



De Wiskunde achter ChatGPT

David van Batenburg
Ömer Eroğlu
Bouke Hoekstra
Jordi Mellema
Midas Scheffers
Nicos Starreveld

Januari 2025

Inhoudsopgave

1 Voorwoord	4
2 Introductie AI	5
2.1 Supervised Learning	6
2.2 Unsupervised Learning	9
2.3 Reinforcement learning	10
2.4 Wat wij gaan doen	12
3 Matrices	13
3.1 Wat is een matrix?	13
3.2 Basisoperaties op matrices	14
3.3 Vector of matrix?	15
3.4 Matrixvermenigvuldiging	15
4 Gradiënt	19
4.1 Functies in meerdere variabelen	19
4.2 Partiële afgeleiden	21
4.3 Gradiënt van functies	22
4.4 Het Gradient descent algoritme	25
4.5 Hoe werkt het algoritme?	25
4.5.1 Nadelen van gradient descent	26
5 Neurale netwerken en het feed forward algoritme	28
5.1 Het netwerk	28
5.2 In de praktijk	29
6 Het backpropagation algoritme	32
6.1 De Cost-functie	32
6.2 De aanpassing bepalen	33

7 Opgaven	35
7.1 Matrices	35
7.2 Gradiënt	36
7.3 Het feed forward-algoritme	38
7.4 Het backpropagation algoritme	40

Hoofdstuk 1

Voorwoord

De toepassingen van kunstmatige intelligentie zijn aanwezig in allerlei aspecten van ons dagelijks leven, van algoritmes die patronen in data kunnen vinden, programma's zoals ChatGPT, Dall-E, en AlphaZero, tot zelfbesturende auto's en beeld- en geluidherkenningssoftware. De maatschappelijke discussie over hoe we hiermee om moeten gaan, zowel op persoonlijk als maatschappelijk niveau is heel actief. In deze module hebben we een aantal belangrijke wiskundige concepten die ten grondslag liggen aan kunstmatige intelligentie verzameld en uitgelegd op een niveau geschikt voor leerlingen van VWO 5 of 6 die wiskunde D volgen, of wiskunde B hebben gevolgd.

Alle materiaal uit deze syllabus mag hergebruikt worden voor educatieve of andere niet-commerciële redenen. Alle materiaal valt onder een Creative Commons Licentie en is intellectueel eigendom van de auteurs en van de Universiteit van Amsterdam. Alle materiaal dat genomen is uit andere bronnen is vermeld met een bronvermelding.



Hoofdstuk 2

Introductie AI

Kunstmatige Intelligentie, of Artificial Intelligence (AI) in het Engels, is een techniek om ingewikkelde taken waar voorheen menselijke intelligentie voor nodig was door een computer te laten uitvoeren, zoals het herkennen van afbeeldingen of geluid, het leren van talen, of het maken van voorspellingen aan de hand van data.

Bekende voorbeelden zijn de chatbot ChatGPT, de photo en video generatoren Dall-E en SORA, en AI's die spellen als schaak of Go spelen¹. Zo lijkt deze ² video te zijn geschoten in een wijk in Tokyo, maar is met slechts een korte prompt door SORA gemaakt. Het model genereert talloze frames om een filmpje te maken en gebruikt bekende herkenningspunten in de stad om het net echt te laten lijken. Kan jij zien waar het in deze video toch nog niet helemaal goed gaat? Andere bekende video's waar AI in wordt gebruikt zijn *deep fakes*, waar het lijkt alsof bestaande mensen iets zeggen wat ze nooit gezegd hebben³. Er zijn natuurlijk ook talloze andere toepassingen van AI, zoals het ondersteunen van dokters in het detecteren van kankercellen op X-rays⁴, of algoritmes die biologen helpen om te begrijpen hoe eiwitten gestructureerd zijn⁵.

Om beter te begrijpen hoe een AI precies gemaakt wordt, nemen we het voorbeeld van een AI programma dat een afbeelding te zien krijgt en vervolgens kan beschrijven wat er op die afbeelding staat. Het hart van een AI is een wiskundig algoritme, oftewel een *model*, dat

¹<https://www.youtube.com/watch?v=WXuK6gekU1Y>, AlphaGo - The Movie | Full award-winning documentary

²<https://www.youtube.com/watch?v=kHwFMFu9PhA>, OpenAI Sora - Text to Video Model tokyo in the snow

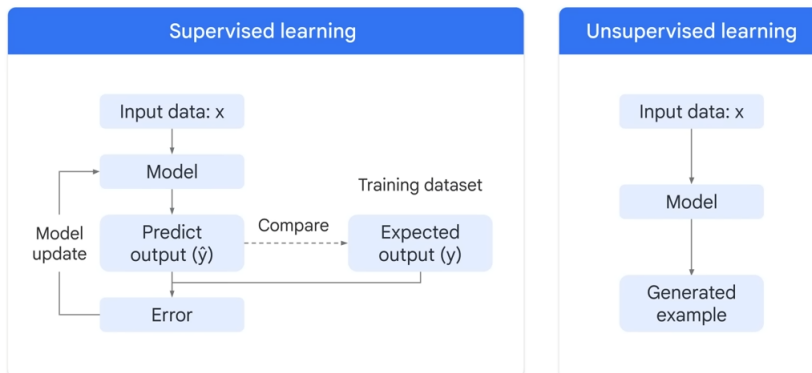
³<https://www.youtube.com/watch?v=oxXpB9pSETo>, This is not Morgan Freeman - A Deepfake Singularity

⁴<https://www.aamc.org/news/it-cancer-artificial-intelligence-helps-doctors-get-clearer-picture>

⁵<https://deepmind.google/technologies/alphafold/>

in staat is zichzelf te trainen. Door het model heel veel afbeeldingen te laten zien en erbij te melden wat voor beschrijving daarbij past, leert het model vanzelf steeds beter zijn taak uit te voeren. De afbeeldingen die we het model tijdens het trainen laten zien heet de *input data*. Als we het model vervolgens een nieuwe afbeelding laten zien zal hij de beschrijving die hij daarbij geeft vergelijken met een beschrijving van een gelijksoortige afbeelding uit de training data. Als de beschrijvingen niet genoeg op elkaar lijken zal hij daarvan leren en het de volgende keer beter doen.

Dit is een voorbeeld van wat we *supervised learning* noemen, omdat we het model eerst een hoop voorbeelden hebben gegeven van wat voor beschrijvingen correct zijn. Er zijn nog twee andere bekende manieren om AI modellen te maken: *unsupervised learning* en *reinforcement learning*. In de komende secties zullen we iets meer over deze drie technieken vertellen.



Figuur 2.0.1. <https://www.youtube.com/watch?v=G2fqAlgmoPo> - Introduction to Generative AI

2.1. Supervised Learning

Om een model te trainen hebben we een grote hoeveelheid input data nodig. Deze data vormt de *inputverzameling* van ons model: $X = \{x_1, \dots, x_N\}$. Hier spreken we van N datapunten waarbij ieder datapunt x_i een afbeelding voorstelt (x_i staat voor het i de datapunt, waarbij $i = 1, 2, \dots, N$). In de praktijk kan het best zijn dat we $N = 100000$ datapunten nemen om het model te trainen. Het model gebruikt deze datapunten om een voorspelling te doen. Voor elk datapunt komt er een voorspelling \hat{y}_i uit die hoort bij een datapunt x_i . Dit proces is te zien in figuur ?? onder het kopje "Supervised learning". Bij supervised learning is vooraf bekend wat de correcte uitvoer is van een datapunt x_i , en dus zeggen we dat de input data gelabeld is. Deze correcte uitvoer bij een datapunt x_i noemen we y_i .

In het voorbeeld kan je y_i zien als een verzameling steekwoorden die de afbeelding beschrijft. Stel dat we een afbeelding x_{10} hebben waar een bus en een rode auto op staan. In dat geval kan y_{10} gelijk zijn aan de steekwoorden "bus", "rode auto". Als we x_{10} in het model stoppen geeft het model een uitvoer, de voorspelling \hat{y}_{10} . Als het model accuraat is in het voorspellen van steekwoorden in een afbeelding kan het zijn dat \hat{y}_{10} exact voorspelt dat de steekwoorden "bus", "rode auto" zijn, maar als je het model voor de eerste keer traint is dit onwaarschijnlijk omdat het model de afbeelding voor de eerste keer ziet. In dat geval kan het zijn dat het model de voorspelling "snelweg", "blauwe auto" doet. Het doel is dat het model beter wordt in het maken van voorspellingen, dus moeten we een manier hebben om de fouten die het model maakt wiskundig te gaan optimaliseren.

Als we kijken naar een voorspelling \hat{y}_i die het algoritme als output bij een input x_i heeft gegeven, kunnen we die vergelijken met het bijbehorende label y_i . Zo kijken we naar hoeveel fouten het algoritme maakt en kunnen we bepalen hoe *accuraat* het model is. De *accuratesse*, *acc*, van een model is gedefinieerd als

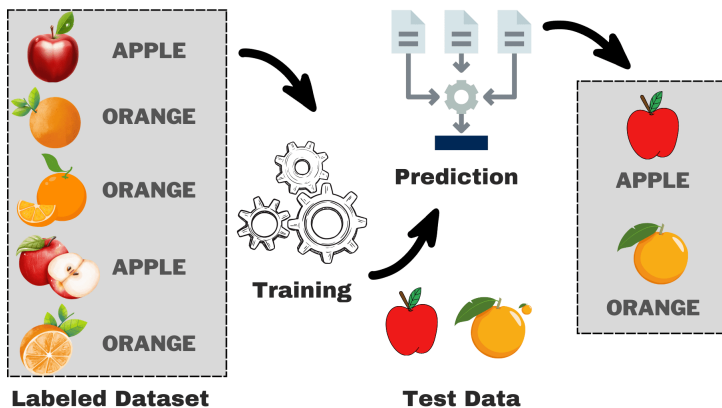
$$acc = \frac{\#(y_i = \hat{y}_i)}{N}, i = 1, \dots, N,$$

oftewel het aantal voorspellingen dat gelijk is aan de exacte uitvoer gedeeld door het totaal aantal datapunten. Met $N = 100$ datapunten en $\#(y_i = \hat{y}_i) = 10$ (10 correcte voorspellingen van de 100) bijvoorbeeld, heeft het model een accuratesse van $acc = 10/100 = 10\%$. Het doel van het herhaaldelijk trainen van het model is onder andere om een steeds accurater model te krijgen. Door de fout, of "error", te bepalen kan het model aangepast worden om beter te letten op de inputwaarden die het model fout voorspeld had.

Als het trainen van het model klaar is kan de uiteindelijke accuratesse van het model bepaald worden. Het is echter onverstandig om je model op alle datapunten te trainen en vervolgens het model te testen met diezelfde datapunten. Het model krijgt dan namelijk een accuratesse van grofweg 99.99%, maar dit geeft een vertekend beeld! Het model heeft alle data namelijk al een keer gezien tijdens het trainen. Daarom worden de N datapunten voorafgaand aan het trainen opgesplitst in *train data* en *test data*. In de praktijk wordt meestal 70% van de datapunten gebruikt voor het testen en 30% van de datapunten gebruikt voor het testen. Zo kan je met nieuwe, onbekende data zien hoe accuraat het model daadwerkelijk is.

Twee veel gebruikte toepassingen van supervised learning zijn *classificatie* en *regressie*. Deze toepassingen gebruiken het feit dat de correcte waarde y_i van een datapunt x_i bekend is. Bij classificatie gaat dit om een verzameling voorgedefinieerde klassen en bij regressie zijn dit voorspellingen van elk datapunt zoals bij het voorspellen van huizenprijzen aan de hand van eigenschappen van het huis.

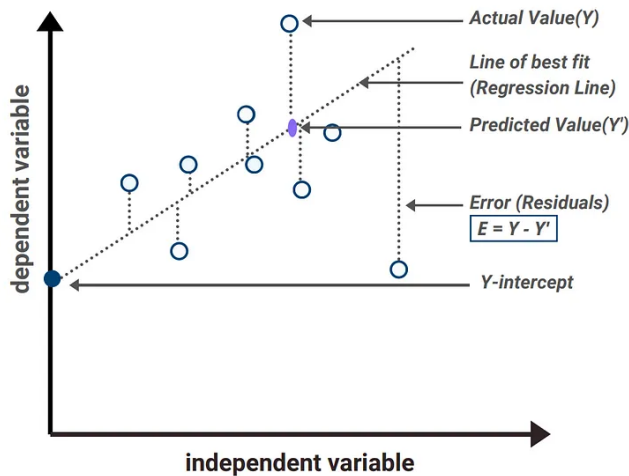
Classificatie Classificatie betreft het categoriseren van data in voorgedefinieerde klassen. In Figuur 2. staat een voorbeeld van een classificatie model. De twee voorgedefinieerde klassen zijn "apple" en "orange" en de dataset bevat afbeeldingen van verschillende appels en sinaasappels.



Figuur 2.1.1.

<https://www.kdnuggets.com/understanding-supervised-learning-theory-and-overview>

Regressie Regressie gaat over het identificeren van patronen en het maken van voorspellingen. Een typisch voorbeeld van regressie is om een lijn door een aantal datapunten te vinden, zodat de afstand van elk datapunt tot de lijn minimaal is. Een voorbeeld van lineaire regressie is te zien in Figuur 3. Bij *lineaire* regressie gaat het om een rechte lijn door een aantal datapunten, maar er zijn ook vormen van regressie waar er kromme lijnen getrokken worden, of lijnen door een meerdimensionale ruimte.



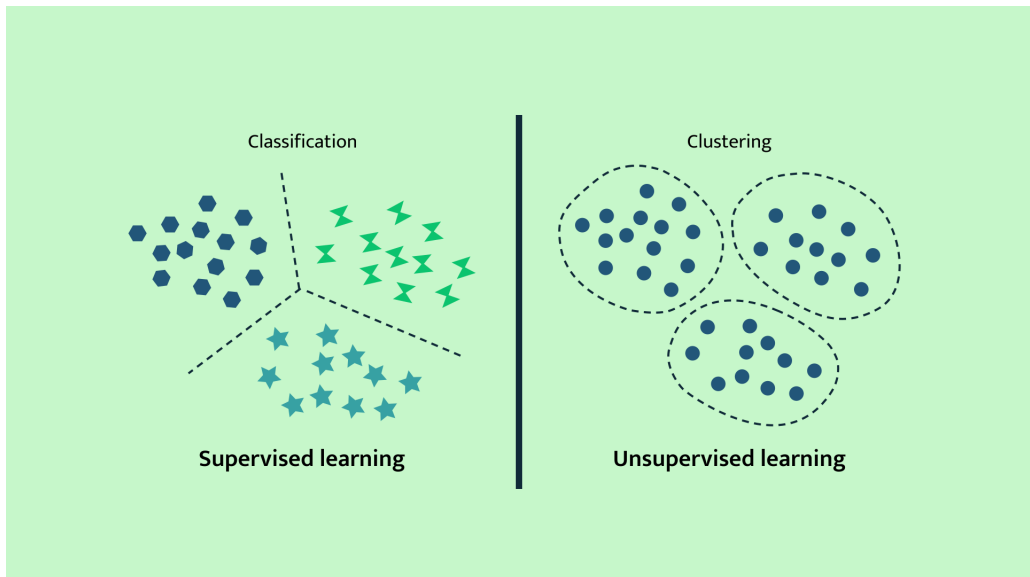
Figuur 2.1.2. <https://medium.com/@jatin2707/understanding-of-linear-regression-with-example-cf0c3b1e22e3>

2.2. Unsupervised Learning

Bij unsupervised learning is de exacte output van de datapunten niet bekend. Het algoritme kan zelf op zoek gaan naar structuur in de data. Unsupervised learning kan gebruikt worden om patronen te vinden in een onbekende dataset, zoals bij *feature learning*. Een *feature* is een meetbare eigenschap die bij de data hoort. Dit klinkt abstract, maar terugkomend op het voorspellen van huizenprijzen kunnen de features eigenschappen zijn zoals: oppervlakte van het huis, heeft het huis een tuin, het energielabel van het huis, hoeveel badkamers en slaapkamers heeft het huis, etc. Dit zijn eigenschappen die invloed hebben op hoe duur een huis is. Bij feature learning gaat een algoritme zelf de features van een dataset achterhalen. Dit heeft voordelen ten opzichte van zelf features verzamelen omdat het erg tijdrovend is om zelf features te bepalen en omdat je subtiele patronen in de data over het hoofd kan zien.

Bij unsupervised learning zijn er voornamelijk twee toepassingen: *clusteren*, oftewel het ontdekken van patronen en het groeperen van data zonder voorkennis, en *dimensionaliteitsreductie*, het reduceren van de dimensionaliteit van data. Clusteren kan op dezelfde manier toegepast worden als classificatie, zoals het categoriseren van appels en sinaasappels. Dit betekent dat het algoritme zelf moet afleiden of een afbeelding een appel of een sinaasappel bevat. Je kan je voorstellen dat gezien de kleur en vorm van een sinaasappel anders zijn dan die van een appel, het algoritme dit onderscheid opmerkt en op deze manier de data in verschillende clusters stopt. Het ene cluster bevat dan afbeeldingen van

sinaasappels en het andere cluster bevat afbeeldingen van appels. Dit is anders dan bij supervised learning waar er vooraf bekend was welke afbeelding een sinaasappel of appel bevatte en de afbeelding in de juiste categorie moest worden geplaatst. Het verschil tussen classificatie en clustering is te zien in Figuur 4.



Figuur 2.2.1.

<https://fastdatascience.com/unsupervised-learning-power-to-the-program/>

Dimensionaliteitsreductie wordt gebruikt om het aantal features (oftewel dimensies) in een dataset te reduceren. Zo wordt de dataset overzichtelijker terwijl het nog wel de essentiële kenmerken bewaart. Een algoritme dat veel gebruikt wordt voor dimensionaliteitsreductie is Principal Component Analysis (PCA). In deze techniek wordt gekeken of er lineaire combinaties van features zijn zodanig dat de integriteit van de originele dataset wordt behouden. Deze lineaire combinaties van features noemen we *principal components*.

2.3. Reinforcement learning

Reinforcement learning werkt met *agents*, wat een soort virtuele poppetjes zijn die beslissingen maken op basis van beloningen en straffen. Het doel van reinforcement learning is om een agent het maximale resultaat te laten behalen bij een opgegeven taak.

Agents worden getraind om bepaalde acties uit te voeren (door positieve en negatieve stimuli) om een doelstelling te maximaliseren in plaats van te focussen op het behalen van

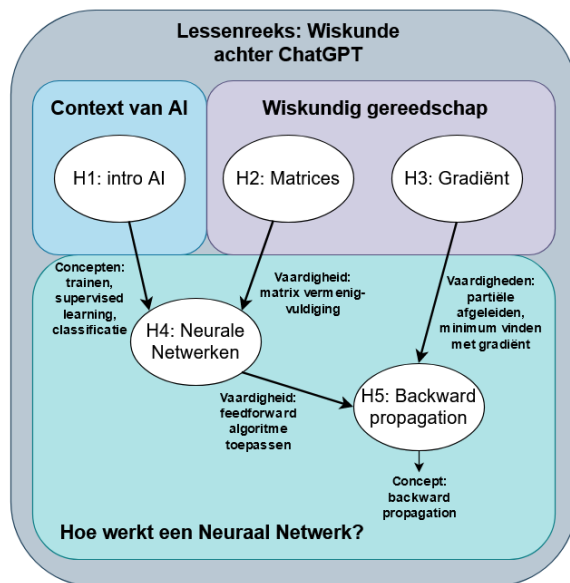
consistente resultaten op basis van gelabelde data. Er zijn twee manieren waarop agents kunnen leren; offline leren en online leren. Met online leren vergaart een agent data door te interacteren met de (virtuele) omgeving waarin die zijn taak moet uitvoeren. Dit is een herhaaldelijk proces waarbij de agent zelf data verzamelt tijdens het uitvoeren van zijn taak en die data vervolgens als feedback gebruikt om er van te leren. Met offline leren heeft een agent niet direct contact met de omgeving maar leert die aan de hand van data die eerder door een mens of een andere agent is geproduceerd. Reinforcement learning kan je mogelijk kennen van het leren rijden van auto's in een simulatie. Hier zijn er verschillende "generaties" van auto's die door vaak in de vangrail te rijden vanzelf beter gaan rijden. De feedback die ze krijgen is bijvoorbeeld dat de auto kapot gaat als die in de vangrail wordt gereden. Dit is niet gewenst gezien het doel is om de auto zo ver mogelijk te laten rijden zonder op de vangrail in te rijden. Dit is dan ook een voorbeeld van online leren omdat de agent zelf continu feedback vanuit de omgeving krijgt. Hier is nog een leuke video van Code Bullet die auto's wil laten rijden met reinforcement learning: https://www.youtube.com/watch?v=r4280_CMcpI.

2.4. Wat wij gaan doen

In deze lessenreeks houden we ons strikt bezig met supervised learning. Bij supervised learning spelen *neurale netwerken* een grote rol. Een *neuraal netwerk* is een wiskundig model dat gebruikt wordt om patronen te herkennen in een collectie van data. Een *neuraal netwerk* bestaat uit meerdere lagen die elk kleine aanpassingen doen aan de inputdata. We gaan laten zien hoe zo'n *neuraal netwerk* gedefinieerd is, hoe het werkt, hoe het getraind wordt, en hoe het gebruikt kan worden in de praktijk. Om dieper in te gaan op hoe een *neuraal netwerk* onder de motorkap werkt, bespreken we de wiskunde van matrices en partiële afgeleiden. Een *matrix* is een rechthoekig getallenschema dat data op een systematische manier

kan weergeven. Dit zal besproken worden in het volgende hoofdstuk. In het derde hoofdstuk worden *partiële afgeleiden* geïntroduceerd. Deze worden gebruikt om de *gradiënt* van een functie op te stellen, waarmee een minimum van een functie in meerdere variabelen kan worden gevonden. Verder zal in het vierde hoofdstuk het *feedforward algoritme* van een *neuraal netwerk* worden uitgelegd. Dit algoritme beschrijft hoe je inputdata door een netwerk kan sturen en en manipuleren om een output te krijgen. Ten slotte bespreken we in hoofdstuk 6 een algoritme om een *neuraal netwerk* accurater te maken, genaamd *back-propagation*. Aan het einde van de syllabus staat een aantal opgaven die gebruikt kunnen worden om te oefenen met de stof. Een overzicht van de hoofdconcepten en -vaardigheden die worden opgedaan in deze lessenreeks is te vinden in Figuur 5.

Figuur 2.4.1. Overzicht van de structuur van de syllabus.



Hoofdstuk 3

Matrices

3.1. Wat is een matrix?

Een neuraal netwerk verwerkt veel data in de vorm van getallen. Een van de gereedschappen die nodig zijn om met al deze getallen aan de slag te gaan zonder dat het allemaal te verwarrend wordt is een matrix. Een matrix kun je zien als een rechthoekig getallenschema dat het mogelijk maakt om samenhangende gegevens en hun bewerkingen op systematische wijze weer te geven. In hoofdstuk 4 gaan we verder in op hoe matrices gebruikt worden bij neurale netwerken.

Voorbeeld: Dit is een 4×3 matrix, omdat er 4 rijen en 3 kolommen zijn.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

Matrices hebben veel toepassingen. Je kunt ze bijvoorbeeld gebruiken om een expliciete formule te geven voor een recursieve rij. Een andere toepassing is het oplossen van een stelsel lineaire vergelijkingen, zoals hieronder uitgewerkt.

Toepassing: We gebruiken een matrix om een stelsel van lineaire vergelijkingen op te lossen. Dit is een globale uitleg en een gedetailleerde uitleg wordt gegeven tijdens het eerste jaar van elke WO-bèta opleiding.

Stel dat we het volgende stelsel van lineaire vergelijkingen hebben:

$$6x - 5y + 2z = 3$$

$$2x + y - 4z = 5$$

$$3x - 3y + z = -1$$

Dit kunnen we als volgt in een matrix stoppen:

$$\left[\begin{array}{ccc|c} 6 & -5 & 2 & 3 \\ 2 & 1 & -4 & 5 \\ 3 & -3 & 1 & -1 \end{array} \right].$$

De eerste kolom bevat de coëfficiënten voor de x -variabele, de tweede kolom voor de y -variabele, en de derde kolom voor de z -variabele. De laatste kolom bevat de constanten waar de lineaire combinatie van x , y , en z aan gelijk is. We kunnen deze matrix reduceren tot de volgende matrix door rij 3 te vermenigvuldigen met 2 en het resultaat van rij 1 af te trekken. Dit noemen we rijvegen, en noteren we als $R_1 - 2R_3$.

$$\left[\begin{array}{ccc|c} 0 & 1 & 0 & 5 \\ 2 & 1 & -4 & 5 \\ 3 & -3 & 1 & -1 \end{array} \right]$$

Zo zien we meteen uit de eerste rij dat $y = 5$. Op dezelfde manier kunnen we ook de x en z variabelen berekenen.

3.2. Basisoperaties op matrices

Op een matrix kun je basisoperaties toepassen zoals optellen, aftrekken, en vermenigvuldigen. Je kunt alleen matrices bij elkaar optellen of aftrekken als ze dezelfde dimensie hebben, dus de matrices moeten evenveel kolommen en evenveel rijen hebben. Om een matrix bij een andere matrix op te tellen of af te trekken, pas je de operaties elementsgewijs toe. Hoe dit werkt is hieronder in een voorbeeld te zien.

Voorbeeld: Je telt matrices op door de elementen elementsgewijs bij elkaar op te tellen.

$$\begin{bmatrix} 12 & 20 \\ 3 & 8 \end{bmatrix} + \begin{bmatrix} 8 & 5 \\ 4 & 2 \end{bmatrix} = \begin{bmatrix} 20 & 25 \\ 7 & 10 \end{bmatrix}$$

Om matrices van elkaar af te trekken, trekken we de elementen elementsgewijs af.

$$\begin{bmatrix} 5 & 12 & 3 \\ 2 & -4 & 0 \\ 1 & -3 & -5 \end{bmatrix} - \begin{bmatrix} 3 & 2 & 5 \\ 2 & -8 & -4 \\ 0 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 10 & -2 \\ 0 & 4 & 4 \\ 1 & -4 & -4 \end{bmatrix}$$

3.3. Vector of matrix?

Matrices met een enkele rij of kolom worden vaak vectoren genoemd. Hieronder staan de volgende 3d vectoren \vec{v} (een kolomvector) en \vec{w} (een rijvector):

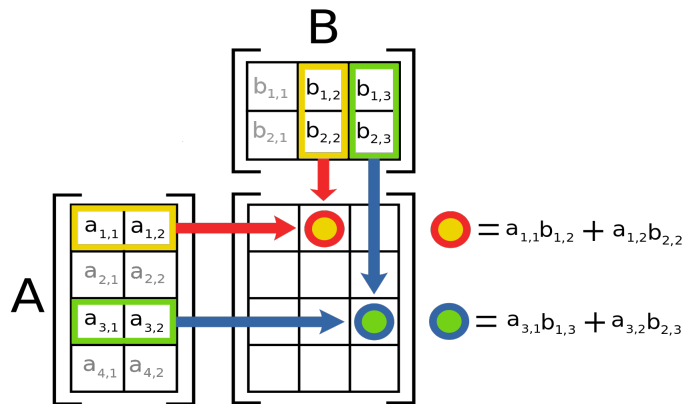
$$\vec{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \vec{w} = (1, 2, 3)$$

We kunnen \vec{v} en \vec{w} respectievelijk zien als een 3×1 matrix en een 1×3 matrix. Merk op dat de basisoperaties uit 2.2 dus ook voor \vec{v} en \vec{w} gelden. Het belangrijke om te onthouden is dat elke vector een matrix is maar niet elke matrix is een vector, en verder, dat je op vectoren matrix operaties (optellen, vermenigvuldiging, etc) kan uitvoeren.

3.4. Matrixvermenigvuldiging

De simpelste vermenigvuldiging met een matrix is een scalaire vermenigvuldiging. In dat geval vermenigvuldigen we alle elementen van de matrix met dezelfde scalar. Vermenigvuldiging van twee matrices met elkaar is echter niet zo simpel, en wordt uitgelegd in figuur 3.4.1. De figuur gebruikt standaard notatie voor het duiden van elementen in een matrix: het element $a_{i,j}$ is het getal dat op de i -de rij en j -de kolom van de matrix staat. Verder zien we dat het aantal kolommen van de eerste matrix gelijk moet zijn aan het aantal rijen van de tweede matrix. Dit betekent dat we niet zomaar elke twee matrices met elkaar kunnen vermenigvuldigen.

Voor de geel-rode cirkel bijvoorbeeld in figuur 3.4.1 nemen we $c_{1,2} = a_{1,1}b_{1,2} + a_{1,2}b_{2,2}$. De kleine tekst is hierbij steeds de index van een getal in de matrix. Het eerst getal staat voor de rij, dus hoe hoog het getal staat vanaf boven. Het tweede getal staat voor de kolom, dus hoever het getal vanaf links staat.



Figuur 3.4.1. Een schematische weergave van welke rijen en kolommen met elkaar vermenigvuldigd worden bij matrixvermenigvuldiging.

Voorbeeld: Een matrix vermenigvuldigd met een scalair.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot 5 = \begin{bmatrix} 5 & 10 \\ 15 & 20 \end{bmatrix}.$$

Voorbeeld: Een matrix vermenigvuldigd met een vector.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ae + bf \\ ce + df \end{bmatrix}.$$

Voorbeeld: Een 2×1 matrix vermenigvuldigd met een 1×2 matrix, oftewel het vermenigvuldigen van deze vectoren levert een 2×2 matrix op.

$$\begin{bmatrix} a \\ b \end{bmatrix} \cdot \begin{bmatrix} c & d \end{bmatrix} = \begin{bmatrix} ac & ad \\ bc & bd \end{bmatrix}.$$

Voorbeeld: Merk op dat het aantal kolommen van de eerste matrix gelijk is aan het aantal rijen van de tweede matrix, deze vermenigvuldiging is dus mogelijk. Een 2×2 matrix vermenigvuldigd met een 2×3 matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f & g \\ h & i & j \end{bmatrix} = \begin{bmatrix} ae + bh & af + bi & ag + bj \\ ce + dh & cf + di & cg + dj \end{bmatrix}.$$

Voorbeeld: Een 3×2 matrix vermenigvuldigd met een 2×3 matrix

$$\begin{bmatrix} 2 & 1 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 & 6 \\ 2 & 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 + 1 \cdot 2 & 2 \cdot 3 + 1 \cdot 4 & 2 \cdot 6 + 1 \cdot 5 \\ 3 \cdot 1 + 4 \cdot 2 & 3 \cdot 3 + 4 \cdot 4 & 3 \cdot 6 + 4 \cdot 5 \\ 5 \cdot 1 + 6 \cdot 2 & 5 \cdot 3 + 6 \cdot 4 & 5 \cdot 6 + 6 \cdot 5 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 10 & 17 \\ 11 & 25 & 38 \\ 17 & 39 & 60 \end{bmatrix}$$

Merk op dat als we de volgorde van de matrices omwisselen, we niet dezelfde uitkomst krijgen uit de matrixvermenigvuldiging. De volgorde is dus heel belangrijk voor matrixvermenigvuldiging:

$$\begin{bmatrix} 1 & 3 & 6 \\ 2 & 4 & 5 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 41 & 49 \\ 41 & 48 \end{bmatrix}.$$

Toepassing: In dit voorbeeld laten we zien hoe we matrices kunnen gebruiken om samenhangende data te verwerken.

Stel dat we een onderneming hebben met 2 winkels en 3 verschillende productgroepen. De ondernemer heeft 5 hamers, 2 schroeven en 0 boormachines in winkel 1. In winkel 2 heeft hij 2 hamers, 0 schroeven en 3 boormachines. Dit kunnen we als volgt in een matrix weergeven:

$$\begin{bmatrix} 5 & 2 & 0 \\ 2 & 0 & 3 \end{bmatrix}$$

Merk op dat rij 1 voor winkel 1 staat en rij 2 voor winkel 2. Verder staat kolom 1 voor de productgroep hamer, kolom 2 voor de productgroep schroeven en kolom 3 voor de productgroep boormachines. Dit is een 2×3 matrix, omdat er 2 rijen en 3 kolommen zijn. Stel dat we nu de waarde van de totale voorraad per winkel willen weten. Hiervoor hebben we de waarde van elk afzonderlijk product nodig, en die plaatsen we in een vector:

$$\begin{bmatrix} 12 \\ 2 \\ 20 \end{bmatrix}$$

We zien dus prijzen van 12, 2 en 20 euro voor een hamer, schroef en boormachine respectievelijk. Door de twee matrices met elkaar te vermenigvuldigen kunnen we de winkelvoorraden berekenen. Merk op dat we deze matrices mogen vermenigvuldigen omdat de eerste matrix 3 kolommen heeft en de tweede matrix 3 rijen.

$$\begin{bmatrix} 5 & 2 & 0 \\ 2 & 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} 12 \\ 2 \\ 20 \end{bmatrix} = \begin{bmatrix} 64 \\ 84 \end{bmatrix}$$

We zien dus dat de voorraad van winkel 1 een totale waarde van 64 euro heeft en die van winkel 2 een totale waarde van 84 euro heeft.

Hoofdstuk 4

Gradiënt

4.1. Functies in meerdere variabelen

We zijn al bekend met functies in één variabele. Voorbeelden hiervan zijn

$$f(x) = x^2, \quad g(x) = \sin(x), \quad h(y) = \ln\left(\frac{y^2 + 1}{y}\right).$$

De functies f en g hebben hier de variabele x en de functie h maakt gebruik van de variabele y . In deze functies kunnen we natuurlijk ook waarden voor onze variabelen invullen. Zo kunnen we bijvoorbeeld zien dat

$$f(2) = 2^2 = 4, \quad g(\pi) = \sin(\pi) = 0, \quad h(1) = \ln\left(\frac{1^2 + 1}{1}\right) = \ln(2).$$

We kunnen ook kijken naar functies die meer dan één variabele hebben. Bij zulke functies kunnen we twee of meer waarden tegelijk invullen.

Voorbeeld:

We definiëren de functie f in twee variabelen x en y . We kunnen nu schrijven dat

$$f(x, y) = x + y.$$

We hebben dus een functie die als input x en y neemt en als output hun som geeft. We kunnen nu verschillende waardes voor x en y invullen in deze functie. Een aantal voorbeelden zijn

$$f(1, 1) = 1 + 1 = 2, \quad f(4, 8) = 4 + 8 = 12, \quad f\left(\frac{3}{2}, -1\right) = \frac{3}{2} - 1 = \frac{1}{2}.$$

Toepassing:

Functies in meerdere variabelen komen heel veel voor in de praktijk, bijvoorbeeld in de natuurkunde. De formule voor zwaartekracht wordt bijvoorbeeld gegeven door

$$F_z = m \cdot g.$$

Deze formule is eigenlijk een formule in meerdere variabelen, namelijk m en g . Wiskundig gezien zouden we dan

$$F_z(m, g) = m \cdot g$$

schrijven.

Bij een functie in meerdere variabelen is het belangrijk om te kijken op welke plek we een waarde in de functie schrijven. Als we bijvoorbeeld kijken naar de functie $f(x, y)$ zal de eerste waarde altijd naar de x worden gestuurd en de tweede waarde altijd naar de y worden gestuurd.

Voorbeeld: We definiëren de functie g in vier variabelen, als volgt:

$$g(a, b, c, d) = ab + c^2 d^2.$$

Dit is een functie in de variabelen a, b, c en d . Ook hier kunnen we verschillende waardes invullen. Zo krijgen we bijvoorbeeld dat

$$g(1, 1, 1, 1) = 1 + 1 = 2, \quad g(2, 3, 1, 2) = 6 + 4 = 10.$$

4.2. Partiële afgeleiden

Net zoals bij functies in één variabele kunnen we ook bij een functie in meer variabelen kijken naar de afgeleide. Hier zit echter wel een aantal ogen en haken aan. We beginnen met de partiële afgeleide. Zoals de naam misschien wel aangeeft, geeft deze geen totaal beeld van hoe de afgeleide van de functie er uit ziet. Bij een partiële afgeleide van een functie f kijken we eigenlijk naar de afgeleide ten opzichte van één van de variabelen van de functie tegelijk.

Definitie: De partiële afgeleide van een functie $f(x_1, x_2, \dots, x_n)$, naar de variabele x_i is genoteerd als

$$\partial_{x_i} f(x_1, x_2, \dots, x_n).$$

Een andere manier om de partiële afgeleide te schrijven is

$$\frac{\partial}{\partial x_i} f(x_1, x_2, \dots, x_n).$$

Dit is de afgeleide naar de variabele x_i waarbij de andere variabelen als constanten worden beschouwd.

We geven een paar voorbeelden.

Voorbeeld: Neem $f(x, y) = x + y^2$. We bekijken eerst $\partial_x f$:

$$\partial_x f(x, y) = \partial_x (x + y^2) = \partial_x (x) + \partial_x (y^2) = 1 + 0 = 1.$$

Dan bekijken we $\partial_y f$:

$$\partial_y f(x, y) = \partial_y (x + y^2) = \partial_y (x) + \partial_y (y^2) = 0 + 2y = 2y.$$

Voorbeeld: We kunnen ook de kettingregel gebruiken bij partiële afgeleiden. Neem de functie

$$f(x, y) = (3xy + 3)^2 + xy.$$

Als we de afgeleide naar x nemen, beschouwen we alle y 'tjes als constanten, dus

$$\partial_x f(x, y) = 2 \cdot 3y(3xy + 3) + y = 6y(3xy + 3) + y.$$

We kunnen ook de afgeleide naar y bepalen:

$$\partial_y f(x, y) = 6x(3xy + 3) + x.$$

4.3. Gradiënt van functies

De afgeleide van een functie in één variabele is vrij goed bekend. Hiermee kunnen we de richtingscoëfficiënt van een raaklijn op een functie bepalen. Als we iets vergelijkbaars doen voor een functie in meer variabelen krijgen we een soort raaklijn in meerdere dimensies. Hiervoor introduceren we de **gradiënt** van een functie f in meerdere variabelen; genoteerd als ∇f .

Definitie: De gradiënt van een functie $f(x_1, x_2, \dots, x_n)$ is een vector met alle partiële afgeleiden erin. Deze kunnen we als volgt opschrijven

$$\nabla f(x_1, x_2, \dots, x_n) = (\partial_{x_1} f, \partial_{x_2} f, \dots, \partial_{x_n} f).$$

De gradiënt van een functie geeft als output een vector die in de richting staat van de grootste toename van de functie f vanaf een zeker punt.

Voorbeeld: We beschouwen de functie $f(x, y) = xy^2 - 3xy$. Dit is een functie in twee variabelen. We gaan nu eerst alle partiële afgeleiden bepalen. We vinden

$$\partial_x f(x, y) = y^2 - 3y, \quad \partial_y f(x, y) = 2xy - 3x.$$

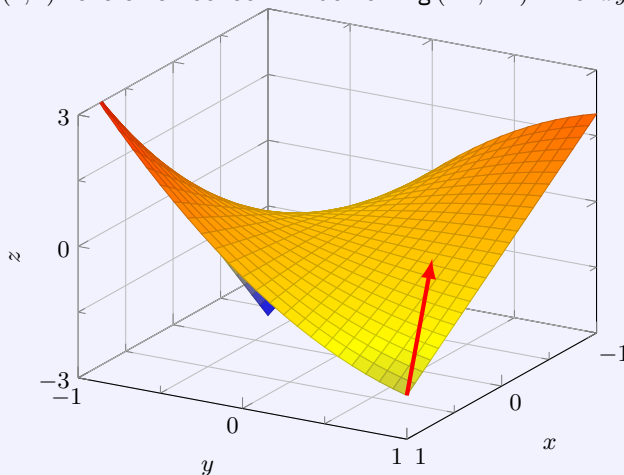
Stel dat we willen weten in welke richting de functie f het sterkst toeneemt in het punt $(1, 1)$. Daartoe bepalen we de gradiënt. Dit is een vector met alle partiële afgeleiden van de functie $f(x, y)$ erin. We krijgen dan dus

$$\nabla f(x, y) = (y^2 - 3y, 2xy - 3x).$$

Merk in dit voorbeeld op dat de gradiënt eigenlijk ook een functie is. Het grote verschil met functies die we eerder hebben gezien is dat we nu in plaats van een enkel getal als output, een vector als output krijgen. We kunnen nu bepalen in welke richting de functie $f(x, y)$ het sterkst toeneemt vanaf het punt $(1, 1)$. Dit is namelijk

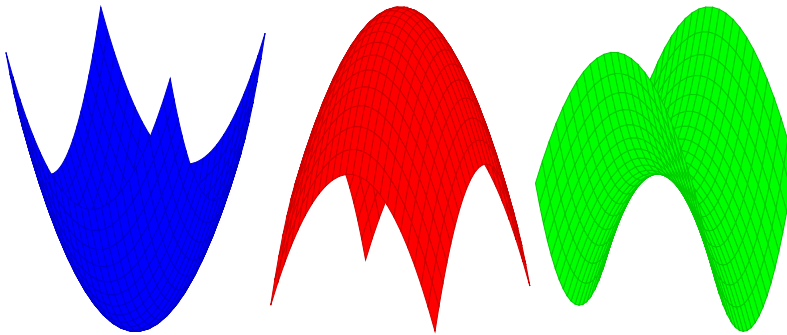
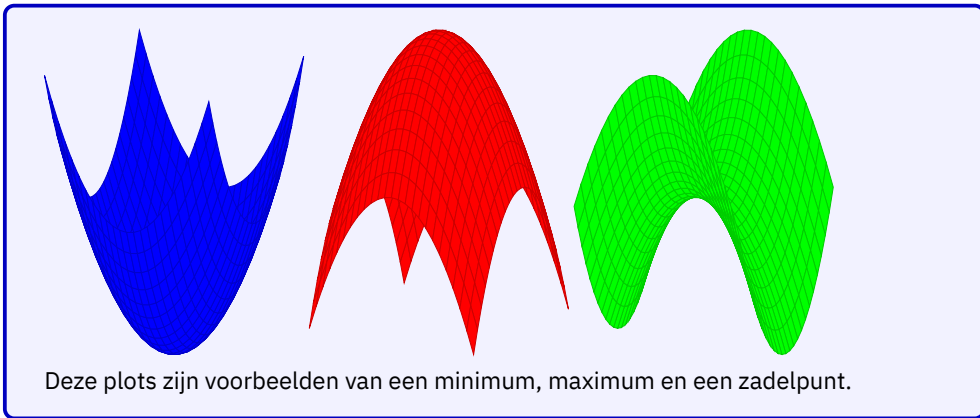
$$\nabla f(1, 1) = (1 - 3, 2 - 3) = (-2, -1).$$

Hieruit kunnen we concluderen dat de functie $f(x, y)$ in het punt $(1, 1)$ het sterkst toeneemt in de richting $(-2, -1)$ in het xy -vlak.



Met behulp van de gradiënt is het ook mogelijk om de extreme waarden van een functie te bepalen. Dit doen we door alle waarden in de gradiënt gelijk te stellen aan 0. Extreme waarden in meer variabelen nemen niet altijd de vorm aan van een maximum of een minimum, maar deze kunnen ook een zogenaamd zadelpunt zijn. Dit zijn punten die in één variabele een maximum zijn, maar in een andere variabele een minimum. Voor neurale netwerken

kijken we niet naar functies die een zadelpunt hebben, dus zullen we hier verder niet veel aandacht aan besteden.



Voorbeeld: We kijken weer even terug naar het vorige voorbeeld. We hebben de gradiënt al gevonden. We willen nu kijken voor welke waarden voor x en y , elke index van de gradiënt 0 is. We vinden dan de vergelijkingen

$$y(y - 3) = 0 \quad \text{en} \quad 2xy - 3x = 0.$$

Uit de linkervergelijking vinden we dat $y = 0$ of $y = 3$. $y = 0$ invullen in de rechtervergelijking geeft $-3x = 0$, dus $x = 0$. Hiermee hebben we gevonden dat $(0, 0)$ een extreme waarde is van $f(x, y)$. $y = 3$ invullen geeft ons dat $3x = 0$, dus wordt de andere extreme waarde gegeven door $(0, 3)$.

4.4. Het Gradient descent algoritme

Met behulp van de gradiënt is het nu mogelijk om gebruik te maken van een algoritme dat *gradient descent* heet. Gradient descent is een algoritme dat een lokaal minimum van een functie probeert te vinden en wordt veel gebruikt bij neurale netwerken. Een neurale netwerk maakt namelijk gebruik van een functie die de *cost-functie* heet. Deze functie geeft een getal die ons zegt hoe slecht ons neurale netwerk op een bepaalde inputwaarde reageert. Bij het trainen van een neurale netwerk willen we natuurlijk zo goed mogelijke resultaten krijgen, dus proberen we een minimum van de cost-functie te bepalen. Om dat minimum te bepalen wordt het gradient descent algoritme gebruikt.

4.5. Hoe werkt het algoritme?

Gradient descent laat als het ware een balletje van een heuvel af rollen. Waar het balletje begint met rollen noemen we (x_0, y_0) , het beginpunt. Het algoritme bepaalt vervolgens de richting waar het balletje naartoe rolt, wat de richting van de steilste helling naar beneden vanaf het beginpunt is. De gradiënt van een functie geeft de richting van de grootste toename, dus om de richting van de grootste afname te vinden hoeven we deze waarde alleen maar om te draaien. Vervolgens lopen we een stapje in deze richting. Dit proces herhalen we totdat we het minimum gevonden hebben. We zullen nu in wiskundig detail toelichten hoe gradient descent werkt.

Definitie: Stel dat we een minimum willen vinden van de functie $f(x, y)$. Het algoritme zal een rijtje (x_n, y_n) van coördinaten geven waarvoor geldt dat als we verder in het rijtje kijken (hogere waarde van n), we dichterbij een lokaal minimum komen. Hiervoor moeten we wel een startwaarde (x_0, y_0) bepalen. Vervolgens definiëren we nu het rijtje als volgt

$$(x_{n+1}, y_{n+1}) = (x_n, y_n) - \gamma \nabla f(x_n, y_n).$$

Hier is γ een positieve parameter die we de *leersnelheid* noemen.

De parameter γ bepaalt hoe lang we langs de richting van de gradiënt bewegen voordat we opnieuw de gradiënt berekenen voor een nieuw stel coördinaten in het rijtje (x_n, y_n) . Hoe kleiner γ is, hoe nauwkeuriger het algoritme is, maar ook hoe langzamer.

Voorbeeld: We beschouwen de functie $f(x, y) = x^2 + y^2$. We kunnen intuïtief inzien dat het minimum van deze functie op het punt $(0, 0)$ ligt omdat een kwadraat altijd positief is en deze functie enkel gelijk is aan 0 op het punt $(0, 0)$. We nemen nu een leersnelheid van $\gamma = 0.3$ en we kiezen het startpunt $(x_0, y_0) = (5, 5)$. De gradiënt van f wordt gegeven door

$$\nabla f(x, y) = (2x, 2y).$$

We krijgen:

$$x_1 = (5, 5) - 0.3 \cdot (10, 10) = (5, 5) - (3, 3) = (2, 2)$$

$$x_2 = (2, 2) - 0.3 \cdot (4, 4) = (2, 2) - (1.2, 1.2) = (0.8, 0.8).$$

Door deze actie een aantal iteraties uit te voeren krijgen we de volgende tabel:

x_0	(5, 5)
x_1	(2, 2)
x_2	(0.8, 0.8)
x_3	(0.32, 0.32)
x_4	(0.13, 0.13)

Hieruit is te zien dat de waardes snel naar $(0, 0)$ toegaan.

(x_0, y_0)	(5, 5)
(x_1, y_1)	(2, 2)
(x_2, y_2)	(0.8, 0.8)
(x_3, y_3)	(0.32, 0.32)
(x_4, y_4)	(0.13, 0.13)

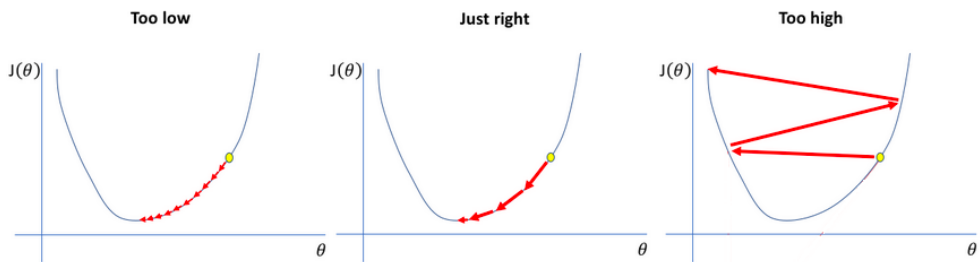
4.5.1. Nadelen van gradient descent

Hoewel gradient descent een heel fijn algoritme is, is er wel een aantal situaties waar het algoritme niet perfect werkt.

Ten eerste hangt het succes van het algoritme af van het gekozen beginpunt. Voor complexere functies met meerdere lokale minima kan een verkeerd beginpunt er namelijk voor zorgen dat het algoritme naar het verkeerde minimum zal convergeren. Daarom is het altijd belangrijk om een startpunt te kiezen wat vermoedelijk dicht bij het te vinden minimum ligt.

Ook kan het verkeerd kiezen van de leersnelheid grote gevolgen hebben op de functionaliteit het algoritme. Een hogere leersnelheid zorgt ervoor dat het minimum sneller wordt bereikt, maar het nadeel is dat het mogelijk is om over het minimum heen te schieten. Probeer maar eens een leersnelheid van $\gamma = 2$ te gebruiken in het voorbeeld met $f(x, y) = x^2 + y^2$ en zie dat we in dit geval juist verder weg zullen gaan van het minimum. In figuur 4.5.1 is te zien wat de gevolgen van verschillende leersnelheden op de uitkomst van gradient descent

zijn.



Figuur 4.5.1. Verschillende keuzes van leersnelheid bij gradient descent (Datacamp).

Hoofdstuk 5

Neurale netwerken en het feed forward algoritme

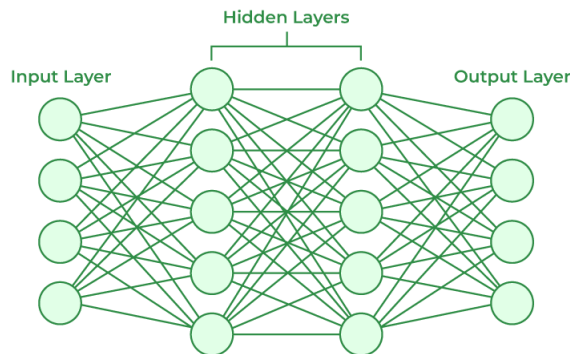
Nu we al deze wiskunde hebben ontwikkeld, kunnen we beginnen aan neurale netwerken. Het doel van dit hoofdstuk is te beschrijven hoe een neurale netwerk eruitziet en hoe we van een input tot een output komen.

5.1. Het netwerk

Om te begrijpen hoe een computer kan “nadenken” moeten we eerst even kijken naar hoe een mens dit doet. In ons hoofd zitten neuronen die door een netwerk met elkaar zijn verbonden. Deze neuronen kunnen activeren als ze bepaalde signalen krijgen van hun burens. Eens geactiveerd geeft de neuron het signaal door. Dit systeem gaan wij nabootsen op de computer.

Een **netwerk** bestaat uit een aantal lagen van **neuronen**. Een neuron is verbonden met alle neuronen van de vorige laag. Deze verbindingen noemen we de **gewichten** (Zie figuur 5.1.1). Elke neuron is verbonden met elke neuron van de vorige en volgende laag.

We geven een invoer die even groot is als het aantal neuronen in de invoer laag. Verder kunnen we zelf bepalen hoe groot de uitvoer laag moet zijn. Ook beslissen we zelf hoe we de uitvoer interpreteren. De lagen die tussen de input en de output lagen zitten noemen we *verborgen lagen*, of *hidden layers*, en die worden gebruikt om de gewichten aan te passen totdat we bij een gewenste output komen.



Figuur 5.1.1. Structuur van een Netwerk (GeeksforGeeks)

Toepassing: Stel dat we ons netwerk naar zwart-wit-afbeeldingen met 128×128 pixels willen laten kijken om cijfers in de afbeeldingen te herkennen. Dan moeten we een invoerlaag hebben van $128 \times 128 = 16384$ neuronen, waar elke pixel in het plaatje gekoppeld zal zijn aan één enkele neuron. Met het aantal lagen en de hoeveelheid neuronen in de verborgen lagen kunnen we spelen. Bij het plaatje willen we natuurlijk zoveel mogelijk details van het plaatje oppikken. Hierdoor zouden we ervoor kunnen kiezen om een groot aantal neuronen in de verborgen laag te zetten. We kunnen ook één enkele neuron in de verborgen laag zetten, maar hierdoor kunnen we nooit alle details van het plaatje oppikken. Als we dan de cijfers 0 tot 9 willen classificeren kunnen we 10 neuronen in de laatste laag zetten. De neuron die de hoogste waarde heeft geeft dan aan wat het netwerk denkt dat er in het plaatje te zien is.

Toepassing: Een ander voorbeeld is een netwerk voor een auto, waar we van een specifieke neuron in de uitvoerlaag een waarde tussen ± 1 lezen. We zeggen dan bijvoorbeeld dat dit het sturingspercentage is dat de auto moet sturen.

5.2. In de praktijk

Elke laag neuronen kunnen we representeren als een vector, waar elk getal in de vector één neuron voorstelt. De gewichten tussen de lagen kunnen we dan met matrices representeren. Om te bepalen hoe sterk de neuronen in een bepaalde laag activeren gebruiken we de neuronen van de vorige laag en vermenigvuldigen deze met de gewichten ertussen. Dit is

waarom het algoritme feedforward heet: de berekeningen gaan van links naar rechts door het netwerk heen. We gebruiken steeds de vorige laag om te bepalen hoe de volgende er uit ziet.

Voorbeeld: We hebben een netwerk met 3 lagen van 2 neuronen. Dan ziet het netwerk er dus zo uit (let op, dit is geen vermenigvuldiging maar alleen een weergave van het netwerk):

$$\begin{bmatrix} \text{in}_1 \\ \text{in}_2 \end{bmatrix} \begin{bmatrix} v_1 & v_2 \\ v_3 & v_4 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} \text{out}_1 \\ \text{out}_2 \end{bmatrix}.$$

In deze vergelijking staan in_1, in_2 voor de inputneuronen, de v - en w -waardes staan voor de gewichten, de h_1 - en h_2 -waardes staan voor de verborgen neuronen en out_1 en out_2 staan voor de outputneuronen. We willen nu $\text{out}_1, \text{out}_2$ bepalen gegeven in_1, in_2 . De verborgen neuronen vinden we door de input neuronen te vermenigvuldigen met de gewichten in de eerste matrix:

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 \\ v_3 & v_4 \end{bmatrix} \begin{bmatrix} \text{in}_1 \\ \text{in}_2 \end{bmatrix}.$$

Net zo kunnen we $\text{out}_1, \text{out}_2$ bepalen:

$$\begin{bmatrix} \text{out}_1 \\ \text{out}_2 \end{bmatrix} = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}.$$

Om de complexiteit van het neurale netwerk te verhogen kunnen we per laag bepaalde constanten aan de neuronen toevoegen. Dit doen we door een vector van **biases** op te tellen bij de vector van die laag. Een tweede manier om het neurale netwerk verder uit te breiden is door middel van de **activeringsfunctie**. Dit is een functie die wordt toegepast op elke neuron in een laag en kan bijvoorbeeld $f(x) = \sin x$ of $f(x) = \max(0, x)$ zijn, waar x de waarde voor één neuron in de laag is. De uitkomst van de functie is dan weer een nieuwe vector. Voor het gemak schrijven we vaak $f(\vec{v})$ met \vec{v} een vector wanneer we bedoelen dat f op elke rij in de vector wordt toegepast.

Voorbeeld: We gebruiken hetzelfde netwerk als bij het eerste voorbeeld maar nu gebruiken we een activeringsfunctie k en biases

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \quad \begin{bmatrix} b_3 \\ b_4 \end{bmatrix}.$$

We willen weer out_1, out_2 bepalen voor gegeven in_1, in_2 . Hier voor bepalen we weer eerst h_1, h_2 we hebben

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = k \left(\begin{bmatrix} v_1 & v_2 \\ v_3 & v_4 \end{bmatrix} \begin{bmatrix} in_1 \\ in_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right).$$

Dan kunnen we nu out_1, out_2 bepalen door

$$\begin{bmatrix} out_1 \\ out_2 \end{bmatrix} = k \left(\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + \begin{bmatrix} b_3 \\ b_4 \end{bmatrix} \right).$$

Nu we een beeld hebben van een klein netwerk kunnen we beschrijven hoe grotere netwerken er uit zien.

Definitie: Voor een netwerk van n lagen zeggen we dat

$$[l_1, l_2, \dots, l_n]$$

de lagen van het netwerk zijn. Dan staat l_i voor het aantal neuronen in laag i .

Verder beschrijven we ook formeel hoe de vectoren en matrices er uit moeten zien voor een bepaald netwerk.

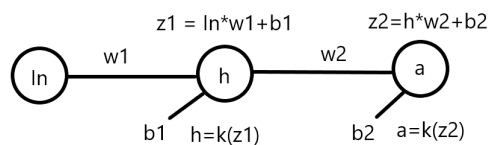
Definitie: Voor een netwerk met lagen $[l_1, l_2, \dots, l_n]$, bestaat laag i uit een vector van grootte l_i . Voor twee lagen l_i, l_{i+1} worden de gewichten ertussen beschreven door een matrix $G^{i,i+1}$ van grootte $l_{i+1} \times l_i$. In deze gewichtenmatrix beschrijft $g_{n,m}$ het gewicht tussen node m van laag i en node n van laag $i + 1$.

Nu kunnen we elk formaat van netwerk beschrijven met de wiskunde die we tot nu toe hebben gezien. In het volgende hoofdstuk gaan we in op het bepalen van de juiste gewichten en biases, zodat het netwerk de juiste uitvoer geeft.

Hoofdstuk 6

Het backpropagation algoritme

In dit hoofdstuk bekijken we hoe biases en gewichten een rol spelen in het trainen van een neuraal netwerk. Hiervoor beschouwen we een simpel netwerk beschreven in figuur 6.0.1, met gewichten w_1, w_2 , biases b_1, b_2 en activeringsfunctie k .



Figuur 6.0.1. Een simpel neuraal netwerk. Hier zijn z_1 en z_2 hulpfuncties die later de berekening zullen versimpelen.

6.1. De Cost-functie

Het idee is dat we door de gewichten en biases van het neurale netwerk aan te passen een nauwkeuriger model kunnen verkrijgen. Daarom is het handig om eerst een manier te vinden om de nauwkeurigheid van het netwerk te meten. Hiervoor definiëren we de cost-functie:

$$C = (a - y)^2,$$

waar y de waarde is die het netwerk had moeten geven voor de gegeven invoer en a de uitvoer van het netwerk. Deze cost hangt natuurlijk af van alle gewichten en biases, omdat a hier van afhangt. Als de uitvoer a erg lijkt op wat we willen van het netwerk y , dan is deze functie klein. Om het netwerk zo nauwkeurig mogelijk te krijgen willen we de cost-functie dus gaan minimaliseren.

6.2. De aanpassing bepalen

Om de cost-functie te minimaliseren gebruiken we het gradient-descent algoritme. Hiervoor moeten we de gradiënt van de cost-functie $C(w_1, w_2, b_1, b_2)$ als functie in meerdere variabelen afhankelijk van de gewichten en biases bepalen. Hiervoor hebben we de partiële afgeleides nodig van C .

We beginnen met het bepalen van $\frac{\partial C}{\partial w_2}$. Om de berekeningen wat overzichtelijker te houden definiëren we de hulpfunctie $z_2 = hw_2 + b_2$. Dan krijgen we namelijk

$$a = k(z_2), \quad \text{i.p.v.} \quad a = k(hw_2 + b_2).$$

Om $\frac{\partial C}{\partial w_2}$ te berekenen gebruiken we de kettingregel:

$$\frac{\partial C}{\partial w_2} = \frac{\partial z_2}{\partial w_2} \frac{\partial a}{\partial z_2} \frac{\partial C}{\partial a}.$$

Deze factoren aan de rechter kant kunnen we allemaal bepalen. We krijgen

$$\frac{\partial C}{\partial a} = 2(a - y), \quad \frac{\partial a}{\partial z_2} = k'(z_2), \quad \frac{\partial z_2}{\partial w_2} = h.$$

Het resultaat is een relatief makkelijke functie, waarvan we alle waarden kennen om het in te vullen.

$$\frac{\partial C}{\partial w_2} = h \cdot k'(z_2) \cdot 2(a - y)$$

Hetzelfde trucje gaan we nu nog een aantal keer herhalen voor $\frac{\partial C}{\partial b_2}$ en $\frac{\partial C}{\partial h}$. Dit geeft

$$\frac{\partial C}{\partial b_2} = \frac{\partial z_2}{\partial b_2} \frac{\partial a}{\partial z_2} \frac{\partial C}{\partial a}$$

met

$$\frac{\partial C}{\partial a} = 2(a - y), \quad \frac{\partial a}{\partial z_2} = k'(z_2), \quad \frac{\partial z_2}{\partial b_2} = 1.$$

We hebben van de achterste laag nu compleet bepaald hoe de cost functie afhangt van de relevante variabelen. We gaan door naar de eerste laag. Hier krijgt het backpropagation

algoritme zijn naam van: we gaan van achter naar voor. We bepalen hoe de cost-functie afhangt van w_1 :

$$\begin{aligned}\frac{\partial C}{\partial w_1} &= \frac{\partial z_1}{\partial w_1} \frac{\partial h}{\partial z_1} \frac{\partial C}{\partial h} \\ &= I \cdot k'(z_1) \cdot \frac{\partial C}{\partial h}.\end{aligned}$$

Met de kettingregel kunnen we ook de laatste term bepalen:

$$\begin{aligned}\frac{\partial C}{\partial h} &= \frac{\partial z_2}{\partial h} \frac{\partial a}{\partial z_2} \frac{\partial C}{\partial a} \\ &= w_2 k'(z_2) 2(a - y).\end{aligned}$$

We krijgen uiteindelijk dus

$$\frac{\partial C}{\partial w_1} = I \cdot k'(z_1) \cdot w_2 k'(z_2) 2(a - y).$$

Als laatste berekenen we $\frac{\partial C}{\partial b_1}$:

$$\begin{aligned}\frac{\partial C}{\partial b_1} &= \frac{\partial z_1}{\partial b_1} \frac{\partial h}{\partial z_1} \frac{\partial C}{\partial h} \\ &= k'(z_1) \frac{\partial C}{\partial h} \\ &= k'(z_1) \cdot w_2 k'(z_2) 2(a - y).\end{aligned}$$

Dan kunnen we nu een klein stapje zetten in de $-\nabla C$ richting, met leersnelheid γ . Dit doen we door eerst een startpunt voor elke variabele te kiezen. Vervolgens passen we iteratief de volgende formules toe:

$$w_{i,new} = w_i - \gamma \frac{\partial C}{\partial w_i}(w_i), \quad i = 1, 2$$

en

$$b_{i,new} = b_i - \gamma \frac{\partial C}{\partial b_i}(b_i), \quad i = 1, 2.$$

De eerste keer vullen we dus de startwaardes in voor w_i en b_i . Daarna vullen we telkens de nieuw verkregen $w_{i,new}$ en $b_{i,new}$ opnieuw de formule in. Op deze manier komen we bij een lokaal minimum van de cost-functie, en vinden we dus de gewichten en biases waarvoor het neurale netwerk zo goed mogelijk werkt.

Hoofdstuk 7

Opgaven

7.1. Matrices

Opgave 1 Bereken of geef aan als de operatie niet mogelijk is.

$$(i) \begin{bmatrix} 2 & -6 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 4 \\ -1 & 9 \end{bmatrix}$$

$$(ii) \begin{bmatrix} 2 & 4 \\ 3 & 8 \end{bmatrix} + \begin{bmatrix} 1 & 8 \\ 0 & 6 \\ 2 & 4 \end{bmatrix}$$

$$(iii) \begin{bmatrix} 3 & 1 & 7 \\ 1 & 5 & 8 \\ 0 & 9 & 4 \end{bmatrix} - \begin{bmatrix} 4 & 0 & 3 \\ 8 & 3 & 4 \\ 2 & 3 & 4 \end{bmatrix}$$

Opgave 2 Bereken of geef aan als de operatie niet mogelijk is.

$$(i) \begin{bmatrix} 2 & 9 \\ 8 & 4 \end{bmatrix} \cdot 5$$

$$(ii) \begin{bmatrix} 2 & 5 \\ 3 & 8 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 4 & 1 \end{bmatrix}$$

$$(iii) \begin{bmatrix} 7 & 2 & 3 \\ 1 & 0 & -2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 2 & -1 \\ 4 & -2 \end{bmatrix}$$

$$(iv) \begin{bmatrix} 6 & 8 \\ 5 & 4 \\ 7 & 10 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 10 & 0 & -6 \end{bmatrix}$$

$$v) \begin{bmatrix} 6 & 7 & 3 \\ 2 & 1 & 5 \\ 3 & 0 & 6 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 7 & 2 \\ 3 & 8 \\ 1 & 0 \end{bmatrix}$$

Opgave 3 Laat zien dat voor

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix},$$

$AB \neq BA$.

Opgave 4 Matrices werken anders dan normale getallen. Zo kunnen we niet zomaar zeggen dat $AB = BA$. Wel zouden we graag willen hebben dat $(AB)C = A(BC)$. Laat zien dat dit het geval is voor de matrices

$$A = \begin{bmatrix} 0 & 3 \\ 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \quad C = \begin{bmatrix} 2 & 1 \end{bmatrix}.$$

Dit is in het algemeen ook waar, maar dat hoeft je niet te bewijzen.

Opgave 5 Voor matrices geldt ook $A(B+C) = AB+AC$. Let wel dat de A steeds aan de linkerkant blijft. Laat dit zien voor

$$A = \begin{bmatrix} 0 & 3 \\ 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 2 \\ 0 & 3 \end{bmatrix}, \quad C = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}.$$

7.2. Gradiënt

Opgave 1: Bepaal alle partiële afgeleiden van de volgende functies:

- (i) $f(x, y) = x^2 + 3xy + y^2$,
- (ii) $f(x, y) = \sin(x) \cos(y)$,
- (iii) $f(x, y) = e^{x^2y+y}$,
- (iv) $f(x, y) = x^y$.
- (v) $f(x, y) = x^y y$.

Opgave 2: We hebben een functie $C(a(x), y)$. Hier in is y een waarde, maar hangt a af van x . Bepaal $\partial C / \partial x$, voor de volgende functies $C(a(x), y)$ en $a(x)$

- (i) $C = a(x) + y$, $a(x) = \sin(x)$.
- (ii) $C = (a(x) - y)^2$, $a(x) = 3x$.
- (ii) $C = \sin(ya(x))$, $a(x) = yx$.

[Hint: gebruik de kettingregel]

Opgave 3: We hebben een functie $C(a(x), y(x))$. Bepaal $\partial C / \partial x$, voor de volgende functies $C(a(x), y(x))$, $a(x)$ en $y(x)$

- (i) $C = a + y$, $a(x) = \sin(x)$, $y(x) = 3x$.

(ii) $C = ay$, $a(x) = 3x$, $y(x) = 4x^2$.

Opgave 4: Bepaal de gradiënt van de volgende functies.

- (i) $f(x, y) = x^2y$.
- (ii) $f(x, y) = x^3$.
- (iii) $f(x, y, z) = \sin(x^2z) \cos(y^2z)$.

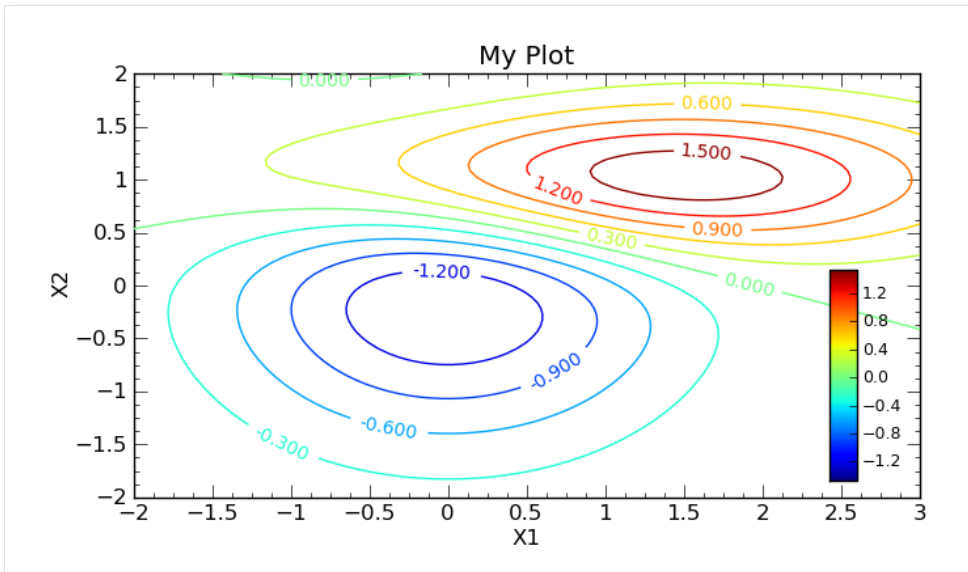
Opgave 5: In de volgende opgave maken we gebruik van de functie

$f(x) = x^4 - 2x^2 + 1$. De grafiek van $f(x)$ heeft twee dalen. In deze opgave gaan we proberen met behulp van Gradient Descent het linker dal van de grafiek van $f(x)$ te benaderen.

- (i) Bereken de gradiënt van $f(x)$.
- (ii) Maak een nette schets van $f(x)$. Zorg hierbij dat de minima en maxima goed zijn (Bereken deze dus ook met behulp van de afgeleide).
- (iii) We gaan nu Gradient Descent toepassen. Neem $x_0 = -2$. We kiezen de stapgrootte $\gamma = \frac{1}{20}$. Voer 3 stappen uit van Gradient Descent voor deze beginwaarde. Teken hierbij ook steeds de punten die gevonden worden.
- (iv) Pas 3 stappen toe van Gradient Descent op $f(x)$ met $x_0 = 0$ en $\gamma = 1$. Wat valt je op?
- (v) Pas 3 stappen toe van Gradient Descent op $f(x)$ met $x_0 = -2$ en $\gamma = \frac{1}{10}$. Wat valt je op?

Bonusopgave: In deze opgave gaan we iets maken wat een contourplot heet. Een contourplot is de manier om een 3D-oppervlak weer te geven in 2D. Een contourplot wordt veel gebruikt voor hoogtekaarten in bijvoorbeeld een atlas. Een voorbeeld van een contourplot is hieronder te zien in Figuur 7.2.1. De getallen in de lijnen betekenen dat alle inputwaardes op deze lijnen als output van een zekere functie die waarde geven die op de lijn staat. Voor deze opgave gaan we een contourplot maken voor de functie $f(x, y) = -\cos(x) - \sin(y)$ op een assenstelsel wat van ongeveer -5 tot 5 gaat over beide assen.

- (i) Voor alle waardes $k = -2, -1, 0, 1, 2$, teken alle contourlijnen voor $f(x, y) = k$ (Hint: maak gebruik van de grafische rekenmachine).
- (ii) Bepaal $\nabla f(x, y)$.
- (iii) Bepaal in welke richting de grafiek van f de grootste toename ervaart in het punt $(0, 0)$.
- (iv) Bepaal de extreme waarden (maxima en minima) van $f(x, y)$.
- (v) Bepaal $\nabla f(\frac{1}{4}\pi, -\frac{1}{3}\pi)$.
- (vi) Pas 2 stappen toe van het gradient descent-algoritme met stapgrootte $\frac{1}{8}$ en teken de punten. Het is handig om hiervoor de rekenmachine te gebruiken.



Figuur 7.2.1. Voorbeeld van een contourplot

7.3. Het feed forward-algoritme

Opgave 1: "Drie lagen diep": Stel dat een neuraal netwerk drie lagen heeft. De verdeling van neuronen in de lagen is $[2, 3, 2]$. Hoeveel gewichten heeft het netwerk? Kan je deze vraag ook beantwoorden als er $[a, b, c]$ neuronen in de lagen zitten?

Opgave 2 Gegeven de activeringsfunctie $k(x) = x^2$, bepaal voor het netwerk

$$\begin{bmatrix} I_1 \\ I_2 \end{bmatrix} \begin{bmatrix} -1 & 3 \\ -2 & 2 \end{bmatrix} \begin{bmatrix} o_1 \\ o_2 \end{bmatrix}$$

met biases

$$\begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

de uitvoer o_1, o_2 als

$$(i) \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$(ii) \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

Opgave 3: Bekijk nu het neurale netwerk met als verdeling van de lagen $[3, 2, 1, 3]$.

We hebben gewichten aan de eerste laag gegeven door $\begin{bmatrix} 1 & 3 & 2 \\ 1 & 0 & 2 \end{bmatrix}$. De gewichten

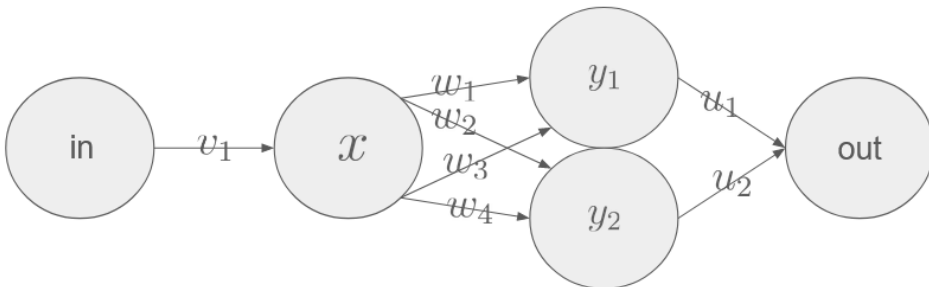
van de eerste laag aan de tweede laag gegeven door $\begin{bmatrix} 2 & 1 \end{bmatrix}$ en de gewichten van de tweede aan de laatste laag die gegeven zijn door $\begin{bmatrix} 2 \\ 1 \\ 10 \end{bmatrix}$. We maken nu gebruik van de

activeringsfunctie $k(x) = x(x - 2)$

- (i) Maak een tekening van dit neurale netwerk.
- (ii) Bepaal de output voor de volgende inputwaarden:

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}.$$

Opgave 4: We bekijken nu het neurale netwerk dat te vinden is in Figuur 7.3.1. In dit netwerk gebruiken we eerst de activeringsfunctie $k(x) = x$ voor alle neuronen in de verborgen laag en gebruiken we geen biases.



Figuur 7.3.1. Neuraal netwerk behorend bij opgave 2

- (i) Geef de verdeling van de neuronen in de lagen.
- (ii) Schrijf elke stap in het netwerk op als matrixvermenigvuldigingen.
- (iii) Bepaal voor een willekeurige input z nu de output door middel van het feed forward-algoritme.

7.4. Het backpropagation algoritme

Opgave 1

Een netwerk ziet er uit als

$$[I][w_1][O]$$

Met

$$b_O = 2, \quad w_1 = 0.5$$

De activeringsfunctie is x^2 . Bereken de fout die het netwerk maakt als $I = 4$ waar we verwachten dat $O = 2$. Doe dit ook met $\partial C / \partial b_O$.

Opgave 2 Bereken voor het netwerk $\partial C / \partial w_1$, als $I = 4$ waar we verwachten dat $O = 2$. Doe dit ook met $\partial C / \partial b_O$.

Opgave 3 Gebruik hetzelfde netwerk als bij opgave 1. Pas nu twee keer een klein stapje van grootte $\epsilon = 0.005$ toe in de $-\nabla C$ richting (door twee keer apart te bepalen hoe het gewicht en de bias aangepast moeten worden). Hoeveel dichterbij het juiste antwoord is het netwerk nu gekomen?

Opgave 4 Gebruik hetzelfde netwerk als bij opgave 1. Voor een andere invoer $I = 6$ verwachten we $O = -1$. Wat is de fout die het netwerk maakt? Is deze fout ook afgenomen door je trainingsschappen bij opgave 3?