouge_scorer.score(reference_summary, generated_summary)

rouge_scores

{'rouge1': Score(precision=0.75, recall=1.0, fmeasure=0.8571428571428571),
 'rouge2': Score(precision=0.2857142857142857, recall=0.4, fmeasure=0.3333333333333333),
 'rougeL': Score(precision=0.625, recall=0.8333333333333334, fmeasure=0.7142857142857143)}
```

Figure 18: ROUGE

between our model output and reference, i.e. the longest sequence of words (not necessarily consecutive, but still in order) that is shared between both. A longer shared sequence should indicate more similarity between the two sequences.

12 BERTScore

BERTScore (uses embedding of words as sentiment) is another metric for text generation. Some highlights from [Huggingface](#):

1. It computes a similarity score for each token in the candidate sentence with each token in the reference sentence.
2. It leverages the pre-trained contextual embeddings from BERT models.
3. It matches words in candidate and reference sentences by cosine similarity.

BERTScore computes precision, recall and F! measures.

```
from bert_score import BERTScorer

# Using a large model
bert_scorer = BERTScorer(lang='en',model_type='microsoft/deberta-xlarge-mnli',rescale_with_baseline=True)

# calculating the score
p, r, f1 = bert_scorer.score([generated_summary], [reference_summary])

print('BERTScorer metrics\n\nReference text:\n{}\nGenerated text:\n{}\n'.format(reference_summary, generated_summary))

print('\nPrecision: {} Recall: {} F1: {}'.format(p[0].item(),r[0].item(),f1[0].item()))

BERTScorer metrics

Reference text:
the quick brown fox jumped over the lazy dog

Generated text:
The fast chocolate fox hopped over the idle dog

Precision: 0.9016466736793518 Recall: 0.90217589378357 F1: 0.9020294547080994

from bert_score import plot_example

plot_example(generated_summary, reference_summary, lang="en", model_type='microsoft/deberta-xlarge-mnli', rescale_with_baseline=True)
```

Figure 19: BERTScore

12.1 Small-scale experiment

We are using a small-scale experiment with original texts, their respective references, and generated summaries. You can have a glimpse by look at the datasets/dataset.txt file.

```
import os

# a simple function used to Load the dataset
def load_dataset():
    dataset = []
    with open(os.path.join( './dataset', 'dataset.txt')) as f_read:
        content = f_read.read().splitlines()
        for line in range(0,len(content),5):
            record = {'source':content[line], 'original': content[line+1], 'reference':content[line+2], 'generated1':content[line+3], 'generated2':content[line+4]}
            dataset.append(record)
    return dataset

dataset = load_dataset()
```

Figure 20: BERTScore experiment

13 Human Judgement

o ensure semantic and factual accuracy, it is important to consult with domain experts who have knowledge of the human-written summaries.

The work of Kryscinski et al. proposes an approach to evaluate the quality of generated text, going beyond current metrics. It includes human annotators as part of the evaluation process. More Information can be found on her article [Neural Text Summarization: A Critical Evaluation](#).

Authors claimed that experts and crowd-sources can evaluate generated text using the following dimensions:

1. relevance (selection of important content from the source)
2. consistency (factual alignment between the summary and the source)
3. fluency (quality of individual sentences)
4. coherence (collenctive quality of all sentences)

14 Summary

- To evaluate the performance of an abstractive summarization model, ROUGE can be used to measure the overlap between the generated summary and the reference summary.
- The BERTScore metric (and a variety of others) attempts to overcome ROUGE limitations, by leveraging the pre-trained contextual embeddings from BERT. Then it matches words in candidate and reference sentences by cosine similarity.
- These automatic metrics can be used as an initial indicator of how much the generated summary overlaps with the human written summary.
- To ensure semantic and factual accuracy, it is important to consult with domain experts who have knowledge of the human-written summaries.

Bonus: spraakherkenning

15 Spraak

- stemhebbende letters: [z], [v], [b], [d]
- stemloze letters: [s], [f], [p], [t]

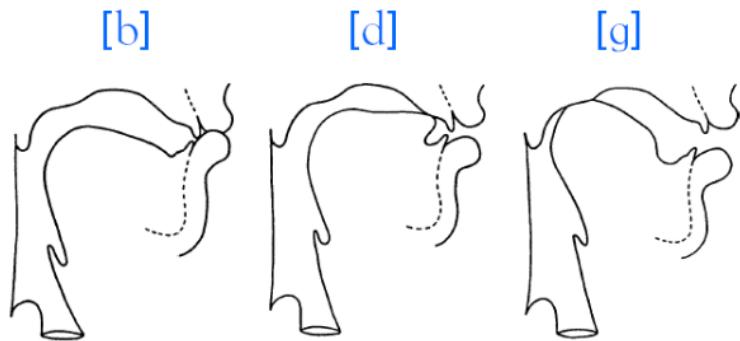


Figure 21: Plaats van articulatie

15.1 Klinkers

- Geen blokkade
- Stemhebbend
- Gemiddelde duur: 70 ms

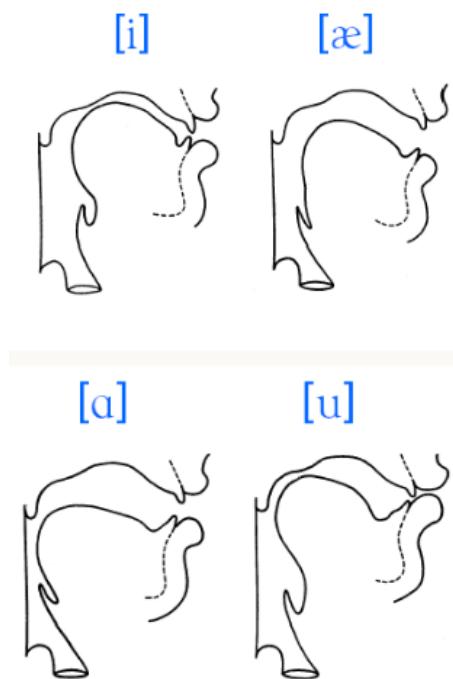


Figure 22: Klinkers

15.2 Co-articulatie

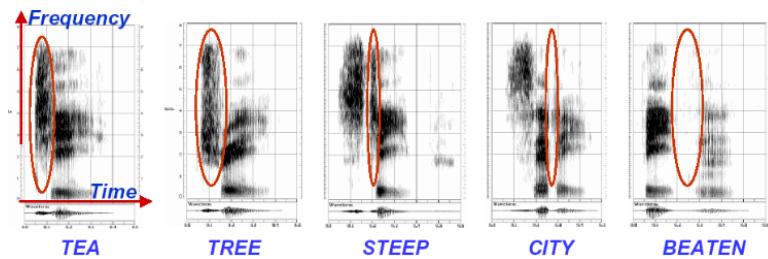


Figure 23: Co-articulatie

15.3 Socio-lingüistische variatie

- Dialect
- Geslacht, leeftijd, sociale status
 - Bijv. mannen gebruiken vaker niet-standaard vormen
- Identiteit
 - Saks, Macy's, Klein: [r] in fourth floor
 - Martha's Vineyard

Medewerkers van diverse klassen warenhuizen wordt gevraagd op welke verdieping een product ligt, wetende dat het op de vierde verdieping ligt. Het bleek dat bij warenhuis Saks (chique) de [r] veel vaker wordt uitgesproken door de medewerkers dan in Klein (minder chique).

[r]	always	sometimes	never
Saks	30	32	6
Macy's	20	31	74
S.Klein	4	17	50

Figure 24: Socio-lingüistische variatie

16 Audio

De audiogolf van een woord:

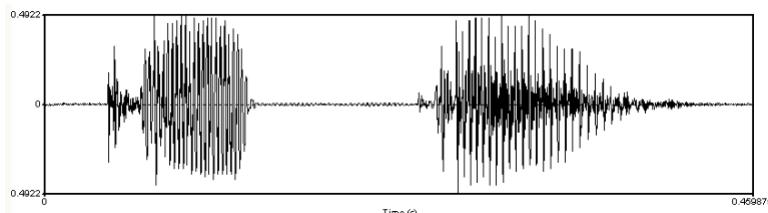


Figure 25: Audiogolf

Een sample (25 ms) van de audiogolf lijkt een patroon te bevatten:

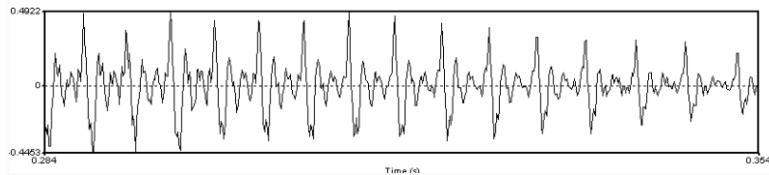


Figure 26: Audiogolf sample

We voeren een Fourier transformatie uit op de sample:

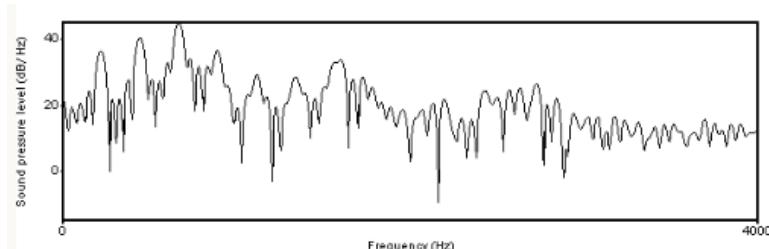


Figure 27: Fourier transformatie

Het audiospectrum van de Fourier transformatie op toon [iy]:

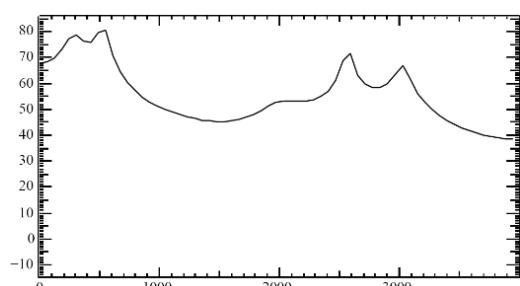


Figure 28: [iy] in 'She'

Dit proces wordt elke 10 ms uitgevoerd. Dat wordt *windowing* genoemd. Als alle audiospectra van de Fouriertransformatie achter elkaar geplakt worden ontstaat de volgende spectrogram van het woord 'bericht':

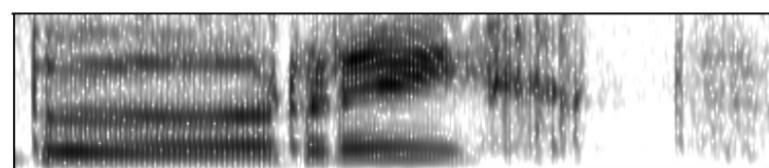


Figure 29: Spectrogram