Universiteit Hasselt

Faculteit Wetenschappen

OPTIMALISATIETECHNIEKEN VOOR NEURALE NETWERKEN

Bachelor Thesis

Sam Beunckens
Tuur Willio

Contents

Abstract

In deze paper doorgronden we theorie achter neurale netwerken en hoe deze tot stand komen. We werkten met de definitie van een *feed-forward* neuraal netwerk in de programmeertaal `Python` om de vorm van dit model beter te begrijpen. We breiden meer uit naar complexere algoritmen om een neuraal netwerk te trainen en vergelijken deze. We bespreken de volgende concepten. We zijn verder gegaan naar een theoretisch concept voor neurale vergelijkingen: de Universele Approximatie Stelling, om deze te bewijzen zoals werd voorgesteld in de literatuur. Om deze daarna te verbeteren met meerdere opties voor activatie functies en meerder lagen. Dit allemaal volgens een gelijkaardig principe, uitbreiding van lagen met Taylor reeksen. En meerdere activatie functies door middel van eerste graad veelterm interpolatie. We bekijken dan ook nog uitgebreid de hyperparameter optimalisatie voor neurale netwerken. We bespreken dien zijn methoden en algoritmes, hoe deze werken en de implementatie in de `Python` code.

1 Inleiding

In ons hedendaags leven valt het haast niet te ontwijken: Artificiële Intelligentie. Dezer dagen kent deze term vele variaties en toepassingen. Deze evolutie van artificiële intelligentie is sterk gefundeerd op *feed-forward* neurale netwerken. Terwijl deze neurale netwerken al enkele decennia meegaan zijn deze nog altijd sterk van belang in nieuwe vormen van artificiële intelligentie. Ze vormen wel degelijk een basis. In deze paper nemen we deze onder de loep en bekijken hoe goed deze juist zijn met hun benaderende eigenschappen. We bespreken hier ten eerste hoe neurale netwerk eigenlijk in elkaar zitten en kunnen convergeren naar een punt dat optimaal is. We zullen dan een neuraal netwerk zijn potentieel uitgebreid bespreken in de vorm van de Universele Approximatie Stelling. Een stelling die in literatuur veel vormen kent, met elks verschillende voorwaarden en gevolgen. Vele draaien rond abstracte ideeën in de analyse. We zullen zelf proberen een algemene, vatbare versie hier van te creëren dat niet te diep ingaat op abstracte theorie. Dit namelijk op basis van een visueel idee. Hierna zullen we verder gaan naar hyperparameters voor neurale netwerken, een term die de lezer nog zal tegenkomen. Een neuraal netwerk is namelijk een bundel van trainbare parameters en parameters om deze vorige te trainen. Net zoals deze trainbare aangepast kunnen worden naar keuze en dus naar een optimale oplossing, kunnen we ook deze hyperparameters, de parameters van de parameters, instellen zodat alle vorige processen efficiënter verlopen. Concreet bespreken we

- In Sectie 2: De vragen die we onszelf stellen over neurale netwerken en verwijzingen naar verwante literatuur en dien hun resultaten.
- In Sectie 3: Een hele uitwerking van wat neurale netwerken zijn, hoe deze getraind met methoden zoals gradiënt descent en backwards propagation en onze insteek op de uniforme approximatie stelling en hoe deze algemeen geformuleerd kan worden.
- In Sectie 4: Een bespreking van wat hyperparameters zijn en verschillende methoden en algoritmes om deze optimaal te bepalen.
- In Sectie 5: Een opsomming van gevonden resultaten.
- In Sectie 6: een korte conclusie van de bevindingen in dit werk.
- In Sectie 7: een appendix van onze gebruikte programmeercode.

2 Probleemstelling en literatuuroverzicht

2.1 Vraagstelling

We stellen onszelf de vraag wat een neuraal netwerk eigenlijk. Hoe deze eruit ziet en hoe deze informatie opneemt door middel van trainen. We bespreken deze delen kort, maar bondig om verder in te gaan op een stelling die laat zien dat een neuraal netwerk een continue functie kan benaderen, de Universele Approximatie Stelling. Wat zijn deze zijn vereisten en beperkingen? En kunnen we bevonden resultaten breder uittrekken tot een meer algemenere stelling? Uiteindelijk spreken we uitgebreid over de hyperparameteroptimalisatie. Een neuraal netwerk heeft naast trainbare parameters ook veel hyperparameters. Parameters die gesteld zijn voor dat het trainingsproces start. Voorbeelden van hyperparameters zijn de learning rate, mini-batch grootte (als dit van toepassing is) of de architectuur van het model. We stellen de vraag of deze invloed hebben en hoe we deze optimaal zouden kunnen kiezen om het trainingsproces zo snel mogelijk en zo effectief mogelijk te maken. Alle termen omtrent neurale netwerken en dien zijn verwanten die gebruikt zijn in deze paper, zullen aan bod komen in hun respectievelijke secties waar deze uitgelegd zullen worden.

2.2 Literatuuroverzicht

De cursussen [**?**][**?**][**?**] gaven de nodige achtergrond theorie over convexe optimalisatie problemen en gradiënt methoden zoals besproken in sectie 3.3.

De bachelorproef van Tom Janssens [**?**] biedt een solide eerste verkenning van de concepten die in dit werk worden behandeld, met name hyperparameter tuning en de bijbehorende methodologieën. Hoewel deze paper een goede introductie biedt tot de besproken concepten, ontbreekt een diepgaande analyse van de onderliggende ideeën en theoretische concepten. De beschrijving van de gebruikte methoden voor hyperparameter optimalisatie blijft rudimentair en biedt geen diepgaand inzicht in variaties op deze concepten. De toegepaste methoden zijn geïllustreerd aan de hand van een referentienetwerk, waarvan de resultaten zijn besproken.

Een belangrijk aspect bij werken over neurale netwerken is de Universele Approximatie Stelling. Deze stelling garandeert dat een willekeurige continue functie willekeurig goed benaderd kan worden op een compact interval. In de literatuur spreekt men hierbij meestal van een neuraal netwerk met een laag. Deze theorie en bewijs worden besproken in volgende papers.

De cursustekst in [**?**] geeft een kort, maar krachtig idee over dit theorema. Wiskundig gezien komt het er op neer dat de verzameling van neurale netwerken representaties dicht

is over de verzameling van continue functies over een compact domein K, $C(K)$. Deze stelling is bewezen onder de beperking van neurale netwerken met twee lagen. Ook geeft dit enkel resultaat voor *sigmoidal* functies als activatie functies. Dit zijn differentieerbare, niet dalende functies waarvoor geldt $\lim_{x\to\infty} f(x) = 1$ en $\lim_{x\to-\infty} f(x) = 0$. `ReLu`, de functie gedefinieerd als $max(0, x)$, ook een veelgebruikte activatie functie, valt hier bijvoorbeeld niet onder. De studie in [**?**], opnieuw deel van een cursus, geeft weer een andere manier van aanpak, ook met de beperking van *sigmoidal* functies, maar geeft de indruk dat dit uiteindelijk ook geldt voor bijvoorbeeld de $\texttt{ReLu}(z)$ functie omdat $\texttt{ReLu}(z) - \texttt{ReLu}(z-1)$ ook voldoet aan de eigenschappen om *sigmoidal* te zijn. Dit zonder verdere uitleg. In een meer uitgebreidere paper [**?**] spreekt men dit weer tegen voor de klassieke `ReLu` functie. Men laat wel zien dat er meer activatie functies mogelijk zijn over de ruimte $L^p(\mathbb{R}^m)$, waarin geldt dat functies Lebesgue integreerbaar zijn, dan over de continue functies $C(\mathbb{R}^m)$. Men stelt zelfs de mogelijkheid om activatie functies te construeren, zodat deze zouden voldoen aan de uniforme approximatie theorema. Dit, uiteraard, is een mooi resultaat. We zullen zelf ook een poging wagen om zo een stelling aan te tonen voor een algemene functie. Men kan opmerken dat in deze paper de `ReLu` functie niet behoort tot de verzameling van activatie functies die deze uniforme approximatie eigenschap bevatten. Deze verzameling omvat alle injectieve continue differentieerbare functies met afgeleiden bijna overal verschillend aan nul, geen vaste punten en niet polynomiaal. Wij zullen het theorema wel aantonen voor de `ReLu` functie. Vaak in literatuur over de universele approximatie theorema bewijst men deze voor een beperkt aantal activatiefuncties en aantal lagen in het neuraal netwerk, zoals in [**?**] of [**?**]. Deze resultaten zijn beperkend wetende dat de capaciteiten van een neuraal netwerk veel breder zijn.
Een meer visueel bewijs wordt besproken in sectie 3.5. Er kan een uitgebreidere versie gevonden worden in het boek [**?**]. Het legt stapsgewijs uit hoe dit bewijs in elkaar zit. Dit bewijs is wel minder analytisch en beschouwt enkel univariate functies, alhoewel het lichtjes meerdere dimensies beschouwt en hoe dit idee zou kunnen veralgemeend worden voor meerdere dimensies. Dit allemaal met *sigmoidal* functies als activatie functie. Dit bewijs zal ons op weg helpen naar een uitgebreidere versie die we bespreken in sectie 3.5.

Voor de sectie over hyperparametertuning zijn we vertrokken van uit [**?**]. Zoals de naam al zegt is dit een rigoureuze introductie tot het concept van Bayesiaanse Optimalisatie als methode om extreme waarden van een bestaande, moeilijk te evalueren, 'target of objective functie' te vinden. Deze paper geeft, na het opleggen van enkele voorwaarden op onze target functie, een introductie tot de algemene formule van het algoritme. Hierna gaan de auteurs dieper in op elk van deze onderdelen. Er wordt gesproken over wat het eigenlijk betekent om een distributie over functies te beschouwen en we introduceren het concept van een gaussian proces. Iets waar we later op terug zullen komen. Vervolgens wordt er gesproken over de verschillende acquisitie functies, een essentiële en belangrijke keuze die men moet maken bij het opstellen van dit algoritme. Er wordt gesproken over de voor- en nadelen bij elke

methode. Deze sectie wordt afgesloten met een korte discussie over het zogenaamde: "exploration versus exploitation" probleem. Wat men hier mee bedoelt is de afweging tussen een gebied met hoge functiewaardes, verder te benutten, tegenover nieuw gebied te ontdekken met een kans op nog grotere functiewaarden. Zoals vaak het geval is binnen het deelgebied van statistiek waar machine learning zich in bevindt, is hier geen eenduidig correct antwoord voor. De rest van dit hoofdstuk gaat over ruis op de input data,zowel i.i.d. als gecorreleerd. Dit zal hier niet besproken worden, maar desalniettemin is dit een interessant stuk dat zeker aan te raden is voor een geïnteresseerde lezer.

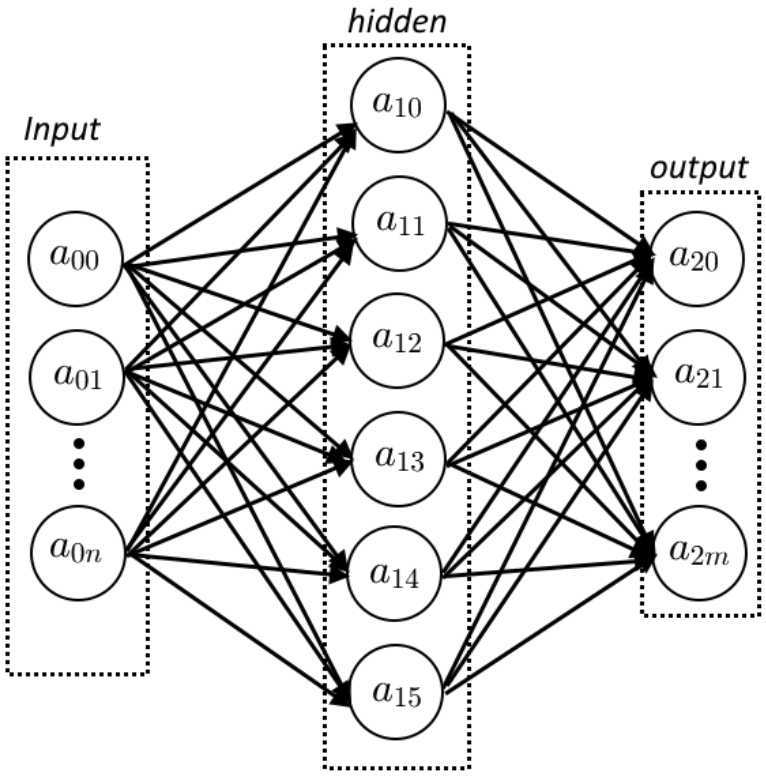
Om een beter beeld te krijgen op wat een gaussian proces is zijn we gaan kijken naar [**?**]. Deze paper brespreekt wat een gaussian proces en geeft verscheidene implementaties in context van Machine learning. Men vertrekt hier vanuit twee verschillende perspectieven om tot het model te komen. In het eerste deel vertrekt men vanaf lineaire regressie en in het tweede deel vanuit functieruimtes. Om binnen deze paper het concept uit te leggen zal de laatste hiervan gebruikt worden als leidraad om hier de essentiële informatie te verklaren.

Gaussian processes zijn niet de enige manier waarop men een model kan maken van de configuratieruimte. Tree-strucured Parzen Estimator (TPE) is de tweede methode die hier besproken zal worden. In [**?**] wordt een inleiding tot de methode gegeven. Bij TPE zal men gebruik maken van niet-parametrische methodes om onze modellen op te stellen. Specifieker met behulp van Kernel density estimation, of Parzen window estimation. Dit is de methoden, besproken in [**?**] die ons toelaten de dichtheidsfunctie te schatten met enkel datapunten gekend, zonder aanname van een onderliggende distributie.

3 De Opbouw van een Neuraal Netwerk

3.1 Architectuur van een Neuraal Netwerk

Een neuraal netwerk bestaat een drie delen: een input-layer, een hidden-layer en een output-layer. De hidden-layer kan meerdere layers bevatten. Al deze layers bestaan uit nodes. De input en output bestaan dus uit een aantal nodes afhankelijk van hoeveel input men wil geven en hoeveel outputs men wil ontvangen. Het aantal nodes in de hidden layers is een keuze die gemaakt moet worden. Deze keuze gaat natuurlijk veel invloed hebben op de komende processen, meer hierover in de volgende hoofdstukken.



Figuur 1: Feed-forward neuraal netwerk met n inputs, m outputs en een hidden layer met vijf nodes

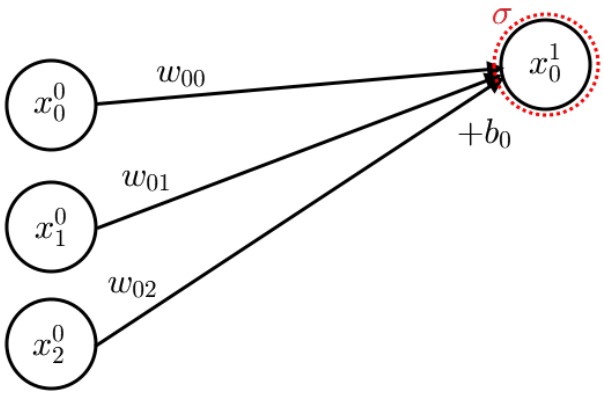
Een neuraal netwerk is een functie afhankelijk van verschillende parameters $f : \mathbb{R}^n \to \mathbb{R}^m$, van n inputs naar m outputs. Meestal zijn dit floats in [0,1], of $f : [0,1]^n \to [0,1]^m$.
Hoe deze functie nu echt berekend wordt is als volgt: Alle nodes van twee aangrenzende layers worden met elkaar verbonden, zoals zichtbaar op de afbeelding, waarbij elke pijl een

gewicht/weight voorstelt w_{ij} van node j in de vorige layer naar node i in de volgende layer. Voor elke node i in een volgende layer hoort er ook een bias b_i. Met deze parameters doet men een simpele lineaire transformatie van vorige nodes naar een volgende node, dit is de term binnen de σ functie in (1). In de huidige opstelling zal dit netwerk, ongeacht de complexiteit, reduceerbaar zijn tot een vorm van lineaire regressie. Om dit te voorkomen gebruikt men een activatiefunctie. Dit introduceert niet-lineariteit in het model en laat toe, complexere fenomenen beter te beschrijven. Deze activatie functie helpt ook om waarden te herschalen. Deze wordt ook meestal aangeduid met σ en is vaak dan ook de sigmoid functie $\sigma(x) = \frac{1}{1+e^{-\epsilon x}}$. Hierbij stelt de epsilon de breedte van deze functie voor. We zullen later nog bespreken. Uiteraard zijn er ook andere functies mogelijk. Dit geeft weer een parameter die aangepast kan worden.

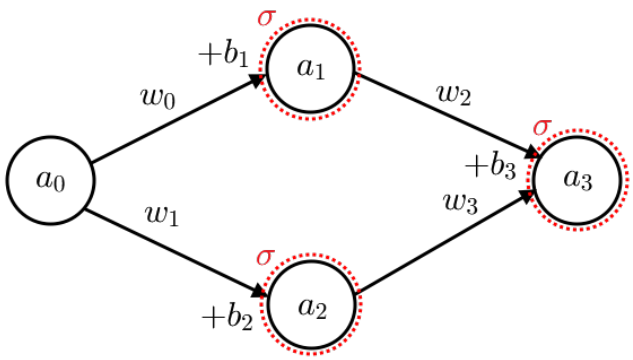
$$x_i^1 = \sigma(\sum_{j=0}^{n}(w_{ij} \cdot x_j^0) + b_i) \tag{1}$$

Waarbij dit de node met index i voorstelt in de eerste hidden layer. In algoritmes worden deze berekeningen dan ook meestal gedaan met matrices. Dit is overzichtelijker. Uiteraard betekent dat in dit geval, de activatie functie toegepast op een vector, de gewone definitie elementgewijs toegepast.

$$X^1 = \begin{bmatrix} x_0^1 \\ \vdots \\ x_m^1 \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{00} & \cdots & w_{0n} \\ \vdots & \ddots & \vdots \\ w_{m0} & \cdots & w_{mn} \end{bmatrix} \begin{bmatrix} x_0^0 \\ \vdots \\ x_n^0 \end{bmatrix} + \begin{bmatrix} b_0 \\ \vdots \\ b_m \end{bmatrix}\right)$$



Figuur 2: De berekening van een volgende node in een neuraal netwerk. Hier zou de waarde van de node $x_0^1 = \sigma(w_{00}x_0^0 + w_{01}x_1^0 + w_{02}x_2^0 + b_0)$ zijn



Figuur 3: Een 'eenvoudig' neuraal netwerk met twee hidden nodes

Deze berekeningen voor een waarde van een node wordt herhaald voor elke layer zodat de uiteindelijke uitkomst een soort recursie is van gewogen sommen in een activatie functie. In literatuur wordt naar deze bewerkingen verwezen als *forward propagation*

3.2 Loss functie

Om een neuraal netwerk te trainen is het eerst zeer belangrijk om te bepalen hoe goed het netwerk presteert aan de hand van de trainingsdata. Hier wordt het begrip Loss functie of ook wel kost functie geïntroduceerd. Een loss functie, C afhankelijk van de output van het netwerk met betrekking tot de gewenste output uit de trainingsdata, is een functie die numeriek gaat bepalen hoe ons netwerk presteert. Een van de meest voor de hand liggende functies is de Mean Squared Error functie:

$$C(Y,\widetilde{Y}) = \sum_{i=1}^{n} \left(Y_i - \widetilde{Y}_i\right)^2 \tag{2}$$

met:

- Y is de vector van de gegeven correcte oplossingen.
- $\widetilde{Y}$ is de vector gegenereerd door het netwerk met $\widetilde{Y}_i$ elementen hier met respectievelijke index i.

Deze vectoren hebben beide lengte n. De MSE kan wordt ook vaak gebruikt bij classificatie problemen. Hier zou elke node in de ouput een andere klasse voorstellen. Deze hebben elks als output van het neuraal netwerk een numerieke waarden wat niet classificatie is. De MSE zou de output kunnen bepalen door het argument te pakken van het maximum van de output. Er zijn nog tal van andere keuzes zoals cross entropy loss en negatieve log-likelihood, echter valt dit buiten de scope van deze paper. Het netwerk "Trainen" is dus eigenlijk equivalent met het minimaliseren van deze Loss functie.

3.3 Gradiënt Methoden

Nu we weten dat een neuraal netwerk een functie is afhankelijk van een reeks parameters (weights, biases) en dat deze een continue functie kan benaderen, kunnen we op zoek gaan naar deze parameters voor een trainingsset van data bestaande uit meerdere combinaties van inputvectoren X en outputvectoren Y zodat $Y \approx f(X)$. Dit kan men doen met een gradiënt methode.
Veronderstel dat de multivariate, continue functie f overal convex is en dus een globaal minimum heeft. De algemene formule van een gradiënt methode heeft volgende vorm:

$$x^k = x^{k-1} - \eta \nabla f(x^{k-1}) \tag{3}$$

waarbij η wordt verwezen als de *learning rate* die de stapgrootte nog kan aanpassen. Voor een startpunt x^0 zal deze dan convergeren naar het minimum, omdat voor een punt a daalt $f(x)$ het snelst in de richting van $-\nabla f(a)$ en er geldt dus $f(x^k) \leq f(x^{k-1} + \eta \nabla f(x^{k-1}))$. Hieruit volgt dan:

$$x^k \longrightarrow argmin_{x \in X} f(x)$$

Men kan deze methode toepassen voor alle weights en biases in het neuraal netwerk met als functie de kost- of *loss*-functie. Men zoekt dan parameters om de MSE te minimaliseren. Dit proces noemt ook wel het trainen van het netwerk. Alleen kan men laten zien dat de loss functie voor een neuraal netwerk geen convexe functie is. Uit figuur 3 kan men de parameters $(w_0, b_1, w_1, b_2, w_2, w_3, b_3)$ nemen, zodat deze dus hier een minimum heeft. Men kan zien dat een simpele permutatie van de parameters, $(w_1, b_2, w_0, b_1, w_3, w_2, b_3)$, eigenlijk hetzelfde resultaat levert, omdat de optelling immers commutatief is in de lineaire transformatie van de nodes, terwijl dit een compleet ander punt geeft in onze oplossingsruimte. Hieruit volgt dus dat het oorspronkelijk minimum geen globaal minimum is maar een lokaal minimum en dus per definitie de loss functie niet globaal convex is. Merk op dat dit niet afhankelijk is van een specifieke loss functie. Men kan dit idee uitbreiden naar alle neurale netwerken. Dit geeft geen probleem voor de gradiënt methoden omdat de Hessiaan in een kritisch punt van het neuraal netwerk in de plaats van positief definitief (voor een convexe functie) ook onbepaald kan zijn, dat een zadelpunt levert op het gegeven punt. Uit het onderzoek in [**?**] halen we dat een gradiënt methode praktisch altijd een strikt zadelpunt zal vermijden, omdat de bereikte zadelpunten Lebesque maat nul hebben. Dit wel mits de Hessiaan begrensd is, wat volgt uit het feit de dat de gradiënt ∇f Lipschitz is met L_2 norm. Wat op zijn beurt weer volgt voor functies die C^2 zijn.

$$||\nabla f(x) - \nabla f(y)||_{L_2} \leq L||x - y||_{L_2} \Longleftrightarrow ||\nabla^2 f||_{L_2} \leq L \qquad f \in C^2$$

Dit laatste volgt uit de middelwaardestelling. Dat de gradiënt Lipschitz continu is, is een aanname en geldt bijna altijd [**?**]. Ook zijn lokale minima meestal zeer dicht bij het werkelijk globaal minimum en door de permutatie van parameters zijn vele ook gelijk aan elkaar.

Er zijn verschillende variaties op deze standaard gradiënt methoden. Een eerste variatie is de mini-batch variatie waarbij de trainingsdata wordt opgesplitst in groepen van een bepaalde grootte. In de plaats van bij een iteratie $x^k = x^{k-1} + \eta \nabla f(x^{k-1})$ de gradiënt van de functie f (*loss* functie) te beschouwen over alle trainingssamples (x_i, y_i) beschouwt men enkel de data die samen in een batch zitten. Dit blijkt computationeel efficiënter te zijn, omdat een volledige trainingsset vaak gigantisch groot is. Door het opsplitsen gebruikt men veel minder rekentijd tijdens een iteratie terwijl uiteindelijk alle trainingssamples wel aan bod komen. Dit laatste doordat bij elke iteratie een andere batch wordt gebruikt. Dit leidt wel naar een iets meer afwijkend pad naar beneden op de *loss* functie (van alle trainingsdata) dan de standaard methode, omdat deze immers minder info bevat op een iteratie. Een nog radicalere methode is de Stochastische Gradiënt methode, deze doet inhoudelijk hetzelfde, maar berekent nu de gradiënt met maar een enkel willekeurig trainingssamples. Dit leidt opnieuw naar snellere berekeningen.
De keuze om niet alle trainingssamples te gebruiken kan ook leiden tot de nodige afwijking om weg te sturen van lokale minima die slecht presterende parameters zou opleveren. Er zijn ook veel lokale minima die slechts dipjes vormen in de oppervlakte van de loss functie. Verder zijn er nog meer variaties op deze methoden, zoals het toevoegen van momentum, bijhouden van gemiddelde en adaptieve learning rates. ADAM is bijvoorbeeld een combinatie van deze aspecten. ADAM vergt dan wel opnieuw meer rekentijd, maar kan sneller convergeren [**?**]. Wij zullen in deze paper verder werken met de minibatch methode.

Een probleem dat nog niet is aangekaart is de berekening van de gradiënt in de praktijk. We bespreken twee manieren. Men kan de gradiënt berekenen door middel van de eindige differentie methode. In de plaats van de limiet $f'(x) = lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$ te nemen, nemen we dan voor h een waarde klein genoeg en berekenen we voor een parameter x de waarde van de loss functie bij toevoeging van een kleine waarde h, $f(x+h)$, en zonder h, $f(x)$. Deze methode blijkt zeer accuraat en kan snel gebeuren afhankelijk van gekozen grootte voor de minibatch om de gradiënt te berekenen.
Een andere optie om de gradiënt te berekenen halen we uit de beginselen *Backwards propagation.*

3.4 Backwards Propagation

Het effectief numeriek berekenen van de gradient met behulp van voorwaartse differentie komt met enkele nadelen. De implementatie is zeer eenvoudig, maar kan snel computationeel duur worden. Bij een naïeve implementatie zou men, voor elk component van de gradiënt, een klein increment h optellen bij de weight of bias en vervolgens de cost opnieuw berekenen. Voor grotere netwerken is dit natuurlijk niet meer haalbaar om te berekenen. Een andere oplossing is om de gradiënt expliciet te berekenen met behulp van backpropagation.

Bij backpropagation zullen we gebruik maken van de kettingregel om deze expliciet te bepalen. Eerst beschouwt men hier een eenvoudig netwerk met een input node, een output node en een hidden node. In dit geval beschouwt men ook enkel een trainingsvoorbeeld. De uitbreiding hiervoor is echter triviaal: men moet de onderstaande berekening herhalen voor elk trainingsvoorbeeld om tot het resultaat te komen.
Beschouw de volgende matrices waaruit het netwerk bestaat:

- $A^{[L]} = [a_j^{[L]}]$, de waarde van de node j in laag L.
- $W^{[L]} = [w_{ji}^{[L]}]$, de waarde van het gewicht naar j, van i in laag L.
- $B^{[L]} = [b_j^{[L]}]$, de waarde van de bias van node j in laag L.
- $Z^{[L]} = W^{[L]}A^{[L-1]} + B^{[L]}$ zodat $\sigma(Z^{[L-1]}) = A^{[L]}$

Men wil de gradient $-\nabla C$ berekenen om tot het minimum te geraken met behulp van de gradientmethoden. Hierbij is C een loss functie. Beschouw daarom de expliciete uitdrukking van de afgeleide$\frac{\partial C}{\partial w^{[L]}}$(, het proces verloopt compleet analoog als men naar de bias wil kijken).

$$\frac{\partial C}{\partial w^{[L]}} = \frac{\partial C}{\partial a^{[L]}} \frac{\partial a^{[L]}}{\partial z^{[L]}} \frac{\partial z^{[L]}}{\partial w^{[L]}} \tag{4}$$

Deze uitdrukkingen zijn allemaal gemakkelijk te bepalen:

1. $\frac{\partial C}{\partial a^{[L]}}$: Als men terugkijkt naar 2 is duidelijk dat deze uitdrukking neerkomt op $2 \cdot (Y - a^{[L]})$
2. $\frac{\partial a^{[L]}}{\partial z^{[l]}}$: $a^{[L]} = \sigma(a^{[L-1]}w^{[L]} + b^{[L]})$, dus dit komt neer op $\sigma'(a^{[L]}w^{[L]} + b^{[L]})$.
3. $\frac{\partial z^{[L]}}{\partial w^{[L]}}$: deze uitdrukking is gelijk aan $a^{[L]}$

Dit zijn allemaal waardes die reeds ergens opgeslagen zijn, of op enkele bewerkingen na. Om nu $\frac{\partial C}{\partial b^{[L]}}$ te berekenen overlopen we volledig dezelfde stappen, enkel is de laatste partieel gelijk aan$\frac{\partial (z^{[L]})}{\partial b^{[L]}} = 1$.

Als men nu de gradiënt wil berekenen voor een weight in de laag hiervoor gebruikt men opnieuw de kettingregel:

$$\frac{\partial C}{\partial w^{[L-1]}} = \frac{\partial C}{\partial a^{[L]}} \frac{\partial a^{[L]}}{\partial z^{[L]}} \frac{\partial z^{[L]}}{\partial a^{[L-1]}} \frac{\partial a^{[L-1]}}{\partial z^{[L-1]}} \frac{\partial z^{[L-1]}}{\partial w^{[L-1]}} \tag{5}$$

Zoals men hier ziet, is dit nog altijd redelijk eenvoudig te berekenen. Merk op dat als men de gewichten heeft geüpdatet in de laatste laag, men door kan gaan naar de laag hiervoor

door $\frac{\partial C}{\partial a^{[L]}}$ door te geven en deze dan voor de gewichten in de laag hiervoor te gebruiken.

Vervolgens wordt het algemene geval beschouwd. Een neuraal netwerk met L lagen, elk met grootte n_L. Voor een gewicht $w_{ji}^{[L]}$ aan het rechteruiteinde van het netwerk past men analoog aan hierboven de kettingregel toe:

$$\frac{\partial C}{\partial w_{ji}^{[L]}} = \frac{\partial C}{\partial a_j^{[L]}} \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \frac{\partial z_j^{[L]}}{\partial w_{ji}^{[L]}} \tag{6}$$

Voor een bias $b_j^{[L]}$ is de bovenstaande vergelijking compleet analoog, alleen zal de laatste partiële afgeleide $\frac{\partial z_j^{[L]}}{\partial b_j^{[j]}}$, uiteindelijk één worden. Als men een laag naar links wil gaan in het backpropagation algoritme zal, zoals hierboven, men moeten opletten. Een node in een laag $L-1$ beïnvloedt elke node op laag L. Dit wil zeggen dat:

$$\frac{\partial C}{\partial a_i^{[L-1]}} = \sum_{j=0}^{n_L} \frac{\partial C}{\partial a_j^{[L]}} \frac{\partial a^{[L]_j}}{\partial z_j^{[L]}} \frac{\partial z_j^{[L]}}{\partial a_j^{[L]}} \tag{7}$$

Als deze waarde bekend is kunnen we wederom met de kettingregel het gradiëntcomponent van gewichten en biases hiervoor berekenen: $\frac{\partial C}{\partial a_i^{[L-1]}} \frac{\partial a_i^{[L-1]}}{\partial z_i^{[}L-1]} \frac{\partial z_i^{[}L-1]}{\partial w_{ki}}$.
Dit hint al op een recursieve manier om backpropagation te implementeren:

1. bepaal de gradiëntcomponenten van de gewichten en biases in de laatste laag
2. bereken $\frac{\partial C}{\partial a^{[}L-1]}$
3. gebruik deze partiële afgeleide om de gradientcomponenten van de gewichten en biases van de laag hiervoor te berekenen
4. herhaal tot men terugkomt bij de eerste laag.

Echter zal in de broncode de recursieve versie niet worden gebruikt, recursie komt vaak gepaard met overhead.

3.5 Universal Approximatie Stelling

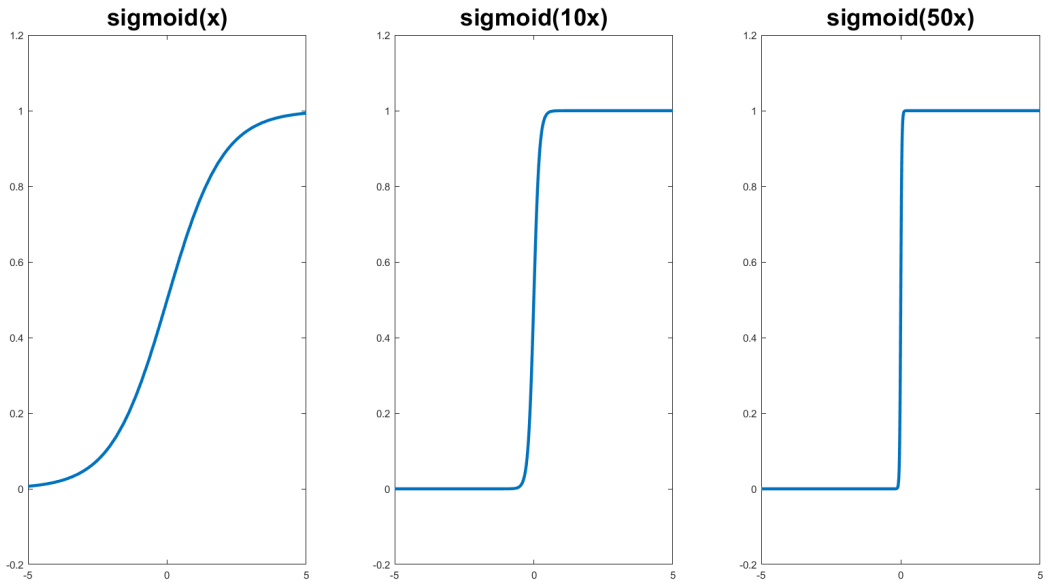
In literatuur wordt er naar dit theorema verwezen als het feit dat verzameling van functies beschreven door neurale netwerken dicht is in de verzameling van continue functies op een compacte set. De implicatie die het geeft, is wat er zo belangrijk is voor neurale netwerken: alle continue functies kunnen willekeurig dicht benaderd worden door een neuraal netwerk op een compacte verzameling.

3.5.1 Sigmoidale functies

In deze sectie laten we hier een visueel bewijs van zien op basis van [**?**]. Deze was wel met de beperking van sigmoidale functies waarvoor geldt:

$$\lim_{x\to\infty} f(x) = 1 \qquad \lim_{x\to-\infty} f(x) = 0 \tag{8}$$

Deze eigenschap is zeer belangrijk omdat men hiermee de functie kan vormen tot een trap functie (zie figuur 4), die nodig zal zijn in het bewijs. We zullen later aantonen dat dit idee uitgebreid kan worden tot functies verschillend van de sigmoidale functies.
Het idee achter het bewijs is om voor elke functie een benaderende trapfunctie te construeren, net zoals de Riemann sommen een integraal benaderen.
Met de transformatie uit figuur 4 hebben we al een trap kunnen vormen. Om een tweede trap te vormen moeten we eerst een afstand creëren. Als referentie punt pakken we, voor de sigmoid functie $\sigma(x) = \frac{1}{1+e^{-x}}$, even het punt waar $\sigma(ax) = 1/2$. Dit komt neer op het punt waar geldt $e^{-ax} = 1$ en dus $x = 0$. Voor $\sigma(ax + b) = 1/2$ krijgen we uiteindelijk $x = -\frac{b}{a}$. We kunnen a en b dus zo veranderen, zodat we een ruimte van grootte $\frac{b}{a}$ creëren tussen functies $\sigma(ax)$ en $\sigma(ax + b)$. Of algemener; een afstand van $\frac{|b_1-b_0|}{|a|}$ voor $\sigma(ax + b_0)$ en $\sigma(ax + b_1)$. Merk op dat dit een lineaire transformatie is binnen de sigmoid functie, net zoals de vector berekeningen voor een neuraal netwerk binnen de activatiefunctie. En dus de functies $s_{0,1}(x) = \sigma(ax + b_{0,1})$ kunnen we beschouwen als de twee eerste hidden nodes, na input, met dezelfde weight a en verschillende biases b_0, b_1. En deze a speelt enkel een rol in het meer hoekig maken van de functies.

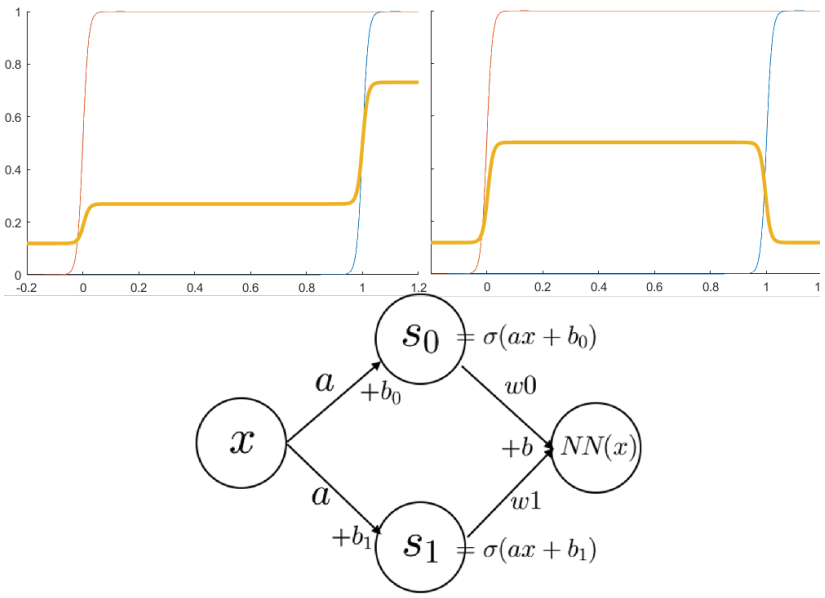


Figuur 4: De sigmoid functie met verschillende input

Men kan dan s_0 en s_1 vermenigvuldigen met een gewenste traphoogte (dit kan door sigmoidale activatiefunctie, ze hebben immers hoogte gaande naar 1), w_0 en w_1 respectievelijk. Optelling met elkaar levert dan weer een trapfunctie met een extra trap. Dit weer gestuurd door de sigmoid functie als activatie functie, zou de output zijn van een neuraal netwerk met een hidden layer die twee nodes (s_0 en s_1) bevat, zoals in figuur 5. Toevoeging van meerdere nodes resulteert dus in meerdere trappen die aan te passen zijn volgens hun parameters in het neuraal netwerk. Op deze manier kan men een functie gaan benaderen.
Merk op dat het beeld van een neuraal netwerk op deze manier altijd in $[0, 1]$ ligt. Dit vormt echter geen probleem voor functies die een beeld hebben buiten dit interval, als we werken met een compact domein. Een compact domein resulteert dan ook in een compact beeld voor een continue functie, wat dan begrensdheid impliceert voor het beeld. Met een begrensd beeld kunnen we makkelijk een functie herschalen, zodat deze het compacte domein afbeeldt in $[0, 1]$. Elke continue functie is dus een triviale factor verwijderd van een benaderend neuraal netwerk met een laag.

We zullen nu laten zien dat we effectief een rechthoek op deze manier kunnen benaderen, gegeven een rechthoek, met $\alpha, \beta \in [0, 1]$ met $\alpha > \beta$:

$$h(x) = \begin{cases} \alpha & a \leq x \leq b \\ \beta & elders \end{cases} \tag{9}$$



Figuur 5: Output van een neuraal netwerk met parameters: $a = 100$, $b_0 = 0$, $b_1 = 100$, $w_0 = \{1, 2\}$, $w_1 = \{2, -2\}$, $b = -2$

Beschouw ook de verzamelingen van alle neurale netwerken met nul, één of n hidden layers.

$$NN_0 = \{\sigma(wx + b) \mid w, b \in \mathbb{R}\}$$

$$NN_1 = \{\sigma(\sum_{i=0}^{k}(w_i s_i(x)) + b) \mid s_i \in NN_0;\ w_i, b \in \mathbb{R};\ k \in \mathbb{N}\}$$

$$NN_n = \{\sigma(\sum_{i=0}^{k}(w_i s_i(x)) + b) \mid s_i \in NN_{n-1};\ w_i, b \in \mathbb{R};\ k \in \mathbb{N}\}$$

$$NN = \cup_{k=0}^{\infty} NN_k \tag{10}$$

Hieruit volgt ook duidelijk dat de lineaire combinatie van neurale netwerk plus een extra factor door de activatie opnieuw een neuraal netwerk is. Dit zal later handig zijn.
Voor een gegeven rechthoek h, stellen we een neuraal netwerk op uit NN_1 met twee nodes van waarden $s_0(x)$ en $s_1(x)$:

$$s_0(x) = \sigma(wx + b_0) = \sigma(w(x - a)) \tag{11}$$

$$s_1(x) = \sigma(wx + b_1) = \sigma(w(x - b))$$

Hierbij nemen we $b_0 = -wa$ en respectievelijk $b_1 = -wb$ door de eerder vermelde redenen. Merk op dat hieruit volgt:

$$s_1(x) = s_0(x - (b - a))$$

De factor w speelt dus een rol in het 'hoekig' maken van onze benadering. Dit zien we duidelijk in de afgeleiden van $\sigma(wx + b)$.

$$\frac{d}{dx}\sigma(wx + b) = w \cdot \sigma'(wx + b) \tag{12}$$

De sigmoidale functie is begrensd en dus ziet men duidelijk dat een grotere waarde voor w zorgt voor een extremere afgeleiden en dus een hoekig karakter. Beschouw nu de voorlopige functie f:

$$f_w(x) = \sigma(w_1 s_0(x) - w_1 s_1(x) + b) \tag{13}$$

We maken deze afhankelijk van een w zoals gedefinieerd bij (11), omdat deze factor uiteindelijk onze benadering zal verbeteren als deze naar $+\infty$ gaat. We stellen het tweede gewicht hier gelijk aan $-w_1$ zodat de begin- en eindhoogte hetzelfde zal blijven zoals β gedefinieerd voor h. Om te bewijzen dat dit een rechthoek wordt, gaan we bewijzen dat deze een afgeleiden zal hebben die willekeurig dicht zal liggen bij het volgende schema:

$f'_{+\infty}(x)$	0	$+\infty$	0	$-\infty$	0
x		a		b	

(14)

Voor we hieraan beginnen, nemen we aan voor de activatie functie dat deze *sigmoidal* is zoals beschreven bij (8). Verder nemen we ook aan dat:

- σ is continu op $\mathbb{R}$
- $\sigma'(x) \geq 0 \quad \forall x \in \mathbb{R}$
- $\lim_{x\to\infty} x\sigma'(x) = 0$

Uit de limieten van een sigmoidale functie volgt dus dat $\lim_{x\to\pm\infty} \sigma'(x) = 0$. Merk op dat dit allemaal geldt voor bijvoorbeeld de sigmoid functie $\sigma(x) = 1/(1 + e^{-x})$
Voor duidelijkheid definiëren we volgende functie g:

$$\begin{aligned} g(x) &= w_1 s_0(x) - w_1 s_1(x) + b \\ g'(x) &= w_1 \cdot w(s'_0(x) - s'_1(x)) = w_1 \cdot w(\sigma'(w(x - a)) - \sigma'(w(x - b))) \end{aligned}$$

Zodat hier uit volgt: $f'(x) = \sigma'\big(g(x)\big)g'(x)$. We merken op dat $g'(x)$ de volgende vorm heeft:

$$w \cdot \sigma'(wx) \tag{15}$$

Er geldt dan, voor $\sigma'(0) = c > 0$:

$$\lim_{w\to+\infty} w \cdot \sigma'(w0) = \lim_{w\to+\infty} w \cdot c = +\infty$$

Als $x \neq 0$ dan krijgt men het volgende:

$$\lim_{w\to+\infty} w \cdot \sigma'(wx) = \lim_{u\to+\infty} u \cdot \sigma'(u)/x = 0/x = 0$$

Als we $x = a$ of $x = b$ nemen zien we dus dat $g'(x)$, voor een w naar oneindig, respectievelijk naar plus en min oneindig gaat in orde van w zelf. Dit omdat $\sigma'(w \cdot \pm(a - b))$ sowieso naar nul gaat voor w naar oneindig in $g'(x)$ en hebben we dus $g'(x) = w_1 \cdot \pm w\sigma'(0)$. Dus kunnen we gewoon $w \cdot \sigma'(0)$ toepassen zoals hierboven beschreven. Voor $x \neq a$, b krijgen we dan voor $g'(x)$ dat dit gewoon nul min nul is door het hierboven beschreven limiet. Omdat $g(a)$ en $g(b)$, voor w naar oneindig, verschillend van nul zijn en niet zal divergeren zal $f'(x)$ naar respectievelijk plus en min oneindig gaan. Voor $x \neq a$ of $x \neq b$ geldt dit ook en zal $f'(x)$ naar nul gaan.
Verder beschouwen we de functiewaarden met het idee we onze parameters te kunnen aanpassen, zodat het beeld naar α en β gaat zoals gedefinieerd in het voorschrift van h.

1. $x < a < b$
   in dit geval krijgen we voor de functie $f_w(x)$, voor alle x,

$$\begin{aligned}\lim_{w\to+\infty} f_w(x) &= \lim_{w\to+\infty} \sigma(w_1 \cdot \sigma(w(x-a)) - w_1\sigma(w(x-b)) + b) \\ &= \sigma(w_1 \cdot 0 - w_1 \cdot 0 + b) = \sigma(b)\end{aligned}$$

2. $a < x < b$
   in dit geval krijgen we voor de functie $f_w(x)$, voor alle x,

$$\begin{aligned}\lim_{w\to+\infty} f_w(x) &= \lim_{w\to+\infty} \sigma(w_1 \cdot \sigma(w(x-a)) - w_1\sigma(w(x-b)) + b) \\ &= \sigma(w_1 \cdot 1 - w_1 \cdot 0 + b) = \sigma(w_1 + b)\end{aligned}$$

3. $a < b < x$
   in dit geval krijgen we voor de functie $f_w(x)$, voor alle x,

$$\begin{aligned}\lim_{w\to+\infty} f_w(x) &= \lim_{w\to+\infty} \sigma(w_1 \cdot \sigma(w(x-a)) - w_1\sigma(w(x-b)) + b) \\ &= \sigma(w_1 \cdot 1 - w_1 \cdot 1 + b) = \sigma(b)\end{aligned}$$

Men ziet dat dit resulteert in een rechthoek met hoogtes $\sigma(b)$ en $\sigma(w_1+b)$. Voor de rechthoek $h(x)$ gedefinieerd bij (9), is dit dan een kwestie van de vergelijking op te lossen.

$$\begin{aligned}\alpha &= \sigma(w_1 + b) \\ \beta &= \sigma(b)\end{aligned} \tag{16}$$

We krijgen dus:

$$b = \sigma^{-1}(\beta)$$
$$w_1 = \sigma^{-1}(\alpha) - \sigma^{-1}(\beta)$$

Moest een of twee van de waarden gelijk zijn aan nul of één kan men de parameters ook naar plus of min oneindig laten gaan zoals w dat doet in de functie $f_w(x)$, door bijvoorbeeld $b = -w$, $w1 = 2w$ voor een rechthoek met hoogtes nul en één. Deze laatste zodat w_1 groter blijft dan b en $w_1 + b$ wel degelijk naar $+\infty$ gaat. Uit (16) kan men ook duidelijk zien dat moesten we een sigmoidale functie hebben waarvoor geldt:

$$\lim_{x\to\infty} \sigma(x) = c \tag{17}$$

dat we dan $\alpha = \sigma(cw_1 + b)$ krijgen en dit dus ook oplosbaar is voor w_1 en b te bepalen. De redenering is volledig identiek.
Gegeven dus een stap functie $h(x)$ kan men hier dus een willekeurig dichte benadering van geven.

$$\forall x,\, \forall \epsilon > 0,\, \exists w(x, \epsilon) > 0 :\, |f_w(x) - h(x)| \leq \epsilon \tag{18}$$

Dit concept kan uitgebreid worden tot meerdere hoogtes door toevoeging van meer nodes in de hidden layer. De uitwerking is analoog aan deze uitwerking voor meerdere trappen en dus meerdere nodes.
Voor de benadering van een continue functie, splitsen we het interval $[0, 1]$ op in n even grote delen $[x_i, x_{i+1}]$ voor $0 \leq i \in \mathbb{N} \leq n$. Op deze intervallen definiëren we een h zodat deze constant is op deze delen. We zullen de fout meten door de L^2 norm te nemen voor functies. We krijgen dus bijvoorbeeld

$$h(s) = max_{x\in[x_i,x_{i+1}]} f(x) \qquad s \in [x_i, x_{i+1}] \tag{19}$$

Dit is goed gedefinieerd omdat de intervallen opnieuw compact zijn. Deze definitie voor h is niet de enige optie, maar men verwacht wel dat de waarden van h goed genoeg liggen, zodat deze hun fout op een punt in het interval kleiner of gelijk is aan $maxf(x) - minf(x)$. De totale fout van de benadering van h_w, afhankelijk van de w parameter uit (18), op een

Lipschitz continue functie f kan dan afgeschat worden als volgt:

$$\begin{aligned}
||f - h_w||_{L^2} = ||f - h + h - h_w||_{L^2} &\leq \left(\sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} (f(x) - h(x))^2 dx \right)^{1/2} + \epsilon_w \\
&\leq \left(\sum_{i=1}^{n} |max_{x\in[x_i,x_{i+1}]} f(x) - min_{x\in[x_i,x_{i+1}]} f(x)|^2 \int_{x_i}^{x_{i+1}} dx \right)^{1/2} + \epsilon_w \\
&\leq \left(\sum_{i=1}^{n} L^2 |x - y|^2 \cdot (x_{i+1} - x_i) \right)^{1/2} + \epsilon_w \\
&\leq \left(\sum_{i=1}^{n} L^2 (x_{i+1} - x_i)^3 \right)^{1/2} = \left(n \cdot \frac{L^2}{n^3} \right)^{1/2} + \epsilon_w \\
&= \frac{L}{n} + \epsilon_w
\end{aligned} \tag{20}$$

Waarbij L uiteraard de maximale Lipschitz constante is, de constante die geldt over het hele interval waar we f op benaderen, en $x = argmax_{x\in[x_i,x_{i+1}]} f(x)$ en $y = argmin_{x\in[x_i,x_{i+1}]} f(x)$. De waarde ϵ_w is dus afhankelijk van de parameter w die naar oneindig gaat. Deze hangt af van hoe snel de σ functie naar één gaat in zijn limiet naar oneindig. Deze is dus functie specifiek en laten we open voor het algemene geval.
Men ziet duidelijk dat de definitie van h niet per se belangrijk is, maar wel goed genoeg moet zijn. De waarden voor n, het aantal intervallen te verdelen over $[0, 1]$, is door het karakter van het neuraal netwerk ook het aantal nodes in die zijn enige hidden layer. Hieruit volgt dan duidelijk:

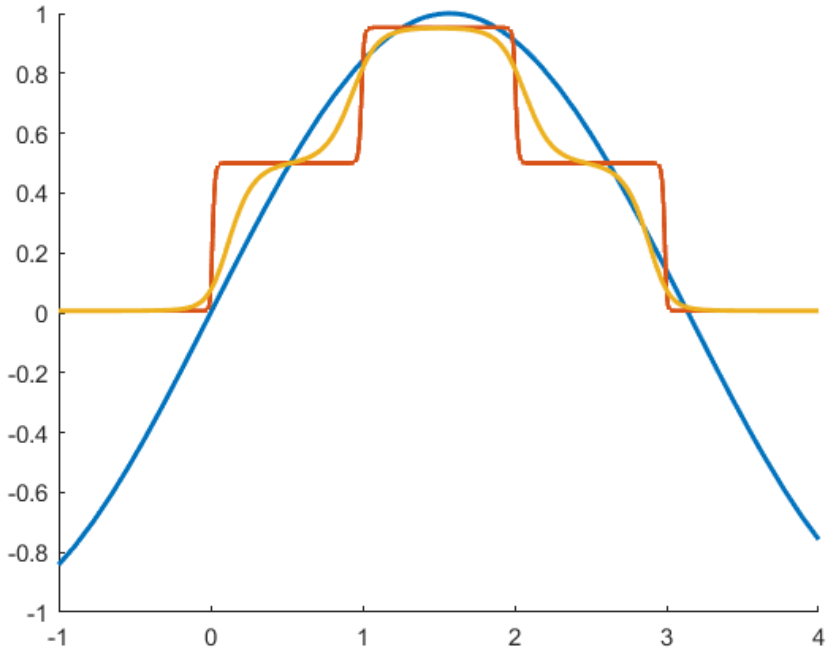
$$\forall \epsilon > 0,\ \exists n_0 > \frac{L}{\epsilon - \epsilon_w} \in \mathbb{N},\ \forall n \geq n_0 :\ ||f - h||_{L^2} < \epsilon \tag{21}$$

Dit bewijst dus de convergentie van h naar f met L_2 norm en dus van het neuraal netwerk met één laag naar de functie f. We kunnen ook de puntsgewijze fout afschatten. We gebruiken dezelfde feiten zoals hierboven al gebruikt.

$$\begin{aligned}
|f(x) - h_w(x)| &= |f(x) - h(x) + h(x) - h_w(x)| \\
&\leq max_{0\leq i\leq n-1} \left[max_{x\in[x_i,x_{i+1}]} f(x) - min_{x\in[x_i,x_{i+1}]} f(x) \right] + \epsilon_w \\
&\leq max_{0\leq i\leq n-1} \left[L|x - y| \right] \leq max_{0\leq i\leq n-1} \left[L|x_{i+1} - x_i| \right] + \epsilon_w \\
&\leq \frac{L}{n} + \epsilon_w
\end{aligned} \tag{22}$$

We hebben nu universele approximatie aangetoond voor functies die voldoen aan volgende voorwaarden:

- $\lim_{x\to-\infty}\sigma(x)=0$ en $\lim_{x\to+\infty}\sigma(x)=c$ voor een $c\in\mathbb{R}$
- σ is continu op $\mathbb{R}$
- $\sigma'(x)\geq 0 \quad \forall x\in\mathbb{R}$
- $\lim_{x\to\infty} x\sigma'(x)=0$



Figuur 6: Een sigmoidal neuraal netwerk dat de sinus functie benadert met vier nodes voor $w=100$ en $w=5$

3.5.2 `ReLu`

We kunnen dit idee ook uitwerken voor de `ReLu` functie, gedefinieerd als $\texttt{ReLu}(x)=max(0,x)$. Enkel gaat het moeilijker zijn om een trap functie, na te bootsen. Dit is echter geen probleem, we kunnen de functie f benaderen met stuksgewijze lineaire functies. Hiermee valt al meteen de factor w die naar oneindig moest gaan (uit (18)). Zoals voorgaand krijgen we bij toevoeging van nodes in de eerste en enkele laag, meer rechten. We zullen zo een node voorstellen opnieuw als $s_i(x)$. We zullen net zoals bij de sigmoidale functies voor gemak alle

eerste weights gelijk stellen aan een w. We krijgen dus:

$$\begin{aligned} s_i(x) &= \texttt{ReLu}(wx + b_i) \\ s(x) &= \texttt{ReLu}(\sum_{i=1}^{n} w_i s_i(x) + b) \end{aligned} \tag{23}$$

We willen dat s een stuksgewijs lineaire functie is op een interval $[a, b]$ met $n-1$ tussenpunten a_i, $a_0 = a$ en $a_n = b$, zodat $[a_i, a_{i+1}]$ even groot is voor alle i. Elk deelinterval met respectievelijke richtingscoëfficiënt r_i op interval $[a_{i-1}, a_i]$. Deze slang van rechten zou beginnen op de functiewaarde van a, deze nemen van even als $f(a) > 0$ voor verwarring te voorkomen (ook al is f niet gedefinieerd). Merk op dat we deze waarden liefst positief houden door de definitie van de `ReLu` functie. Dit vormt echter geen probleem omdat we over een compact gebied makkelijk het beeld kunnen verhogen door sommatie met een term zodat het minimum positief is op dat gebied. Dit vormt geen verschil in het gedrag van die zo bepaalde functie f. Met neurale netwerken gaan dan ook enkel op zoek naar patronen en gedrag van functies. Men heeft geen neuraal netwerk nodig om een sommatie uit te voeren. Om dit resultaat te verkrijgen stellen we volgende parameters voor:

- $b_i = -a_{i-1}w$
- $b = f(a)$
- w_i's zijn oplossingen van volgend stelsel $\left\{ (\sum_{k=1}^{i} w_k)w = r_i \quad \forall i = 1, \ldots, n-1 \right.$

Om dit allemaal te vereenvoudigen stellen we vanaf nu gewoon $w = 1$. We krijgen dus:

$$\begin{aligned} s_i(x) &= \texttt{ReLu}(x - a_{i-1}) \\ s(x) &= \texttt{ReLu}(\sum_{i=1}^{n} w_i s_i(x) + f(a)) \end{aligned} \tag{24}$$

We bekijken of deze waarden kloppen. Voor $x = a$ willen we dus de functiewaarde van a. Per definitie weten we dat $a < a_i \quad \forall i = 1, \ldots, n$, en dus $s_i(a)$ zal dus nul geven voor alle i, met als gevolg dat $s(a) = \texttt{ReLu}(f(a)) = f(a)$. Voor waarden x in het interval $[a_{i-1}, a_i]$ willen we dat de rico r_i is. We hebben we ook dat geldt $x \leq a_i < a_{i+1} < \ldots$ en dus de termen $s_j(x)$ worden opnieuw nul voor $j > i$. We krijgen dan voor x:

$$s(x) = \sum_{k=1}^{i} w_k(x - a_{k-1}) \; + f(a)$$

met afgeleide:

$$s'(x) = \sum_{k=1}^{i} w_k = r_i$$

Dit laat het gewenste resultaat zien. We moeten ons geen zorgen maken over discontinuïteiten want `ReLu` is zelf een continue functie en we voeren met een neuraal netwerk enkel operaties uit die continuïteit overdragen.
We kunnen nu een continue functie f benaderen over een interval $[a, b]$ met behulp van de Lagrange interpolatie. We weten al dat elke node in onze eerste laag een deel rechte voorstelt. We delen het interval dus op in n aantal delen, met n het aantal nodes. In elk van deze deelintervallen stellen we parameters zodat het neuraal netwerk de functie benaderd met een lineaire functie door interpolatie. Met andere woorden op een deel interval $[a_i, a_{i+1}]$, met $x \in [a_i, a_{i+1}]$, willen we volgende functie verkrijgen:

$$y(x) = \frac{f(a_{i+1}) - f(a_i)}{a_{i+1} - a_i}(x - a_i) + f(a_i)$$

De r_i worden dus $\frac{f(a_i - f(a_{i-1})}{a_i - a_{i-1}}$. De puntsgewijze fout kunnen we als volgt aantonen. Merk op dat dit sterk analoog zal zijn aan het bewijs van het theorema van Taylor.

Stelling(puntsgewijze fout van `ReLu`)*Laat $f : [a, b] \to \mathbb{R}$ in $C^2([a, b])$ continue differentieerbaar met $[a, b]$ een compact domein. Zij NN_1 een neuraal netwerk met n nodes in de enige, eerste verborgen laag. De gebruikte activatie is de* **`ReLu`** *functie. Deze gedefinieerd zoals we hierboven hebben beschreven op deel intervallen $[a_i, a_{i+1}]$. Dan geldt voor $x \in [a, b]$:*

$$|f(x) - NN_1(x)| = \frac{|f''(\xi_0)|}{2n^2}(b - a)^2 \tag{25}$$

Met $\xi \in [a, b]$

Bewijs. We werken in het volgende interval $[a_i, a_{i+1}]$. Hierbij zijn al deze deelintervallen even groot, namelijk $\frac{(b-a)}{n}$. Uit de middelwaarde stelling weten we dat we $f'(c) = \frac{f(a_{i+1}) - f(a_i)}{a_{i+1} - a_i}$ voor een bepaalde $c \in [a_i, a_{i+1}]$. We definiëren volgende functies:

$$A(x) = \frac{f(x) - f'(c)(x - a_i) - f(a_i)}{(x - a_i)(x - a_{i+1})} \qquad x \in (a_i, a_{i+1})$$
$$G(s) = f(s) - \frac{f(a_{i+1}) - f(a_i)}{a_{i+1} - a_i}(s - a_i) - f(a_i) - A(x)(s - a_1)(s - a_{i+1})$$

De functie $G(s)$ bevat de functie $A(x)$, maar deze A is slecht een constante in functie van s. Er geldt duidelijk dat voor $s = a_i$, $s = a_{i+1}$ en $s = x$, $G(s) = 0$. Gebruikmakende van de stelling van Rolle, weten we het volgende:

$$\exists d \in [a_i, x] : \; G'(d) = 0$$
$$\exists e \in [x, a_{i+1}] : \; G'(e) = 0$$

We hebben opnieuw twee nulpunten en kunnen we dus opnieuw de stelling van Rolle toepassen.

$$\exists \xi \in [d, e] : \ G''(\xi) = 0 \tag{26}$$

De tweede afgeleide luidt als volgt:

$$G''(s) = f''(s) - 2A(x) \tag{27}$$

Hier uit volgt dus:

$$\begin{aligned} & G''(\xi) = f''(\xi) - 2A(x) = 0 \\ \Longrightarrow & A(x) = \frac{f''(\xi)}{2} \\ \Longrightarrow & f(x) - f'(c)(x - a_i) - f(a_i) = \frac{f''(\xi)}{2} \cdot (x - a_i)(x - a_{i+1}) \qquad x \in (a_i, a_{i+1}) \end{aligned}$$

De termen $(x - a_i)$ en $(x - a_{i+1})$ kunnen we afschatten door de volledige lengte van het interval $[a_i, a_{i+1}]$. Deze is $(b-a)/n$. Merk op dat als x een tussenpunt a_i is, dat de fout dan nul is. We stellen dus het volgende:

$$\forall x \in [a, b], \exists \xi \in [a, b] : \ |f(x) - NN_1(x)| \leq \frac{|f''(\xi)|}{2n^2}(b - a)^2 \tag{28}$$

Als we nu $\xi = argmax_{x \in [a,b]} |f''(x)|$ nemen. krijgen we de gewenste uitkomst. □

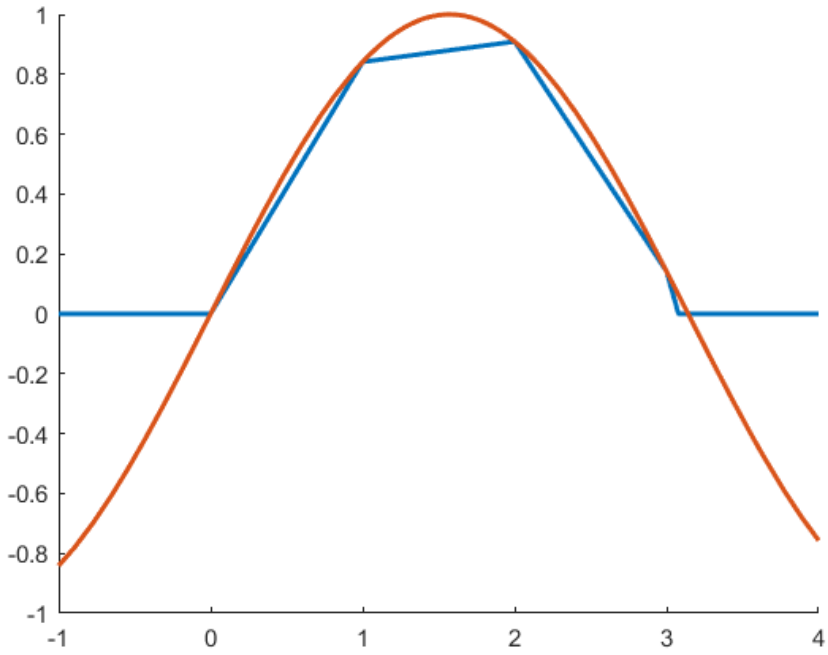
We kunnen hiermee fout met L_2 norm bekijken over een interval $[a, b]$:

$$\begin{aligned} ||f - NN_1||_{L_2} &= \left(\int_a^b |f(x) - NN_1(x)|^2 dx \right)^{1/2} \\ &\leq \left(\int_a^b \left[\frac{f''(\xi)}{2n^2}(b - a)^2 \right]^2 dx \right)^{1/2} \\ &\leq \left(\left[\frac{f''(\xi)}{2n^2} \right]^2 (b - a)^5 \right)^{1/2} \\ &= \frac{|f''(\xi)|}{2n^2}(b - a)^{5/2} \end{aligned}$$

We hebben dus:

$$||f - NN_1||_{L_2} \leq \frac{|f''(\xi)|}{2n^2}(b - a)^{5/2} \qquad \xi \in [a, b] \tag{29}$$

Deze gaan duidelijk naar nul voor hogere n en dus meerdere nodes.



Figuur 7: Een `ReLu` neuraal netwerk dat de sinus functie benadert met vier nodes

3.5.3 Algemeen

We proberen nu universele approximatie aan te tonen voor een algemene functie. We definiëren volgende verzameling:

$$\Phi = \{f \in C^1(\mathbb{R}) \mid lim_{x\to+\infty} f(x) = 0 \vee lim_{x\to-\infty} f(x) = 0,\ supp(f) \neq \varnothing,\ f|_{supp(f)}\ injectief\} \tag{30}$$

Het is duidelijk dat sigmoidale functies hieraan voldoen. Merk op dat het limiet maar naar nul kan in een van de twee oneindigheden. Dit komt immers omdat de functie continu moet zijn en injectief op de support, een support die ook niet leeg mag zijn. Bij twee limiet naar nul en continuïteit impliceren we dat de functie niet injectief is en dit kan dus niet. Er geldt ook dat op de support de functie dus strikt monotoon dalend of stijgend zal zijn. Over heel het domein zal deze ook monotoon zijn, maar niet strikt. De afgeleide zal dus ook altijd positief of negatief zijn
We definiëren een deelverzameling voor functies zoals `ReLu`.

$$\Psi = \{f \in C^2(\mathbb{R}) stuksgewijs \mid \exists M \in \mathbb{R},\ \forall x \geq M \vee \forall x \leq M : f(x) = 0,\ supp(f) \neq \varnothing,\ f|_{supp(f)} injectief\} \tag{31}$$

Er geldt dus $\Psi \subset \Phi$. Voor alle functies in Φ kunnen we hun limiet beschrijven als volgt:

$$\forall \epsilon > 0,\, \exists M(\epsilon) > 0 \in \mathbb{R},\, \forall x \geq M \vee \forall x \leq M :\, |f(x)| < \epsilon \tag{32}$$

Bij de functies uit Ψ geldt uiteraard dat $f \equiv 0$ voor M groot of klein genoeg. Voor een willekeurige activatie functie $\sigma \in \Phi$, definiëren we opnieuw een neuraal netwerk als volgt:

$$s_i(x) = \sigma(\pm(x - a_i) + M(\epsilon))$$
$$NN_1(x) = \sigma(\sum_{i=0}^{n} w_i s_i(x) + b)$$

Hierbij hangt M dus af van een gegeven epsilon, zoals in de limiet definitie. Zodat voor alle epsilon, afhankelijk van x, s_i naar nul gaat. Die afhankelijkheid van x hangt af van het teken er voor, plus of min. Voor activatiefuncties uit Ψ is M dus ook gewoon vast zoals in Ψ zijn definitie. We zullen volgende onderscheiding maken

1. $\lim_{x\to+\infty} \sigma(x) = 0 \;\Rightarrow\; \sigma(-(x - a_i) + M(\epsilon))$
2. $\lim_{x\to-\infty} \sigma(x) = 0 \;\Rightarrow\; \sigma(+(x - a_i) + M(\epsilon))$

Dit zodat als x kleiner is dan een a_i, dan zal het verschil $(x - a_i)$ respectievelijk boven of onder M liggen voor als σ boven of onder M naar nul gaat, of al is. Met andere woorden, of symbolen, geldt dan:

$$x < a_i \implies |\, \sigma(\pm(x - a_i) + M(\epsilon))\, | < \epsilon, \quad \forall \epsilon > 0 \tag{33}$$

Zonder verlies van algemeenheid beschouwen we het plus geval in volgende bewerkingen. Als men nu $NN_1(a_0)(= NN_1(a))$ neemt en omdat $a_0 \leq a_i$ voor alle i, krijgt men het volgende:

$$NN_1(a_0) = \sigma(\sum_{i=0}^{n} w_i \cdot \epsilon + b) \tag{34}$$

Wetende dat σ C^1 en we werken op een compact domein, kunnen we stellen dat σ Lipschitz continu is. We krijgen dan:

$$|\, \sigma(\sum_{i=0}^{n} w_i \cdot \epsilon + b) - \sigma(b)\, | \leq L\, \epsilon\, |\sum_{i=0}^{n} w_i|$$

Hierbij is L de Lipschitz constante. De som is eindig voor $n \in \mathbb{N}$ en dus ook goed gedefinieerd. Dit zal dus convergeren. We stellen dan $b = \sigma^{-1}(f(a))$ zodat voor alle epsilon geldt $|NN_1(a_0) - f(a)| < \epsilon$. Merk op dat we opnieuw f zullen moeten transformeren (indien

nodig) zodat deze in het beeld ligt van de gekozen activatiefunctie. Dit is een triviale lineaire transformatie $af + b$. Voor de volgende punten a_i lossen we vergelijkingen op als volgt om w_{i-1} te bepalen. We doen dit voor $f(a_1)$. We krijgen:

$$NN_1(a_1) = \sigma(\sum_{i=1}^{n} w_i\epsilon + w_0\sigma((a_1 - a_0) + M) + \sigma^{-1}(f(a))) = f(a_1) \tag{35}$$

We stellen w_0 zodat dit klopt voor w naar oneindig.

$$w_0 = \frac{\sigma^{-1}(f(a_1)) - \sigma^{-1}(f(a_0))}{\sigma((a_1 - a_0) + M)} \tag{36}$$

Merk op dat een $M(\epsilon)$ voor geen singulariteiten zal zorgen. Wetende dat de term (zoals in dit voorbeeld) $a_1 - a_0$ altijd positief gaat zijn en omdat het teken bij (1) in de lijst hierboven zo is gekozen, zal dit uitdraaien naar het oneindig waar de activatiefunctie niet nul is. Merk op dat de activatie functie niet nul kan zijn aan beide kanten van oneindig volgens onze definitie. Voor een M is w_0 nog altijd goed gedefinieerd. Deze zal immers gewoon een waarde verschillend van nul zijn, als σ niet naar oneindig gaat aan zijn andere kant (de niet nul kant), of zal gewoon nul worden als σ wel naar oneindig gaat. Voor alle punten a_i kunnen we zo een vergelijking opstellen en deze oplossen, zodat voor alle i geldt dat $|NN_1(a_i) - f(a_i)| < \epsilon$ voor een M in functie van epsilon. Dit zal resulteren dat de functie $NN_1(x)$ puntsgewijs zal kunnen convergeren naar een functie die we zullen noteren als $NN_1^*(x)$. Het verschil in deze functies kan men noteren in functie van w:

$$|NN_1(x) - NN_1^*(x)| < \epsilon \tag{37}$$

We hebben nu dus dat het neuraal netwerk gelijk is aan f op de tussenpunten voor w naar oneindig, afhankelijk of de activatie functie al dan niet in Ψ zit. Als de activatie functie in $\Phi \backslash \Psi$ zit, zal voor epsilon verschillend zijn van nul. Als deze in Ψ zit hebben we al dat epsilon gelijk is aan nul. Het is al duidelijk dat meer nodes, meer tussenpunten betekenen en dus meer juiste benaderingen.

Omdat we voor functies uit $\Phi \backslash \Psi$ niet kunnen garanderen of aantonen, binnen dit werk en deze zijn tijdgebrek, dat het limiet NN_1^* continu, zullen we deze verzameling even achterwegen laten en spreken we verder enkel over functies uit Ψ. Als men de nodige voorwaarden vindt voor Φ, kan men de komende redenering gebruiken.

We weten dat NN_1^* continu is omdat we voor de Ψ verzameling niet werken met een limiet en er gewoon geldt $NN_1 = NN_1^*$, waarbij NN_1 duidelijk continu is. Dit wetende kunnen we voor $x \in [a_i, a_{i+1}]$ het volgende doen:

$$\begin{aligned} |f(x) - NN_1(x)| &\leq |f(x) - s(x)| + |s(x) - NN_1| \\ &\leq \frac{|f''(\xi_0)|}{2n^2}(b-a)^2 + \frac{|NN_1''(\xi_1)|}{2n^2}(b-a)^2 \end{aligned} \tag{38}$$

Hierbij is de functie $s(x)$ zoals we deze gezien hebben in sectie (3.5.2). We bedoelen dus de stuksgewijze lineaire functie die enkel op de tussenpunten a_i gelijk zal zijn aan f. Deze zal dus ook gelijk zijn aan NN_1 op deze tussenpunten, zodat s beide functies tegelijk benaderd.

$$\forall i = 0, ..., n : \ s(a_i) = f(a_i) = NN_1(a_i) \tag{39}$$

Merk op dat we werken op een compact domein $[a_i, a_{i+1}]$ en omdat we zowel f als σ in $C^2(\mathbb{R})$ nemen, zal er een maximum (en minimum) bestaan voor de tweede afgeleiden en zijn $f''(\xi_0)$ en $NN_1''(\xi_1)$ goed gedefinieerd.
We kunnen de waarden voor xi nu zo kiezen zodat deze over heel $[a, b]$ geldt.

- $\xi_0 = argmax_{x\in[a,b]}|f''(x)|$
- $\xi_1 = argmax_{x\in[a,b]}|NN_1''(x)|$

Voor de L_2 norm krijgen we dan het volgende:

$$\begin{aligned}
||f - NN_1||_{L_2} &= \left(\int_a^b |f(x) - NN_1(x)|^2 dx \right)^{1/2} \\
&\leq \left(\int_a^b \big[\frac{|f''(\xi_0)| + |NN_1''(\xi_1)|}{2n^2}(b-a)^2 \big]^2 dx \right)^{1/2} \\
&\leq \left(\big[\frac{|f''(\xi_0)| + |NN_1''(\xi_1)|}{2n^2} \big]^2 (b-a)^5 \right)^{1/2} \\
&= \frac{|f''(\xi_0)| + |NN_1''(\xi_1)|}{2n^2}(b-a)^{5/2}
\end{aligned} \tag{40}$$

3.5.4 Meerdere Lagen

Nu we dit weten over de activatie functies kunnen we dit idee ook uitbreiden tot twee en meerdere aantal lagen via inductie. We weten dat dit geldt voor een laag. Voor twee lagen kunnen we gebruik maken van de Taylor polynoom in een getal c binnen het compact domein $[a, b]$, van een gladde functie. We werken dit uit met een algemene error term voor een laag. We noteren deze met een simpele ϵ_m, waarbij m het aantal nodes is in de eerste laag. Zoals eerder vermeld gaan we bij de activatie functies er vanuit dat deze mooi zijn herschaald, zodat f zich afbeeldt op het respectievelijke beeld van de activatie functie. Dit zodat we f kunnen herdefiniëren als $f := \sigma^{-1}(f)$. Zodat we een gewogen som hebben, 'zonder' activatiefunctie, om te benaderen. We schrijven ook Taylor benaderingen van orde k als volgt op:

$$f_k(x) = \sum_{i=0}^{k} \frac{f^{(i)}(c)}{i!}(x-c)^i$$

We weten dat we de fout van deze Taylor benadering dan als volgt kunnen gelijk stellen

$$|f - f_k| = \frac{|f^{(k+1)}(\zeta_0)|}{(k+1)!}|x-c|^{k+1} \qquad \zeta_0 \in [x,c] \vee [c,x]$$

Omdat we werken op een compact domein kunnen we $(x-c)$ van boven afschatten door $(b-a)$. Verder is de $(k+1)!$ ook exponentieel. Merk ook op de operaties binnen de 2de laag een gewogen som is van neurale netwerken met een laag. Met deze neurale netwerken met een laag kunnen we dan vervolgens de deelfunctie $(x-c)^k$ benaderen volgens de hierboven beschreven methoden. We stellen de gewichten dan gelijk aan de factoren $\frac{f^{(k)}(c)}{k!}$ voor een c. De fout van een neuraal netwerk $NN_2(x)$ met twee hidden layers met respectievelijk n en m nodes geldt dan:

$$\begin{aligned}
|NN_2(x) - f(x)| &= |NN_2(x) - f_m(x) + f_m(x) - f(x)| \\
&\leq |f_m(x) - f(x)| + |NN_2(x) - f_m(x)| \\
&\leq \frac{|f^{(m+1)}(\zeta_0)|}{(m+1)!}|x-c|^{m+1} + |\sum_{k=0}^{m}\frac{f^{(k)}(c)}{k!}|f_k(x) - NN_1^k(x)|\,| \\
&\leq \frac{|f^{(m+1)}(\zeta_0)|}{(m+1)!}|x-c|^{m+1} + |\epsilon_1 \sum_{k=0}^{m}\frac{f^{(k)}(c)}{k!}| \\
&\leq \frac{|f^{(m+1)}(\zeta_0)|}{(m+1)!}|x-c|^{m+1} + \epsilon_1(|f(c+1)| + \frac{|f^{(m+1)}(\zeta_1)|}{(m+1)!})
\end{aligned} \tag{41}$$

We laten dus de nodes in de tweede laag elks een afgeleiden van de functie f benaderen. Dit kan omdat ze elks dus een neuraal netwerk met één laag voorstellen. De gewogen som van de tweede laag is dus een Taylor polynoom die benaderd wordt. Voor een begrensde afgeleiden gaat deze fout dan ook naar nul voor m en n naar $+\infty$. We bekijken de functie f immers op een compact gebied, waardoor $f(c+1)$ niet meer is dan een constante. De breuk van $(x-c)^m < (b-a)^m$ op $m!$ gaat ook naar nul gaan. Er geldt namelijk dat voor eender welke $c \in \mathbb{R}^+$:

$$\lim_{n\to\infty} \frac{c^{n+1}}{(n+1)!} \Big/ \frac{c^n}{n!} = \lim_{n\to\infty} \frac{c}{n+1} = 0 \tag{42}$$

Er volgt dan uit het kenmerk van d'Alembert dat de breuk $(x-c)^n/n!$ naar nul zou gaan. Ook gebruiken we dat

$$\sum_{k=0}^{m}\frac{f^{(k)}(c)}{(k)!} = f(c+1) - \frac{f^{(m+1)}(\zeta_1)}{(m+1)!} \qquad \zeta_1 \in [c, c+1] \tag{43}$$

Als $c+1$ buiten het interval $[a,b]$ ligt kan men alsnog dit toepassen, omdat $[a,b] \cap [c,c+1]$ compact is in $\mathbb{R}$ en dus is het beeld ook begrensd voor de continue functie f en is $f(c+1)$

dus begrensd. Men kan dit idee ook uitbreiden voor $\mathcal{L}$ hidden lagen met elks n_i nodes voor $1 \leq i \leq \mathcal{L}$. We kunnen dit proces herhalen door op dezelfde manier voor $\mathcal{L}$ lagen de Taylor som te benaderen met de nodes in laag $\mathcal{L}$ die elks neurale netwerken van $\mathcal{L}-1$ lagen voorstellen. Om deze afschatting makkelijker te maken, nemen we volgende notaties aan :

$$\begin{aligned} R^0_m &:= \frac{|f^{(m+1)}(\zeta)|}{(m+1)!}|x-c|^{m+1} \qquad \text{voor } \zeta \in [x,c] \vee [c,x] \\ R^1_m &:= \frac{|f^{(m+1)}(\zeta)|}{(m+1)!} \qquad \text{voor } \zeta \in [c, c+1] \\ C &:= |f(c+1)| \end{aligned} \tag{44}$$

We krijgen als afschatting van de fout voor een neuraal netwerk met $\mathcal{L}$ lagen en m_i nodes per i-ste laag de volgende formule:

$$|NN_{\mathcal{L}}(x) - f(x)| \leq \sum_{i=2,\,\mathcal{L}>1}^{\mathcal{L}} (R^0_{m_i} \cdot \prod_{k=i+1,\,i<\mathcal{L}}^{\mathcal{L}} (C + R^1_{m_k})) + \epsilon_{m_1} \prod_{i=2,\,\mathcal{L}>1}^{\mathcal{L}} (C + R^1_{m_i}) \tag{45}$$

We duiden hieronder van waar deze formule komt. Uit (42) halen we dus dat alle termen $R^{0,1}_m$ naar nul gaan voor $m \to \infty$. Hieruit volgt dus dat de fout naar nul gaat voor $\mathcal{L}$ lagen als alle nodes in alle lagen m_i naar oneindig gaan in een punt x. We zien dat voor $\mathcal{L} = 2$ deze formule geldt (zie (41)). We kunnen deze nu met inductie aantonen. Stel dat deze al geldt voor $\mathcal{L} = L$. We krijgen dan als fout voor $\mathcal{L} = L+1$:

$$\begin{aligned} |NN_{L+1}(x) - f(x)| &= |NN_{L+1}(x) - f_{m_{L+1}}(x) + f_{m_{L+1}}(x) - f(x)| \\ &\leq R^0_{m_{L+1}} + |\sum_{k=0}^{L+1} \frac{f^{(k)(c)}}{k!}|f_k(x) - NN^k_L(x)|\,| \\ &\leq R^0_{m_{L+1}} + (C + R^1_{m_{L+1}}) \cdot |NN_L(x) - f(x)| \\ &\leq R^0_{m_{L+1}} + \sum_{i=2,\,\mathcal{L}>1}^{\mathcal{L}} (R^0_{m_i}(C + R^1_{m_{L+1}}) \cdot \prod_{k=i+1,\,i<\mathcal{L}}^{\mathcal{L}} (C + R^1_{m_k})) \\ &\quad + \epsilon_{m_1}(C + R^1_{m_{L+1}}) \cdot \prod_{i=2,\,\mathcal{L}>1}^{\mathcal{L}} (C + R^1_{m_i}) \\ &= \sum_{i=2,\,L+1>1}^{L+1} (R^0_{m_i} \cdot \prod_{k=i+1,\,i<L+1}^{L+1} (C + R^1_{m_k})) + \epsilon_{m_1} \prod_{i=2,\,L+1>1}^{L+1} (C + R^1_{m_i}) \end{aligned}$$

Dit concludeert het inductiebewijs.
Om nu de Universele Approximatie Stelling aan te tonen moeten we nog L_2 convergentie

aantonen. Om dit makkelijk aan te tonen laten we nog even zien dat R_m^0 in (44) ook afgeschat kan worden, zodat deze onafhankelijk is van een x.

$$\frac{|f^{(m+1)}(\zeta)|}{(m+1)!}|x-c|^{m+1} \leq \frac{|f^{(m+1)}(\zeta)|}{(m+1)!}(b-a)^{m+1} =: R_m^0$$

Dit zodat onze afschatting van $|NN_{\mathcal{L}}(x) - f(x)|$ onafhankelijk is van een x en we deze als een constante over x kunnen beschouwen.

$$\begin{aligned} ||NN_{\mathcal{L}} - f||_{L^2} &= \left(\int_a^b |NN_{\mathcal{L}} - f(x)|^2 dx \right)^{1/2} \\ &\leq \sqrt{b-a} \cdot \left(\sum_{i=2,\, \mathcal{L}>1}^{\mathcal{L}} (R_{m_i}^0 \cdot \prod_{k=i+1,\, i<\mathcal{L}}^{\mathcal{L}} (C + R_{m_k}^1)) + \epsilon_{m_1} \prod_{i=2,\, \mathcal{L}>1}^{\mathcal{L}} (C + R_{m_i}^1) \right) \end{aligned} \tag{46}$$

Net zoals onze uitkomst bij (45) zal deze afschatting voor dezelfde redenen naar nul gaan voor m_i naar oneindig. We hebben dus nu de uniforme approximatie theorema bewezen voor elk aantal lagen in een neuraal netwerk met een sigmoidale, of `ReLu` als activatie functie.

Theorema (Universele Approximatie Theorema) *Zij $NN_{\mathcal{L}}$ een neuraal netwerk uit de verzameling NN, met $\mathcal{L}$ lagen met respectievelijk m_i nodes voor de i-ste laag $(0 < i \in \mathbb{N} \leq \mathcal{L})$. Zij f een gladde functie, uit C^∞, over een compact domein $\mathcal{A} \subseteq \mathbb{R}$. Dan geldt:*

Er bestaat een willekeurige dichte benadering van een neuraal netwerk, voor activatie functies in Ψ of een sigmoidale activatie functie uit (3.5.1), naar f of een triviale lineaire transformatie van f, $f := af + b$ in functie van het aantal nodes m_i:

- sigmoidale activatie functie uit (3.5.1)

  $\forall \epsilon > 0, \exists (w_0, m_1^0, \ldots, m_L^0) \in \mathbb{N}^L : \forall (w, m_1, \ldots, m_L) \geq (w_0, m_1^0, \ldots, m_L^0)$ elementsgewijs

- activatiefunctie uit Ψ

$$\forall \epsilon > 0,\ \exists (m_1^0, ..., m_{\mathcal{L}}^0) \in \mathbb{N}^{\mathcal{L}} :\ \forall (m_1, ..., m_{\mathcal{L}}) \geq (m_1^0, ..., m_{\mathcal{L}}^0)\ elementsgewijs$$

$$\implies ||NN_{\mathcal{L}} - f||_{L^2} < \epsilon$$

De expliciete fout van deze benaderingen in functie van het aantal nodes kan benaderd worden zoals in vorige delen (46), (40), (20).

Bewijs. zie hierboven. □

Als we in (45) alle lagen evenveel nodes geven, $m_i = m_j = m,\ \forall 0 < i, j \leq \mathcal{L}$, krijgen we volgende versimpelde uitdrukking:

$$\begin{aligned} |NN_{\mathcal{L}}(x) - f(x)| &\leq R_m^0 \sum_{i=2,\,\mathcal{L}>1}^{\mathcal{L}} (C + R_m^1)^{\mathcal{L}-i} + \epsilon_m (C + R_m^1)^{\mathcal{L}-1} \\ &\leq R_m^0 (\mathcal{L} - 1)(C + R_m^1)^{\mathcal{L}-2} + \epsilon_m (C + R_m^1)^{\mathcal{L}-1} \end{aligned}$$

We kunnen dit analoog doen voor (46), L_2 convergentie. In deze formule zit duidelijk geen structuur die ons zou kunnen helpen met de architectuur van een neuraal netwerk te optimaliseren. Bij toevoeging van lagen $\mathcal{L}$ wordt deze afschatting enkel groter. Deze zal enkel dalen bij toevoeging van nodes in alle lagen. In 5.3.1 bespreken we een experiment omtrent de topologie van het netwerk in meer detail.

4 Hyperparameter Optimalisatie

4.1 Hyperparameters

Als men spreekt over hyperparameters binnen de context van machine learning, bedoelt men de parameters die gekozen worden voordat het leerproces begonnen is.Dit zijn dan parameters zoals: de topologie van het netwerk, de learning rate, de cost functie, de activatiefuncties of zelfs convergentie methoden (variaties op gradiënt descent). Deze hebben dan ook enkel invloed op het leerproces, en geen invloed meer op het netwerk zelf. De juiste parameters kiezen is essentieel om niet alleen het leerproces te versnellen, maar ook de kwaliteit van het netwerk te verhogen. Het manueel tunen van deze parameters kan nogal wat tijd in beslag nemen en zal vaak niet het beste of zelfs een goed resultaat opleveren. Daarom zal in dit hoofdstuk gekeken worden, om met behulp van verscheidene methodes dit proces te automatiseren en optimaliseren.

Formeel gezien, beschouw Θ de hyperparameterruimte. Dit is het cartesisch product van de ruimtes waar elke hyperparameter zich in bevindt. Nu zoekt men een $\theta \in \Theta$ zodat $\theta = \underset{\hat{\theta} \in \Theta}{\text{argmin}}\, \text{f}(\hat{\theta})$. De functie f is hier de loss functie van het neuraal netwerk dat een optimale set hyperparameters nodig heeft.
Hier onder zullen verschillende methodes besproken worden, die gebruikt worden om een maxima te vinden binnen deze hyperparameterruimte.

4.2 Grid search

Grid search is een methode waarbij men itereert over de verschillende hyperparametercombinaties binnen (een deel van) de parameterruimte. Vervolgens kijkt men welke hyperparametercombinatie de laagste cost heeft. Dit algoritme is rekenkundig zeer intensief, maar het proces is parallelliseerbaar. Het algoritme is,zoals random search, niet erg efficiënt.

4.3 Random Search

Random search is een alternatieve methode voor hyperparameteroptimalisatie. In tegenstelling tot grid search, waarbij alle mogelijke combinaties van hyperparameters worden uitgeprobeerd, kiest random search willekeurig verschillende combinaties om te evalueren. Wederom is deze methode enorm parallelliseerbaar aangezien geen enkele uitvoering van random search afhangt van de volgende of vorige.

4.4 Bayesiaanse Optimalisatie

Bayesiaanse optimalisatie is een methode om de extreme waarden te vinden van een moeilijk te evalueren 'black box' functie $f(x)$, waar traditionele optimalisatiemethodes niet werken. Het vergt geen kennis van de afgeleide van de hypothetische functie. Er zijn verschillende varianten op deze methode. Wetende dat de functie moeilijk te evalueren is, werkt het algoritme op zo weinig mogelijk geëvalueerde punten om de extreme waarde op een compact gebied te bepalen. Het probleem wordt in [**?**] voor dit algoritme geformuleerd als volgt:

$$max_{x \in \mathcal{A}} f(x)$$

waarbij $\mathcal{A}$ een compact gebied is op het domein van de functie, in dit geval de configuratieruimte. Bij hyperparameter optimalisatie gaan we uiteraard op zoek naar de set hyperparameters waarvoor de loss functie het laagste is of waar onze R^2 het hoogste is. Voor het zoeken van een minimum kan dit algoritme worden toegepast door te werken met een nieuwe functie $\hat{f}(x) = -f(x)$.
De term 'Bayesiaans' is afkomstig van het Theorema van Bayes voor voorwaardelijke kans:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{47}$$

A zou de effectieve functiewaarden kunnen zijn met B de gegeven geëvalueerde punten, zodat men met $P(A|B)$ de kans voorstelt dat we het gedrag van de functie f kunnen reproduceren met het gedrag van de gegeven punten in B. Dan is de kans $P(B|A)$ de likelihood van onze gekregen punten voor de functie f, oftewel de kans dat de gegeven punten uit B een goed beeld geven voor heel de functie en dus de verzameling A.

4.4.1 Gaussian Process regressie

Voordat men begint bij Bayesiaanse optimalisatie moet eerst verklaard worden wat een gaussian proces is. Gegeven n aantal datapunten $\mathcal{D} = \{(\mathbf{x}, f(\mathbf{x})|\mathbf{x} \in \mathbb{R}, f(\mathbf{x}) \in \mathbb{R}\}$ waarbij $\mathbf{x}$ de inputvector is en $f(\mathbf{x})$ de, in ons geval, scalaire reële waarde.

Een Gaussian proces is een distributie over functies waarbij elke eindige verzameling van punten over zijn compact domein, gezamelijk multivariaat normaal verdeelt zijn.
De nadruk ligt op een distributie over functies, ergo, voor elke x waarde in ons domein, bestaat er een gemiddelde en variantie. Een Gaussian proces is een gezamelijke distributie van een geheel domein waar een continue functie over leeft.

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{48}$$

Een Gaussian proces heeft 2 parameters:

- $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$: Dit is de mean functie oftewel de verwachtingswaarde van de mogelijke outputfuncties, ter hoogte van inputvector x.
- $k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x} - m(\mathbf{x})))(f(\mathbf{x}') - m(\mathbf{x}'))]$

Dit is de covariantie functie, die bekijkt hoe twee functiewaarden elkaar beïnvloeden, afhankelijk van hun afstand $||\mathbf{x} - \mathbf{x}'||$. Logischerwijs hebben punten met gelijkwaardige input vectoren, invloed op elkaars gedrag bij continue functies. Men verwacht dan ook dat de covariantie tussen nabije punten hoog zal zijn. En de covariantie van punten, ver gelegen van elkaar, eerder laag. Een regelmatig gekozen covariantiefunctie is de radial basis functie (RBF) die er als volgt uitziet:

$$k(\mathbf{x}, \mathbf{x}') = e^{-\frac{||\mathbf{x}-\mathbf{x}'||_2}{2L}} \tag{49}$$

De factor $L > 0$ is een parameter die bepaalt hoeveel invloed nabije input waarden hebben op elkaars output. Een hoge L zorgt dat het effect minder snel afzwakt waardoor een datapunt meer invloed globaal heeft, terwijl een lage L er voor zal zorgen dat een punt eerder enkel lokaal invloed heeft. Deze definitie breiden we uit tot $K(X, X') = \left[k(\mathbf{x_i}, \mathbf{x'_j})_{i,j}\right]$, waarbij X,X' een vector input waarden zijn.

Om nu een regressiemodel op te stellen beschouwen we nu $X^\star$, dit is de verzameling van input waarden waarover de regressie wordt uitgevoerd. In praktijk zal dit een rooster van punten zijn, verdeelt over het domein. Men is nu geïnteresseerd in $\mathbf{f}^\star = f(X^\star)$. Het is redelijk om aan te nemen dat de functiewaarden van de punten in het rooster hetzelfde verdeelt zijn als de gekende datapunten. men gaat dus uit van de volgende distributie:

$$\mathbf{f}^\star \sim \mathcal{N}(0, K(X^\star, X^\star)) \tag{50}$$

Hierboven is de prior distributie van de uitkomstwaarden $\mathbf{f}^\star$. Er is nog geen kennis van $\mathcal{D}$ in verwerkt. Dit geeft een gezamelijke distributie van:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^\star \end{bmatrix} \sim \mathcal{N}\left(0, K = \begin{bmatrix} K(X,X) & K(X,X^\star) \\ K(X^\star,X) & K(X^\star,X^\star) \end{bmatrix}\right). \tag{51}$$

Men wil uiteraard dat het model kennis gebruikt die vergaard is door $\mathcal{D}$. Daarom stellen we het conditionele model op van $\mathbf{f}^\star$,gegeven $(\mathbf{x_i}, \mathbf{f_i})$. Dit is genoteerd als: $\mathbf{f}^\star \mid X^\star, X, \mathbf{f}$. In [**?**], vind men het bewijs terug van de parameters van de posterior distributie. In grote lijnen gaat men met behulp van lineaire algebra de inverse berekenen van de matrix K. En vervolgens eigenschappen van de normale verdeling toe te passen. Dit geeft ons voor gemiddelde en standaardafwijking:

- $m(x) = (K(X^\star, X)K(X,X))^{-1}\,\mathbf{f}$
- $\sigma(x)^2 = (K(X^\star,X^\star) - K(X^\star,X)K(X,X))^{-1}\,K(X,X^\star)$

$$\text{zodat } \mathbf{f}^\star \mid X^\star, X, \mathbf{f} \sim \mathcal{N}\left(m(\mathbf{x}), \sigma(\mathbf{x})^2\right)$$

Dit geeft ons nu een distributie van de functiewaardes van $X^\star$ waaruit we, voor elke niet-geobserveerde input vector, trekkingen voor de output waarde kunnen genereren.
In sommige gevallen is het niet altijd correct om er van uit te gaan dat de outputmetingen in een punt x nauwkeurig zijn omwille van onder andere meetfouten of andere onzekerheden. Het is daarom ook niet ongebruikelijk om GP regressie uit te voeren op $y = f(\mathbf{x}) + \epsilon$, ϵ $i.i.d \sim \mathcal{N}(0, \sigma)$. Dit valt echter buiten het bestek van deze scriptie. Voor de geïnteresseerde lezer verwijzen we door naar [**?**].

4.4.2 Acquisitie functie

Het idee is nu om een model op te stellen met een initieel aantal datapunten, en vervolgens iteratief een datapunt bij te voegen en de regressie opnieuw uit te voeren, tot men met een hoge zekerheid kan zeggen waar het maximum of minimum ligt op een bepaald domein. De volgende vraag die dan opkomt, is: "Hoe bepaalt men welk datapunt het beste toegevoegd wordt?" Hier komen acquisitiefuncties bij kijken. Deze functies zullen maximaal zijn bij punten die het interessantste lijken om toe te voegen aan het regressiemodel. Wat de punten interessant maakt is afhankelijk van het type acquisitie functie. Vaak is er een debat over deze vraag, waarbij "exploitation versus exploration" ter sprake komt. Met exploration bedoelt men hier het onderzoeken van nog niet reeds gekende deelgebieden van het domein, en met exploitation bedoelt men het verder onderzoeken van regio's waar reeds goede doelfunctiewaarden werden gevonden. Dus eigenlijk een afweging tussen het modelleren van de surrogaatfunctie en het vinden van een maximum in een lokale regio. De volgende acquisition functies zullen hier op verschillende manieren mee omgaan en verschillende afwegingen

maken.
Hier zullen er enkele acquisition functies besproken worden:

- **probability of improvement**
  Beschouw $I(\mathbf{x}) = \max(f(\mathbf{x}) - f^\star(\mathbf{x}), 0)$. Hier is $f(\mathbf{x})$ het GP model en $f^\star(\mathbf{x})$, de hoogste gemeten output vector van onze data $\mathcal{D}$. I noemt men de improvement functie, deze zal nul zijn als het surrogaat model kleiner is dan de effectieve functiewaarde. Dan is $\hat{\mathbf{x}} = \text{argmax}_\Omega(P(I(\mathbf{x}) > 0))$ het punt wat toegevoegd moet worden aan onze regressie. Probability of improvement zal proberen de kans op een hogere functiewaarden te maximaliseren. Omdat $f(\mathbf{x})$ een Gaussian proces is geldt dat op ieder punt er een normale verdeling plaatsvindt:

  $f(\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x}))$ oftewel $\frac{f(\mathbf{x})-\mu(\mathbf{x})}{\sigma^2(\mathbf{x})} \sim \mathcal{N}(0, 1)$. Dit wil zeggen dat men wilt berekenen wat $P(I(\mathbf{x}) > 0)$ is. Nu geldt:

$$\begin{aligned} P(I(\mathbf{x}) > 0) &= P(f(\mathbf{x}) - f^\star(\mathbf{x}) > 0) \\ &= P(f(\mathbf{x}) > f^\star(\mathbf{x})) = 1 - P(f(\mathbf{x}) < f^\star(\mathbf{x})) \\ &= 1 - \Phi\left(\frac{f(\mathbf{x})^\star - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \end{aligned}$$

  Φ is hier de cumulatieve dichtheidsfunctie van de standaard normaal verdeling.

  Deze functie is een uitstekende kandidaat als acquisitie functie. Een opmerking die hier gemaakt moet worden is dat probability of improvement niet zal kijken naar hoe groot de verbetering effectief is. Deze zal enkel de input vector met een hogere kans op verbetering selecteren. Een 90 % kans met een verbetering van 0.01 is in dit geval beter dan een % 80 kans op een verbetering van 1. Dit is niet altijd gewenst gedrag. Wat in de praktijk vaak gaat gebeuren is dat deze functie redelijk beperkt blijft in zoekdomein, zodra deze een regio van hoge functiewaarden heeft gevonden.

- **Expected improvement** Bij expected improvement wordt niet alleen gekeken naar de kans dat een willekeurig punt een hogere of betere waarde heeft, maar ook hoeveel beter deze is. De functie die hier gemaximaliseerd wordt is dan ook $\mathbb{E}[I(\mathbf{x})]$. Om de formule hiervoor af te leiden wordt er een truc gebruikt. Beschouw z standaardnormaal verdeeld, dan is$f(\mathbf{x}) = \mu(\mathbf{x}) + \sigma(\mathbf{x}) \cdot z$ met $z \sim \mathcal{N}(0, 1)$, normaal verdeeld met gemiddelde $\mu(\mathbf{x})$ en standaardafwijking $\sigma(\mathbf{x})$ zoals hierboven.

  Beschouw nu $\mathbb{E}[I(\mathbf{x})] = \int_{-\infty}^{\infty} I(\mathbf{x})\varphi(z)\,dz = \int_{-\infty}^{\infty} \max(f(\mathbf{x}) - f(\mathbf{x}^\star), 0)\varphi(z)\,dz$ Merk op dat de integraal 0 is, als $f(\mathbf{x}) < f(\mathbf{x}^\star)$. Dit gebeurt als $z < \frac{f(\mathbf{x})^\star - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$, beschouw

dan $z_0 = \frac{f(\mathbf{x})^\star - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$. Het volstaat dus om de integraal te berekenen tussen z_0 en ∞:

$$\begin{aligned}
\mathbb{E}\left[I(\mathbf{x})\right] &= \int_{-\infty}^{\infty} \max(f(\mathbf{x}) - f(\mathbf{x}^\star), 0)\varphi(z)\, dz \\
&= \int_{z_0}^{\infty} (\mu(\mathbf{x}) + \sigma(\mathbf{x})z - f(\mathbf{x}^\star))\varphi(z)\, dz \\
&= \int_{z_0}^{\infty} (\mu(\mathbf{x}) - f(\mathbf{x}^\star))\varphi(z)\, dz + \int_{z_0}^{\infty} \frac{\sigma(\mathbf{x})z}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}\, dz \\
&= (\mu(\mathbf{x}) - f(\mathbf{x}^\star)) \int_{z_0}^{\infty} \varphi(z)\, dz + \sigma(\mathbf{x}) \int_{z_0}^{\infty} \frac{z}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}\, dz \\
&= (\mu(\mathbf{x}) - f(\mathbf{x}^\star))(1 - \Phi(z_0)) + \frac{\sigma(\mathbf{x})}{\sqrt{2\pi}} \int_{z_0}^{\infty} z e^{-\frac{z^2}{2}}\, dz \\
&= (\mu(\mathbf{x}) - f(\mathbf{x}^\star))(1 - \Phi(z_0)) - \frac{\sigma(\mathbf{x})}{\sqrt{2\pi}} \int_{z_0}^{\infty} (e^{-\frac{z^2}{2}})'\, dz \\
&= (\mu(\mathbf{x}) - f(\mathbf{x}^\star))(1 - \Phi(z_0)) - \frac{\sigma(\mathbf{x})}{\sqrt{2\pi}} [e^{-\frac{z^2}{2}}]_{z_0}^{\infty} \\
&= (\mu(\mathbf{x}) - f(\mathbf{x}^\star))\Phi(-z_0) + \sigma(\mathbf{x})\varphi(z_0) \\
&= (\mu(\mathbf{x}) - f(\mathbf{x}^\star))\Phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^\star)}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\varphi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^\star)}{\sigma}\right)
\end{aligned}$$

Dit resultaat is de uitdrukking van expected improvement. Deze maximaliseren geeft het punt in het domein waar men de hoogste verbetering verwacht.

Een Gaussian proces wordt meestal gebruikt voor Bayesiaanse optimalisatie. Hier stelt men een surrogaatfunctie op met de gegeven punten en de gekozen covariantiefunctie. De surrogaat functie gaat om een geschat gemiddelde van een punt gegeven de punten. Het idee van het algoritme is nu eigenlijk vrij simpel: Beschouw een verzameling van startpunten $\mathcal{D}$. Voer op deze punten een gaussian proces regressie uit. Nu tot een bepaald stopcriteria vervuld is: stel de acquisition functie op en maximaliseer deze en voer regressie uit met dit nieuw gevonden punt. Meer in detail:

Algorithm 1 Bayesiaanse optimalisatie met een Gaussian proces

Input :black box functie f, domein D, acquisition functie $A(\mathbf{x})$
Output :beste input vector $\mathbf{x}^\star$,gevonden door het algoritme

Initialiseer Gaussian regressiemodel met $\mathcal{D}$,de initiële observaties

while stopcriterium niet voldaan is **do**
Stel de acquisition functie $A(\mathbf{x})$,gegeven de voorgaande datapunten uit $\mathcal{D}$ op en maximaliseer deze: $\mathbf{x}_{volgend} = \mathbf{argmax}_{\mathbf{x}\in\mathcal{D}}(A(\mathbf{x}))$
Evalueer de black box functie: $y_{\text{volgend}} = f(\mathbf{x}_{\text{volgend}})$
Werk dataset bij: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_{\text{volgend}}, y_{\text{volgend}})\}$
Werk het surrogaatmodel bij met het nieuwe datapunt.
return Beste oplossing $\mathbf{x}^\star$ gevonden in $\mathcal{D}$

Er zijn verschillende mogelijkheden om te gebruiken als stopcriteria. Een veelgebruikte mogelijkheid is kijken naar het verschil tussen opeenvolgende berekende doelfunctiewaarden. Als men na een aantal iteraties geen verbetering meer ziet, of een verbetering kleiner dan ϵ, zal het algoritme stoppen. Bij sommige moeilijk te trainen netwerken kan men ook een maximaal aantal iteraties instellen. Dit kan aantrekkelijk zijn als men maar een beperkte rekentijd heeft, en geen tijd of middelen kan spenderen om te wachten tot de verbetering kleiner is als een bepaalde tolerantie.

4.4.3 Tree-structured Parzen Estimator

Tree-structured Parzen Estimator is een variant van bayesiaanse optimalisatie. In plaats van te werken met een Gaussian proces gaat men hier een dichtheidsfunctie, opgesteld met behulp van Parzen window estimation, gebruiken. Het voordeel hiervan is dat er geen veronderstelling wordt gemaakt over welke distributie de data volgt.
Parzen window estimation:

Parzen window estimation, of Kernel distribution estimation is een niet parametrische schattingsmethode om de dichtheidsfunctie van data te benaderen. Intuïtief gezien zou men verwachten dat waar de meeste datapunten liggen, de dichtheidsfunctie het hoogste is. Men

zou dit zelfs kunnen bekijken als een continue verderzetting van een histogram. Wat men gaat doen is op elk datapunt een functie plaatsen. De schatting van de kansdichtheidsfunctie is dan de som van deze functies. Echter moet er op gelet worden dat de eigenschappen van een dichtheidsfunctie behouden worden, e.g. $\int_\Omega f(x)dx = 1$ en $f(x) \geq 0 \; \forall \; x \in \Omega$. Hierdoor zal men deze functies nog herschalen. Formeel gezien:
Beschouw $h > 0$ de bandwidth van de schatting, $K(x)$ de kernel functie en $(x_i)_{i \leq N}$ de N gegeven datapunten, dan wordt de schatting met bandbreedte gegeven door:

$$f_N(x) = \frac{1}{Nh} \cdot \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right) \tag{52}$$

Er zijn enkele voorwaarden die men aan deze kernel stelt:

$$\begin{aligned} &k(x) \geq 0 && \text{voor alle } x \in \mathbb{R} \\ &\int_{\mathbb{R}} k(x)dx = 1 && \\ &k(x) = k(-x) && \text{voor alle } x \in \mathbb{R} \end{aligned} \tag{53}$$

Enkele voorbeelden zijn:

1. de box kernel:

$$k(x) = \begin{cases} \frac{1}{2}, & \text{als} |x| \leq 1 \\ 0, & \text{elders} \end{cases}$$

2. de trianguliere kernel:

$$k(x) = \begin{cases} 1 - |x| & \text{voor } |x| \leq 1 \\ 0 & \text{elders} \end{cases}$$

3. Epanechnikov:

$$k(x) = \begin{cases} \frac{3}{4\sqrt{5}} \left(1 - \frac{1}{5}x^2\right) & \text{voor } |x| < \sqrt{5} \\ 0 & \text{elders} \end{cases}$$

4. Gaussian:

$$k(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

In 52 is de term $\frac{x-x_i}{h}$ verantwoordelijk voor het verschuiven van x rond x_i. De factor $1/h$ wordt de bandbreedte of bandwidth genoemd. Een kleine h zal ervoor zorgen dat de deelfuncties $K(\frac{x-x_i}{h})$ smal en gecentreerd zijn rond hun respectievelijke datapunt x_i. Bij kernels zoals de box-functie zal h zelfs invloed hebben op het aantal datapunten dat in aanmerking wordt genomen bij elke deelfunctie. Hoe minder datapunten in aanmerking worden genomen per deelfunctie, hoe minder glad de functie zal zijn. Het correct kiezen van deze parameter is uiterst belangrijk voor een nauwkeurige schatting. Is h te klein, dan zal de functie te ruw zijn en veel te veel lokale details tonen. Is h te groot, dan zal de functie zeer glad worden en kunnen eigenschappen van de dichtheidsfunctie verstopt geraken. Voor een detailanalyse verwijzen we naar [**?**].

Nu men bekend is met hoe de Parzen-window estimator een dichtheidsfunctie kan schatten, wordt er verder gekeken naar de effectieve methode. Allereerst beschouwt men bij dit model niet direct de kans op een functiewaarde $f(x)$, gegeven een x waarde. Maar men beschouwt het omgekeerde: de kans op een x waarde, gegeven f(x). Men defineert de kans $p(x|y)$ als volgt:

$$p(x|y) = \begin{cases} l(x) \text{ als } f(x) < y^\star \\ g(x) \text{ als } f(x) \geq y^\star \end{cases} \tag{54}$$

Hierbij is $l(x)$ de dichtheidsfunctie geschat met de datapunten$(x^{(i)}, f(x^{(i)}))$ zodat $f(x^{(i)}) < y^\star$. De dichtheidsfunctie is geschat met de eerder genoemde Parzen window-methode. $g(x)$ is de dichtheidsfunctie geschat met de datapunten $(x^{(i)}, f(x^{(i)})$ zodat $f(x^{(i)}) \geq y^\star$. $y^\star$ is het gamma-percentiel van de geobserveerde functiewaarden, waarbij gamma een hyperparameter is,gekozen door de analyst. Met de vorige definities volgt hier al uit dat $p(x) = \gamma \cdot l(x) + (1-\gamma) \cdot g(x)$. Het is natuurlijk de bedoeling dat men de prior van dit model kan verbeteren met behulp van nieuwe datapunten. Dit is eenvoudig te doen als er een geordende lijst wordt bijgehouden van alle koppels datapunten. Met een binary search kan dan in logaritmische tijd een datapunt in deze lijst toegevoegd worden en vervolgens worden de dichtheidsfuncties opnieuw geschat.

Nu resteert de vraag nog hoe het volgende punt bepaald wordt vanuit een huidig model. Het doel is om een waarde x te zoeken zodat de verbetering van $y = f(x)$ maximaal is. Men vertrekt hier van:

$$p(f(x)|x) = \frac{p(x|f(x)) \cdot p(f(x))}{p(x)} \tag{55}$$

Zodat:

$$\begin{aligned}
\mathbb{E}(I(x)) &= \int_{y^\star}^{\infty} \max(y - y^\star), 0) p(y|x), dy \\
&= \int_{y^\star}^{\infty} (y - y^\star) p(y|x) dy \\
&= \int_{y^\star}^{\infty} (y - y^\star) \frac{p(x|y) \cdot p(y)}{p(x)} dy \\
&= \frac{1}{p(x)} \left[\int_{y^\star}^{\infty} y \cdot p(x|y) \cdot p(y) dy - \int_{y^\star}^{\infty} y^\star p(x|y) \cdot p(y) dy \right] \\
&= \frac{1}{\gamma \cdot l(x) + (1-\gamma) g(x)} g(x) \left[\int_{y^\star}^{\infty} y \cdot p(y) dy - \int_{y^\star}^{\infty} y^\star \cdot p(y) dy \right] \\
&= \frac{g(x) \cdot \int_{y^\star}^{\infty} y \cdot p(y) dy - y^\star \cdot (1-\gamma) g(x)}{\gamma \cdot l(x) + (1-\gamma) g(x)} \\
&= \frac{\int_{y^\star}^{\infty} y \cdot p(y) dy - y^\star \cdot (1-\gamma)}{\gamma \cdot \frac{l(x)}{g(x)} + (1-\gamma)}
\end{aligned} \tag{56}$$

Merk op dat aangezien $y \in [y^\star, \infty[$ geldt dat $p(x|y) = g(x)$.

Met deze uitdrukking kanmen zien hoe de expected improvement gemaximaliseerd kan worden. Punten die een hoge kans hebben om te komen uit de verdeling van g(x) en een lage kans om uit de verdeling van l(x) zijn uiteraard geweldige kandidaten om als volgend punt te beschouwen in onze optimalisatie. In [**?**] is een analoog bewijs terug te vinden wat een uitdrukking heeft gezocht indien men de doelfunctie wilt minimaliseren. Hier is het omgekeerde waar: punten die een hoge kans hebben om tot l(x) te behoren en een lage kans om van g(x) afkomstig te zijn hebben hier voorkeur. Dit hangt uiteraard af van het doel van het model.

5 Rekenresultaten

Een groot deel van deze scriptie is het zelf implementeren van de hiervoor vernoemde systemen, de eerste vraag die gesteld wordt is dan ook: "Welke programmeertaal zullen we gebruiken om deze methoden te implementeren?" Er is een afweging gemaakt tussen een low level taal zoals C, of de populaire high level taal Python. Python had hier duidelijk de voorkeur. Hoewel de hogere performance bij een taal zoals C zeker een groot voordeel is, is het ontwerpen van software veel gemakkelijker en veel sneller in Python. Bij Python moet men niet zelf memory management uitvoeren en heeft veel libraries ter beschikking om software hier snel bovenop te ontwikkelen. Echter zijn we niet vertrokken vanaf bestaande softwarepakketten zoals TensorFlow of PyTorch. Een deel van de opdracht was het zelf implementeren van deze concepten. Dit is gedeeltelijk gelukt, maar bepaalde optimalisaties die buiten Python plaatsvinden bij bestaande libraries zijn niet geïmplementeerd. Dit zorgde er voor dat zelfgeschreven code niet altijd op dezelfde snelheid werkte als bestaande platformen, maar voor deze doeleinden is het resultaat adequaat.

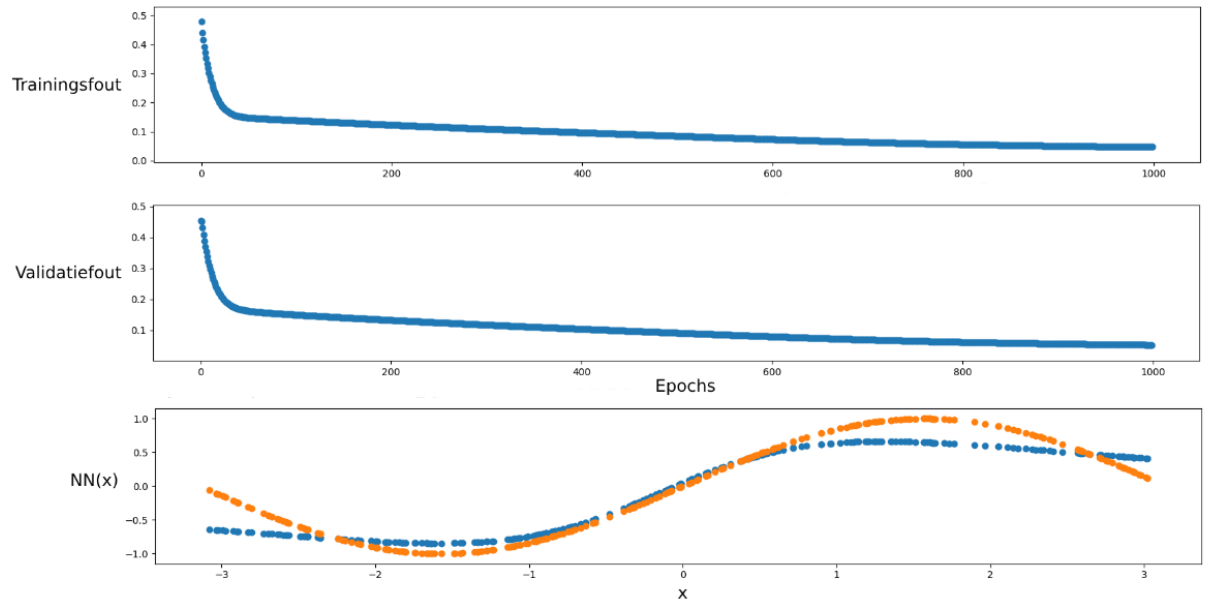
Allereerst was het doel een modulair neuraal netwerk op te bouwen waarbij verschillende eerder uitgelegde parameters ingeseld kunnen worden zoals: Learning rate, topologie van het neuraal netwerk (aantal layers en aantal nodes per layer), mini-batch grootte in het geval van stochastische gradiëntmethode, activatiefunctie en aantal epochs.

Ons "Toy Problem" was een willekeurig gegenereerde dataset van coördinatenpunten $(x, \sin(x))$. Het doel was uiteraard de afstand tussen sin(x) en NN(x) te minimaliseren, specifieker de L^2 norm. Dit stelde ons in staat ons netwerk te testen met een dataset die niet veel verwerking nodig had, voordat we konden beginnen met een zwaarder probleem zoals classificatie.

5.1 Het neurale netwerk

5.1.1 Gradient descent

Allereerst beschouwen we een eerste versie van een neuraal netwerk, getraind met de gradiënt descent methode.

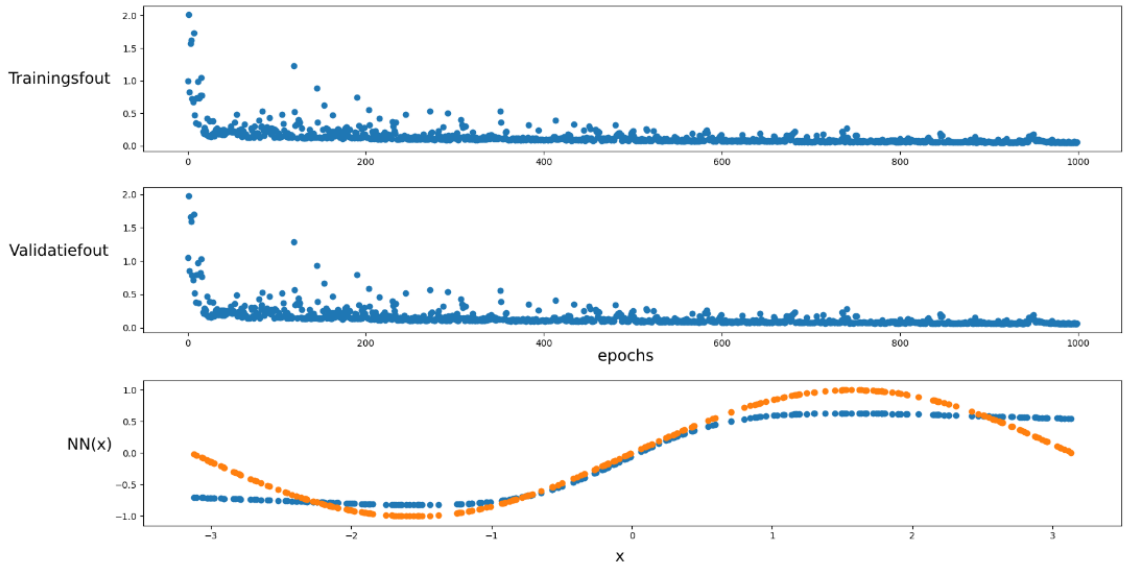


Figuur 8: Gradiëntmethode: resultaat en fout t.o.v. validatie- en trainingsdata

In 8 vindt u de eerste versie van het neuraal netwerk terug, uitgerust met de standaard gradiënt descent methode terug te vinden in sectie 3.3 en getrained met 200 willekeurig geselecteerde datapunten. Allereerst merken we op dat de rekentijd van dit netwerk substantieel langer is dan de implementaties die volgen. Dit is gedeeltelijk omdat hier de componenten van de gradiënt nog numeriek werden benaderd in plaats van expliciet bepaald met backpropagation. Ook vergt de gradiënt descent methode meer rekentijd dan de hierna besproken stochastische gradiënt descent methode. Het aantal epochs dat hier doorlopen wordt is ook niet zo groot als hierna. Ook op te merken is dat naarmate men richting epoch 1000 neigt, geen grootte verbetering meer zien is. Alle rekentijd die daarna zou plaatsvinden is in principe niet meer nuttig. Om dit te voorkomen wordt er vaak gekeken naar een vorm van early stopping. Dit houdt in dat, als de cost functie van het neurale netwerk niet significant verandert over een aantal iteraties, de simulatie tijdig kan worden gestopt om onnodige rekentijd te voorkomen. Een correcte threshold vinden om deze early stop te implementeren is echter niet zo eenvoudig als het lijkt.

5.1.2 Stochastische gradiënt descent

Een vervolg in het ontwikkelingsproces was het implementeren van de computationele goedkopere variant: de stochastische gradiëntmethode.



Figuur 9: Stochastische Gradiëntmethode: resultaat en fout t.o.v. validatie- en trainingsdata

Hier wordt opgemerkt dat zowel de validatie- als trainingsfout geleidelijk afnemen richting een minimum, zij het met aanzienlijke variabiliteit vanwege de aard van de stochastische gradiënt. Dit komt doordat slechts 1 datapunt per iteratie wordt gebruikt. Een vergelijking tussen de traditionele gradiëntmethode en de stochastische variant toont aan dat de kosten lager liggen bij de eerste. Echter, dit is geen eerlijke vergelijking. Zoals eerder opgemerkt, vereist de stochastische methode slechts een fractie van de rekentijd. Een eerlijker vergelijkingspunt zou zijn om de stochastische methode meer epochs te laten doorlopen. Wanneer het aantal epochs wordt verdubbeld, wat nog steeds resulteert in een kortere rekentijd, zal de stochastische methode uiteindelijk de traditionele gradiëntmethode evenaren en zelfs overtreffen.

Een variant hierop is de mini-batch gradient descent methode. In plaats van 1 datapunt mee te nemen in de berekening zal deze, afhankelijk van een hyperparameter:'mini batch', een aantal datapunten gebruiken.

5.2 Performance grafiek

Een interessante vraag die opkwam, is hoe snel een neuraal netwerk leert en hoe men de competentie van het netwerk beoordeelt. Een maat die men hiervoor kan gebruiken is de R^2 coëfficiënt, gedefinieerd als:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}.$$

waarbij:

- y_i is het i-de geobserveerde datapunt
- $\hat{y}_i$ is de schatting van het i-de geobserveerde datapunt volgens het regressiemodel
- $\bar{y}$ is het gemiddelde van de geobserveerde datapunten

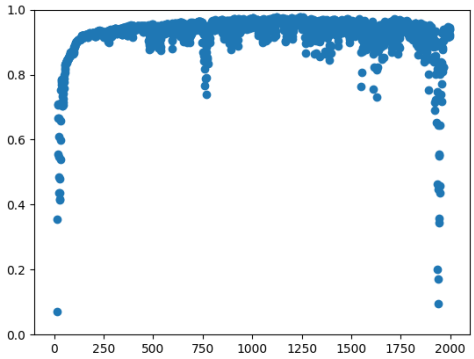
Het is nu bekend dat een persoon, die vertrekt vanaf nul voorkennis, zijn competentie volgt volgens deze differentiaalvergelijking:

$$\frac{dP(t)}{dt} = k \cdot (M - P(t)) \tag{57}$$

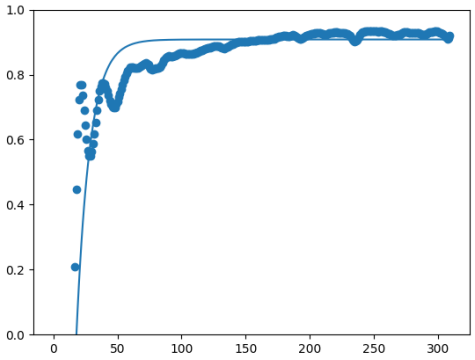
waarbij $P(t)$ is de competentie van een persoon in functie van tijd, k en M zijn hier parameters die specifiek zijn aan de persoon. Het oplossen van deze differentiaalvergelijking geeft ons de oplossing:

$$P(t) = M - Me^{-kt} \tag{58}$$

Als men uitgaat van de veronderstelling dat $P(0) = 0$. Hierbij geeft M een theoretische bovengrens van iemand zijn bekwaamheid. Voor R^2 zou dit $M = 1$ kunnen zijn. De factor k is hier een soort learning rate, maar dan niet in de context van neurale netwerken. Vervolgens is bekeken hoe dat de R^2 coëfficiënt evolueert in functie van aantal epochs bij het neuraal netwerk. Een verrassend resultaat is aan het licht gekomen:



(a) R^2 van het netwerk in functie van epochs



(b) R^2 van het netwerk in functie van epochs, uitgeglad door het gemiddelde te nemen van punten in de omgeving

Met behulp van k-nearest neighbour methode, kunnen we de data smoother maken zodat deze een mooier, gladder geheel vormt. Het resulterende verband blijkt gelijkenis te vertonen met de hiervoor genoemde differentiaal vergelijking. dit is empirisch bewijs dat een neuraal netwerk inderdaad deze trend volgt bij het trainen van een netwerk.

5.3 hyperparameter tuning

In deze subsectie zullen er verschillende methoden worden besproken om inzicht te brengen in de hyperparameterruimte. Het eerste deel zal gaan over de topologie van een neuraal netwerk en de inzichten die onze experimenten hier teweeg brachten. Tot slot wordt er een kleine Bayesiaanse optimalisatie uitgevoerd met behulp van de library GPY opt. Deze library bevat eenvoudige functies om het visualiseren van de optimalisatie eenvoudig te maken.

5.3.1 Topologie van het netwerk

We halen uit volgend experiment dat er wel degelijk een verschil is tussen layouts van nodes in een neuraal netwerk. We deden een experiment om de functie $f(x) = sin(x) + cos(5x)$ te benaderen met verschillende configuraties, namelijk de volgende:

1. `[1,200,1]`
2. `[1, 100,1]`
3. `[1,100,100,1]`
4. `[1,50,50,1]`
5. `[1,25,25,25,25,1]`
6. `[1,50,50,50,50,1]`
7. `[1,25,25,25,25,25,25,25,25,1]`

Hierbij stellen we de architectuur voor als een vector waarbij elk element een laag voorstelt. De in- en output zijn dus het eerste en laatste element en zijn beide een, voor een 1-dimensionale functie. De eerste hidden layer zou dus bijvoorbeeld 200 nodes hebben en de volgende hidden layer, het volgende element in de vector. Het doel van dit experiment is om te onderzoeken of er verschillen optreden tussen configuraties van lagen en nodes met hetzelfde totaal aantal nodes. We laten alle configuraties 200 keer trainen met 250 epochs met een mini-batch grootte van 32, op basis van 100 trainingspunten en 100 validatiepunten waarmee met deze laatste de R^2 waarde wordt berekend en opgeslagen. Onderstaande data zijn de p-waarden als output van een `R` commando voor paarsgewijs t-testen met Bonferroni correctie. Hierbij is de nulhypothese dat de gemiddelde niet verschillend zijn. We gebruiken Bonferronni correctie om er voor te zorgen dat we niet de nulhypothese foutief gaan verwerpen, wat kan gebeuren bij meerdere paarsgewijze testen. Bonferonni correctie houdt de kans op een type I fout bij hypothese testen onder controle.

```
                                  [1, 100, 1] [1, 100, 100, 1] [1, 200, 1]
[1, 100, 100, 1]                     < 2e-16     -                -
[1, 200, 1]                          0.43        < 2e-16          -
[1, 25, 25, 25, 25, 1]               < 2e-16     0.19             < 2e-16
[1, 25, 25, 25, 25, 25, 25, 25, 25, 1] < 2e-16   < 2e-16          < 2e-16
[1, 50, 50, 1]                       < 2e-16     2.3e-06          < 2e-16
[1, 50, 50, 50, 50, 1]               < 2e-16     < 2e-16          < 2e-16

                                     [1, 25, 25, 25, 25, 1]
[1, 100, 100, 1]                     -
[1, 200, 1]                          -
[1, 25, 25, 25, 25, 1]               -
[1, 25, 25, 25, 25, 25, 25, 25, 25, 1] < 2e-16
[1, 50, 50, 1]                       8.6e-14
[1, 50, 50, 50, 50, 1]               4.3e-10

                                     [1, 25, 25, 25, 25, 25, 25, 25, 25, 1]
[1, 100, 100, 1]                     -
[1, 200, 1]                          -
[1, 25, 25, 25, 25, 1]               -
[1, 25, 25, 25, 25, 25, 25, 25, 25, 1] -
[1, 50, 50, 1]                       < 2e-16
[1, 50, 50, 50, 50, 1]               < 2e-16

                                     [1, 50, 50, 1]
[1, 100, 100, 1]                     -
```

```
[1, 200, 1]                             -
[1, 25, 25, 25, 25, 1]                  -
[1, 25, 25, 25, 25, 25, 25, 25, 25, 1] -
[1, 50, 50, 1]                          -
[1, 50, 50, 50, 50, 1]                  < 2e-16
```

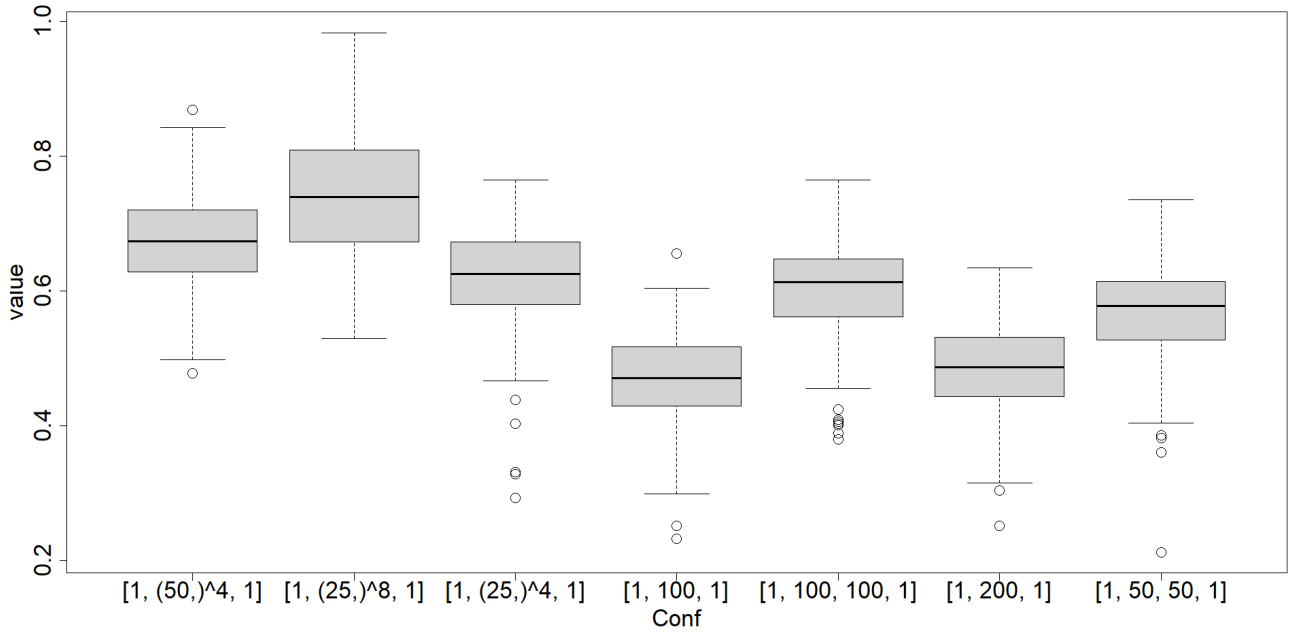
Deze output zegt dat alle configuraties gemiddeld significant verschillend zijn behalve bij:

- `[1, 200, 1]` met `[1, 100, 1]` ($p = 0.43$)
- `[1, 25, 25, 25 25, 1]` met `[1, 100, 100, 1]` ($p = 0.19$)

Deze zijn niet significant op 95% significantieniveau. Voor configuraties die hetzelfde aantal nodes bevatten zijn de waarden dus wel significant verschillend. Sterker nog, omdat de student-t verdeling symmetrisch is zal dit ook resulteren dat de configuraties ook onderling significant groter of kleiner zullen zijn zoals ze op de boxplot (hieronder) staan (11). Namelijk, een tweezijdige t-test moet zijn verwerpingsgebied verdelen over de 2 staarten van de verdeling. Een eenzijdige t-test test maar langs een kant van de verdeling en zal daar dus een groter verwerpingsgebied hebben vergeleken met een tweezijdige t-test aan die staart van de verdeling. Verwerpen bij een tweezijdige test resulteert dus op een verwerping voor een linkse of rechtste eenzijdige test. Uit de verschillen van de populaties kunnen we een vermoeden opstellen omtrent welke het grotere gemiddelde heeft, ze zijn immers al significant verschillend.
Uit onze formule in (46),

$$||NN_{\mathcal{L}} - f||_{L_2} \leq \sqrt{b-a} \cdot \left(\sum_{i=2,\, \mathcal{L}>1}^{\mathcal{L}} (R^0_{m_i} \cdot \prod_{k=i+1,\, i<\mathcal{L}}^{\mathcal{L}} (C + R^1_{m_k})) + \epsilon_{m_1} \prod_{i=2,\, \mathcal{L}>1}^{\mathcal{L}} (C + R^1_{m_i}) \right)$$

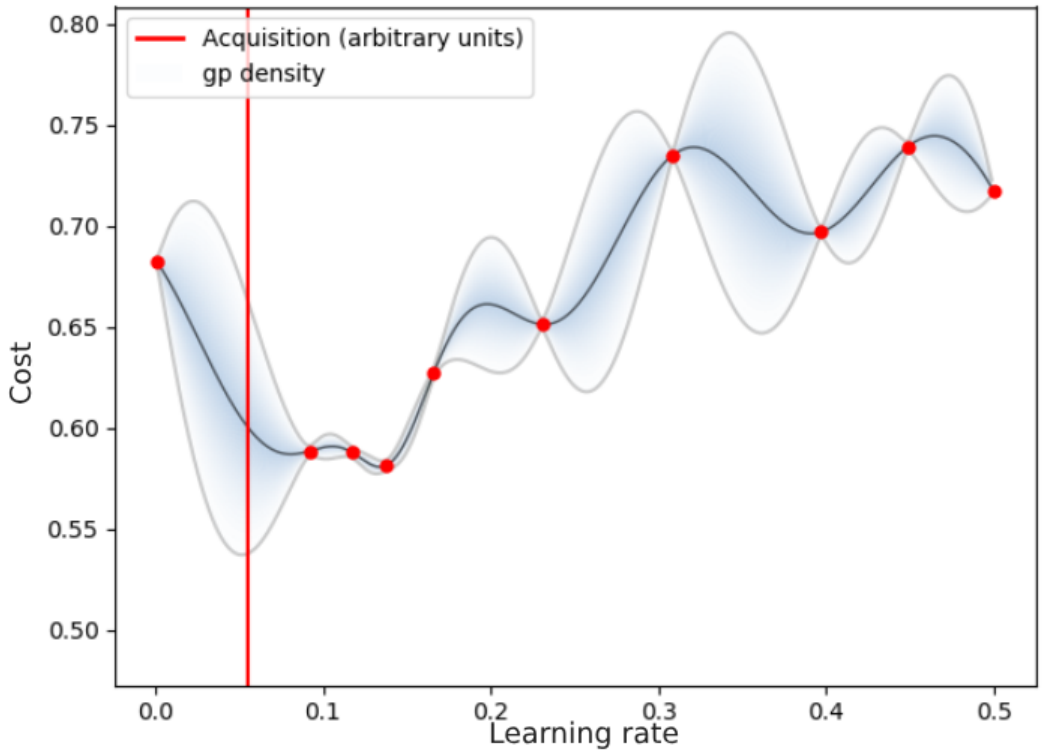
halen we dat toevoeging van lagen zou leiden tot grotere fouten. Meerdere lagen zijn extra termen in de som, plus ze maken de termen ook groter omdat er minder nodes in elke nieuwe laag komen. De formule suggereert dat een meer gespreide verdeling een grotere fout oplevert. Dit klopt echter helemaal niet met de resultaten uit ons experiment. De formule is namelijk een bovenafschatting van het echte gedrag. We slechts hieruit dat de fout zal dalen bij toevoeging van nodes. De resultaten van het experiment zijn logischer: toevoeging van lagen, is een toevoeging van complexiteit en dus van capaciteit om te benaderen. Het spreiden van nodes heeft dus wel degelijk (experimenteel) zin. Spijtig is de formule te 'grof' afgeschat om hier een onderliggende structuur in te vinden. We kunnen spreiding van nodes enkel aanmoedigen.



Figuur 11: Data van het experiment in boxplots

5.3.2 optimalisatie van learning rate

In dit onderdeel laten we het Bayesiaanse optimalisatie algoritme met behulp van Gaussian processes los op de learning rate van een neuraal netwerk. Getraind op de functie $f(x) = sin(x) + cos(5x)$. Men genereert 200 datapunten, 40 hiervan worden op voor trainen gebruikt en de rest voor valideren, en laat herhalend een netwerk trainen met behulp van deze trainingsdata voor 250 epochs. De gebruikte acquisitie functie is expected improvement met de radial basis function als kernel. Het netwerk bestaat uit 2 hidden layers met 8 nodes. Het doel is hier niet om een netwerk voldoende te trainen,maar kijken welke learning rate het beste presteert. Vervolgens wordt de cost waarde beschouwd, berekent met de L_2 norm. Deze wordt geminimaliseerd in de iteraties.

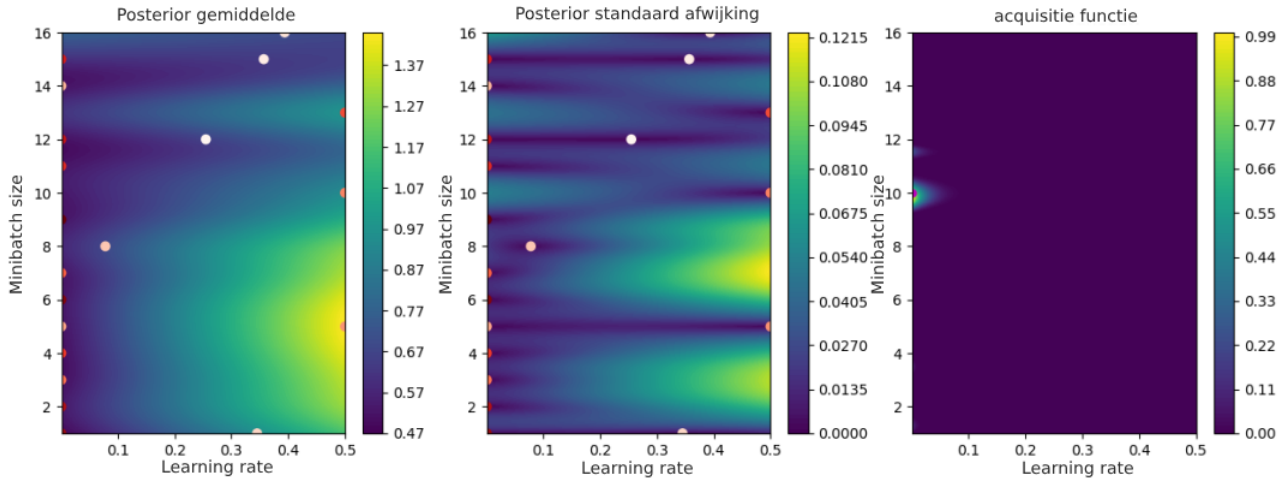


Figuur 12: Bayesiaanse optimalisatie uitgevoerd op de learning rate, De rode stippen representeren nieuw toegevoegde datapunten.

Wat men hier ziet ligt redelijk in lijn met de resultaten van een manuele zoektocht. De learning rate mag zeker niet te hoog zijn. Dan zou men in een zekere zin over een minima springen na elke stap. Een learning rate die te klein is, zou bij een laag aantal epochs, niet in de buurt van een minimum geraken.

5.3.3 optimalisatie van 2 parameters

Hier zal de optimalisatie beperkt blijven tot 2 parameters. Hierdoor blijft het mogelijk om te visualiseren wat er aan het gebeuren is.De opstelling van het neurale netwerk is analoog aan het model hiervoor. De keuze is gemaakt voor wederom de learning rate, alsook de minibatch size. Nu hebben we dus te maken met zowel een continue parameter (learning rate) en een discrete parameter (minibatch size). Merk op dat dit nog altijd geen categorische parameter is zoals activatiefunctie. Er zit een bepaalde orde in.



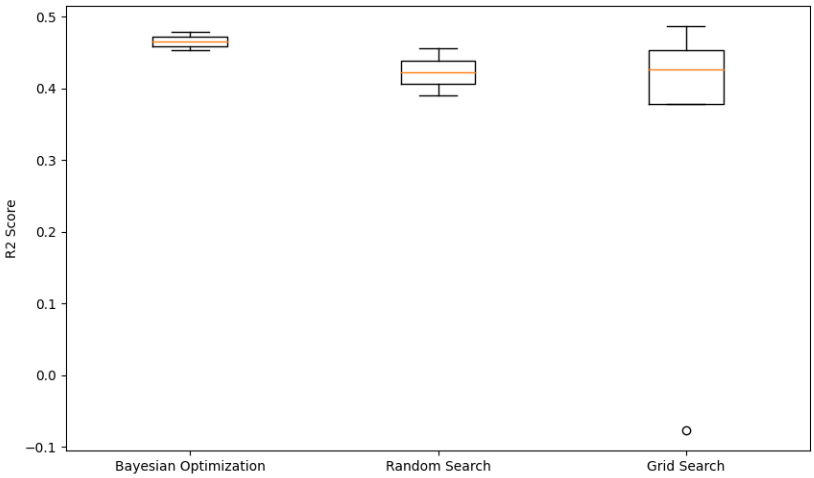
Figuur 13: Rechts: verwachte waarde van cost functie
Midden: Standaardafwijking van de cost functie
Links: Acquisition functie (Expected improvement)

Een belangrijke kanttekening hierbij is dat men voorzichtig moet zijn met enkel te vertrouwen op de verwachte gemiddelde costfunctie. Een voorbeeld hiervan is dat op het eerste zicht, de functie constant blijft over de gehele strook van minibatch 6-8.hier is echter de standaardafwijking zeer hoog. Dit wil zeggen dat de waarde die hier wordt weergeven hoogst waarschijnlijk niet nauwkeurig is. Dat gezegd zijn is te zien dat een hogere learning rate weer wijst naar een lagere cost. Ook is te zien dat hoe hoger de batchsize, hoe lager de cost is. Dit resultaat is te vergelijken met een eerdere conclusie de we hadden getrokken bij 5.1.2. Hoe groter de minibatch size, hoe meer de stochastische gradiënt (of beter gezegd minibatch-gradiënt) op een normale gradiëntmethode gaat lijken. En zoals hiervoor vermeld zal de gewone gradiënt beter presteren bij een gelijk aantal epochs. Belangrijk om erbij te vermelden is dat een lagere minibatch zal ervoor zorgen dat de rekentijd hoger wordt.

5.3.4 vergelijking van methoden

Ten slotte worden de drie methoden: grid search, random search en bayesiaanse optimalisatie met elkaar vergeleken. Dit gebeurt door elke methode meerdere keren uit te voeren op een willekeurige dataset. Bijvoorbeeld, bayesiaanse optimalisatie wordt zeven keer uitgevoerd met drie initiële punten, random search wordt tien keer geprobeerd, en grid search wordt verdeeld in tien datapunten. Vervolgens worden ze in 3 boxplots weergeven in 14:

Enkele merkwaardigheden die hier te zien zijn. Bayesiaanse optimalisatie is niet alleen



Figuur 14: Vergelijking van optimalisatiemethoden

gemiddeld gezien de beste presteerder, maar zal ook het meest consistent het beste resultaat geven. Ook is te zien dat random search niet zo slecht presteert als verwacht, maar het is uiteraard niet competitief met de 2 andere methoden. Grid search kan in ideale gevallen beter presteren als bayesiaanse optimalisatie. Dit is uiteraard ook een logisch geval. Als de zoekruimte van Gridsearch fijn is opgesteld, zal deze ook een waarde vinden met hoge performantie. Dit is alleen niet haalbaar computationeelgewijs.

Als we kritisch kijken naar de hierboven uitgevoerde analyses doorheen het hoofdstuk valt er een trend op. Enkel kijken naar een prestatieparameter zoals de R^2 coëfficiënt of de cost geeft ons slechts een zijde van het verhaal. Als men bayesiaanse optimalisatie zou laten lopen op de topologie van het netwerk, met enkel de cost om te minimaliseren, zal de conclusie altijd maar een groter en groter netwerk zijn. Zoals eerder vermeld is dit niet computationeel haalbaar. Een logisch vervolg hierop zou zijn de rekentijd in acht te nemen in bij het optimalisatie proces. Hiermee zou er een afweging kunnen gemaakt worden tussen rekentijd en performantie.

6 Conclusies

In dit onderzoek zijn verschillende methoden voor het optimaliseren van neurale netwerken onderzocht en vergeleken. Python werd gekozen boven C vanwege de gebruiksvriendelijkheid en de uitgebreide bibliotheken, ondanks de potentieel hogere prestaties van C. Het zelf implementeren van de concepten zonder gebruik te maken van bestaande softwarepakketten zoals TensorFlow of PyTorch bracht uitdagingen met zich mee, maar leidde tot een mooi resultaat voor de doeleinden van deze studie.
We hebben ook de Universele Approximatie Stelling aangetoond voor sigmiodale, `ReLU` en een verzameling algemenere functies op een relatief intuïtieve wijze. We hebben ook convergentie voor meerdere lagen aangetoond, bij convergentie van één laag. De experimenten toonden aan dat een hogere learning rate en een grotere minibatch size resulteren in een lagere cost functie. minibatch size heeft echter wel een invers verband met de rekentijd. De bevindingen onderstrepen dat de stochastische gradientmethode efficiënter wordt met een grotere minibatch size, wat het meer doet lijken op een normale gradientmethode, die beter presteert bij een gelijk aantal epochs.
Bij de vergelijking van optimalisatiemethoden bleek bayesiaanse optimalisatie niet alleen gemiddeld het beste te presteren, maar ook het meest consistent resultaten te leveren. Random search presteerde beter dan verwacht, maar bleef achter bij grid search en bayesiaanse optimalisatie. Grid search kan in ideale gevallen beter presteren dan bayesiaanse optimalisatie, vooral als de zoekruimte fijn is ingesteld, hoewel dit computationeel vaak niet haalbaar is.

DANKWOORD
We willen onze dank betuigen aan professor Inneke Van Nieuwenhuyse en professor Fred Vermolen, voor hun goede begeleiding, ondersteuning en expertise tijdens deze bachelor thesis. Hun nodige inzichten en kritische feedback hebben ons geholpen om de juiste vragen te formuleren en op zoek te gaan naar de bijhorende antwoorden.

References

[1] R. A. Adams and C. Essex. *Calculus a complete Course, 10th Edition.* Pearson, 2022.

[2] A. Y. C. akmak. *Universal Approximation Theorem.* Istanbul Technical University, 2022.

[3] J. Bergstra, R. Bardenet, B. Kégl, and Y. Bengio. Algorithms for hyper-parameter optimization. 12 2011.

[4] V. M. C. Eric Brochu and N. de Freitas. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning.* 2020.

[5] T. Janssens. *Hyperparameter tuning for Artificial Neural Networks applied to inverse mapping parameter updating.* Eindhoven University of Technology, 2022.

[6] D. P. Kingma and J. L. Ba. *ADAM: A method for stochastic optimization.* University of Amsterdam, OpenAI and University of Toronto, 2017.

[7] A. Kratsios. *The Universal Approximation Property.* McMaster University, 2020.

[8] Y. Liang and Z. Shi. *Lecture 4 Approximation II.* University of Wisconsin–Madison, 2022.

[9] M. Nielsen. Neural networks and deep learning. `http://neuralnetworksanddeeplearning.com/`, 2019.

[10] I. Panageas and G. Piliouras. *Gradient Descent Only Converges to Minimizers: Non-Isolated Critical Points and Invariant Regions.* Georgia Institute of Technology and Singapore University of Technology Design, 2016.

[11] E. Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065 – 1076, 1962.

[12] C. Rasmussen, O. Bousquet, U. Luxburg, and G. Rätsch. Gaussian processes in machine learning. *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures, 63-71 (2004)*, 3176, 09 2004.

[13] M. Schmidt. *CPSC 540: Machine Learning, Convergence of Gradient Descent.* University of British Columbia, 2017.

[14] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning*. Cambridge University Press, The Hebrew University, Jerusalem and University of Waterloo, Canada, 2019.

[15] G. STRANG. *Linear Algebra and Learning from Data*. Wellesley-Cambridge Press, Massachusetts Institute of Technology, 2019.

7 Appendix

De code die gebruikt is deze thesis is terug te vinden op de githubpagina:Bachelorproef. Via deze link kan u er geraken:
`https://github.com/SamBe-2158037/bachelorProef.git`