Studieretningsopgave

Navn og klasse: Christian Ottosen, 3mms

Første fag og lærer: 7615A Matematik, Frank Carlsen (fc@marselisborg-gym.dk)

Andet fag og lærer: 6678C Informatik, Per Selmann Andreassen (pa@marselisborg-gym.dk)

Område

Machine Learning og kunstige neurale netværk

Opgaveformulering

Du skal redegøre for matematikken bag kunstige neurale netværk.

Du skal ud fra et selvvalgt projekt konstruere en model for et kunstigt neuralt netværk, som kan visualiseres i et it-system.

Endelig skal du diskutere muligheder og begrænsninger inden for kunstige neurale netværk.

Opgavebesvarelsen skal have et omfang på 15-20 normalsider á 2400 anslag inkl. mellemrum

CHRISTIAN OTTOSEN

Resume

Dette studieretningsprojekt har i samspil med fagene matematik og informatik undersøgt kunstige neurale netværk. Herunder er der blevet redegjort for de underliggende matematiske formler og operationer samt de strukturelle dele nødvendige for at udarbejde en Machine Learning model. På baggrund af dette er et selvvalgt projekt i form at et kunstigt neuralt netværk blevet konstrueret. Modellen er blevet kodet i programmeringssproget Python, hvori de matematiske algoritmer såsom backprograpgion er blevet inkorporeret og anvendt. I denne opgave har matematik og informatik sammen skabt rammerne for udarbejdelsen af en klassifikationsmodel, der med høj nøjagtighed kan foretage forudsigelser og klassificere billeder. I forlængelse af dette analyseres modellens struktur og de anvendte matematiske funktioner. Ved brug af den iterative metode kunne denne model med fordel blive eksperimenteret med samt optimeres gennem en længere træningsproces. Til sidst bliver mulighederne og begrænsningerne for kunstige neurale netværk diskuteret. Herunder indses nødvendigheden for dataforberedelse samt problematikkerne bag opsamling af store datasæt og deres betydning for træningsprocessen. Generelt konkluderes det at faget matematik har være et vigtigt redskab til at beskrive matematikken bag kunstige neurale netværk, hvortil informatik har implementeret den matematiske notation.

Indholdsfortegnelse

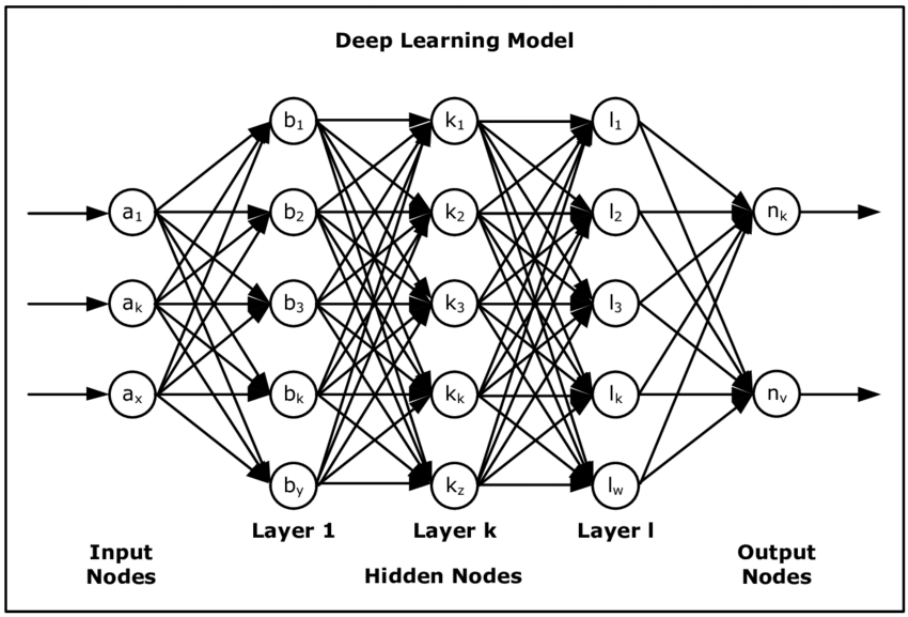
1. Indledning

I denne opgave undersøges Machine Learning og kunstige neurale netværk. Der findes flere forskellige former for Machine Learning såsom Supervised Learning, der alle kan løse komplekse problemer. Kunstige neurale netværk er komplekse matematiske funktioner, der gennem en træningsproces kan trænes til at udføre en given opgave. Dette gøres gennem Supervised Learning, hvilket anvender store labelled datasæt samt matematiske algoritmer såsom backpropagation algoritmen. Denne algoritme anvender differentialregning, matrixmultiplikation samt gradienter til at optimere parametrene i et kunstigt neuralt netværk og dermed minimere omkostningsfunktionen. Omkostningsfunktionen samt andre målinger anvendes til at beskrive hvor godt et kunstige neurale netværk præsterer.

I denne opgave vil der blive redegjort for de 7 trin bag konstruktionen af en Machine Learning model. Den samme trinbaseret tilgang vil senere blive anvendt i samarbejde med den iterative metode til at udarbejde en klassifikationsmodel i programmeringssproget Python. Modellens konstruktion vil dertil blive analyseret, og de anvendte matematiske formler og funktioner bag det neurale netværk vil blive forklaret. Til sidst vil fordelene og ulemperne bag kunstige neurale netværk diskuteres. Herunder diskuteres problematikkerne med at indsamle store mængder data samt fænomenet overfitting og den tidskrævende proces bag optimeringen af disse modeller.

2. Redegørelse for Machine Learning og Kunstige neurale netværk:

Machine Learning (ML) er en gren indenfor kunstig intelligens[1]. Det helt centrale i Machine Learning er anvendelsen af store mængder data, hvormed computeralgoritmer kan lære og forbedres over tid. Disse algoritmer danner nogle modeller baseret på den data, programmet har set tidligere. Dette kaldes ofte træningsdata. Ud fra denne model kan algoritmen, uden eksplicit kode, behandle nye data og komme med forudsigelse eller tage nogle bestemte valg. På denne måde kan intelligente computerprogrammer skabes. Der findes mange forskellige modeller inden for Machine Learning, og en af de mere kendte er det kunstige neurale netværk (Artificiel Neural Network). Kunstige neurale netværk er et forsøg på at genskabe strukturen og funktionaliteten bag den menneskelige hjerne. På samme måde som hjernen er kunstige neurale netværk sammensat af neuroner, der kan signalere til hinanden.



Figur 1, model af et kunstigt neuralt netværk[2]

Ovenfor på figur 1 kan strukturen for sådan en model ses. Hver cirkel repræsenterer et neuron, der hver især har deres egne aktiveringsværdier. Alle linjer mellem neuronerne beskriver de sammenhænge/forbindelser, der er mellem dem. På denne måde kan et neuron videregive information til andre neuroner. Hvert lags output bliver input for det næste lag, og der er dermed en sammenhæng

[1]Afsnittet er skrevet ud fra (Sanderson, But what is a neural network? | Chapter 1, Deep learning, 2017)
[2] Kilde: https://www.researchgate.net/profile/Will-Serrano/publication/313408173/figure/fig8/AS:669010169438243@1536515862984/Artificial-Neural-Network-Deep-Learning-model.png

mellem dem. Fælles for alle neurale netværk er det første og sidste lag. Kunstige neurale netværk kan anses for at være funktioner, der får et input og omdanner det til et output. Det er dog langt fra en simpel funktion, da neurale netværk kan have flere tusinder af variabler og parametre. Det første lag kaldes et inputlag (input layer), hvilket blot er der hvor funktionen indtager inputdata. Ud fra dette data vil der i sidste ende blive udgivet et output, hvilket gengives i det sidste outputlag (output layer). Alt imellem disse 2 lag kaldes i fagtermer for skjulte lag (the hidden layers). Dette er hvor modellen forsøger at kortlægge inputdaten til det korrekte/ønskede output. Det kunstige neurale netværk lærer at gøre dette gennem matematiske algoritmer og differentialregning, hvilket vil blive gennemgået i afsnit 3.

2.1 Forskellige former for Machine Learning

Der findes mange forskellige former for Machine Learning, og hver især har deres egen use case. Forskellene mellem dem afhænger i høj grad af, hvordan de behandler inputdata, og hvordan algoritmerne optimeres og lærer på baggrund af denne data. Kunstige neurale netværk er blot en type model der anvender Machine Learning.

I denne opgave vil der i høj grad være fokus på Supervised Learning[3], hvilket henviser til den specifikke måde, man træner en model på. Ved Supervised Learning oplyser alle datapunkter også det ønsket resultat som modellen skal give som output. Dette hjælper det kunstige neurale netværk at lære gennem matematiske beregninger. Man taler om labelled data. Ansigtsgenkendelse er et godt eksempel på anvendelsen af denne slags Machine Learning. For at træne et kunstigt neuralt netværk til at genkende ansigter, bliver det under træningsprocessen vist tusindvis af billeder. Hvert billede har dermed også en label, der fortæller modellen om billedet var et ansigt eller ej. Dette kan senere bruges til at forbedre det neurale netværks evne til at klassificere billeder af ansigter som ansigter. Supervised Learning bliver typisk anvendt til klassifikation af data eller regression, hvor modellen forsøger at finde en sammenhæng mellem de afhængige og uafhængige variabler. På denne måde kan modellen lave forudsigelser.

Der findes også Unsupervised Learning hvilket behandler unlabelled data. Dette anvendes ofte til at finde ligheder mellem data, og modeller kan dermed gruppere dataset. Et eksempel på dette kunne være når et selskab gerne vil kunne gruppere deres kunder ud fra deres tidligere opførsel.

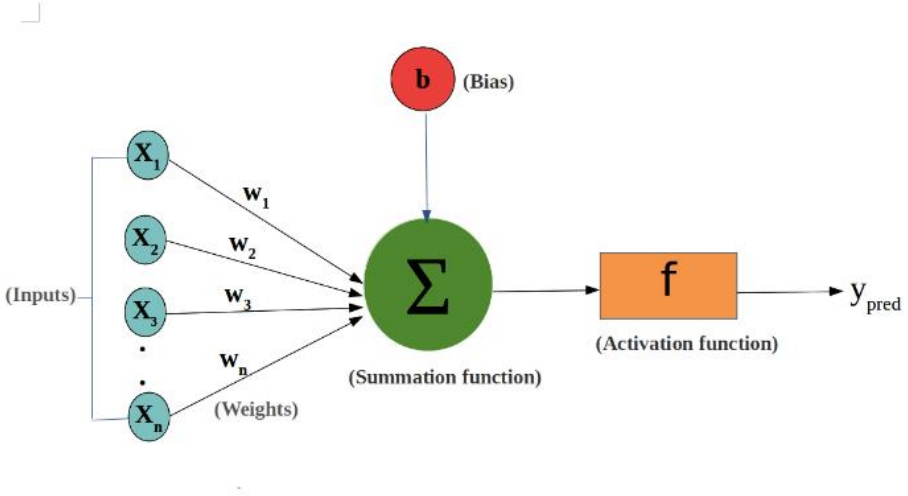
[3] (Education, Supervised Learning, 2020)

Reinforcemenmt Learning (RL)[4] er endnu en underkategori inden for Machine Learning. Ved brug af RL lærer en model ud fra tidligere oplevelser. Modellen bliver dermed straffet, når den udfører uhensigtsmæssige handlinger. Samtidig vil modellen blive opfordret til at udføre handlinger der giver den positiv feedback. Denne form for Machine Learning bliver blandt andet brugt til at udvikle selvkørende biler.

3. Redegørelse for matematikken bag kunstige neurale netværk:

Matematik er arbejdshesten bag Machine Learning og kunstige neurale netværk[5]. Som beskrevet i det tidligere afsnit, kan et kunstigt neuralt netværk beskrives som en meget kompleks funktion med et massevis af variabler og parametre. For at et kunstigt neuralt netværk kan lære at kortlægge inputdata til det korrekte output data, skal disse parametre justeres under træningsperioden. Dette sker gennem differentialregning.

For at kunne forstå matematikken og algoritmerne bag kunstige neurale netværk, er det vigtigt at kunne beskrive de matematiske forhold mellem neuronerne og lagene. Et kunstigt neuralt netværk er opbygget af neuroner, der hver især har deres egen aktiveringsværdi. Denne aktiveringsværdi kan ses som et tal mellem 0 og 1. Et neuron kan dermed være "slukket" eller "tændt".



Figur 2, Et flowdiagram, der beskriver beregningerne for et neurons aktiveringsværdi[6]

[4] (Reinforcement learning, 2021)
[5] Afsnittet Redegørelse for matematikken bag kunstige neurale netværk er skrevet ud fra (Sanderson, But what is a neural network? | Chapter 1, Deep learning, 2017)
[6] Kilde: https://miro.medium.com/max/1400/1*bV-zR_GgCeSvB1aqdxxehA.png

Figur 2 viser et flowdiagram for et neurons aktiveringsværdi. Som beskrevet er der en forbindelse mellem hvert neuron i et lag med hvert neuron i det næste. Hver af disse forbindelser bliver tildelt en vægt w, der under træningsprocessen kan blive justeret for at opnå det ønskede resultat. For at beregne et specifikt neurons aktiveringsværdi findes summen af alle de vægtede aktiveringsværdier fra det tidligere lag. Dette kan ses illustreret i figur 2, og den tilsvarende matematiske notation kan skrives således:

$$a_n = a_1 * w_1 + a_2 * w_2 + a_3 * w_3 \ ....$$

Denne aktiveringsværdi a_n kan være hvilket som helts reelt tal, og for at skalere værdien ned til et interval mellem 0 og 1 anvendes en aktiveringsfunktion. Her anvendes ofte logistiske funktioner såsom Sigmoidfunktionen:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Denne funktion kortlægger alle negative og positive input til et interval mellem 0 og 1. Meget negative summer vil dermed ligge tæt på nul, og omvendt vil store positive summer ligge tæt på 1. Begrundelsen for anvendelsen af netop denne funktion, stammer dog også fra bekvemmeligheden af dens afledte funktion, hvilket senere vil blive anvendt.

$$\sigma\ '(x) = \sigma(x) * (1 - \sigma(x))$$

Det faktum, at sigmoidfunktionens afledte funktion kan blive udtrykt i sig selv, kan give nogle fordele under beregningsprocessen i computeren, da tidligere beregnet funktionsværdier kan blive genbrugt. Der findes mange andre forskellige typer af aktiveringsfunktioner med deres egne fordele og ulemper. Dette vil blive diskuteret senere i afsnit 7. For nu at beregne neuronets aktiveringsværdi indsættes den beregnede vægtede sum i sigmoidfunktionen. Det kan være en fordel at tilføje det, som kaldes en bias b. Bias er egentlig bare er en numerisk værdi, der kan forhindre neuronet i at være tændt, hvis den vægtede sum ikke overgår et bestemt tal. Dermed kan aktiveringsværdien for et specifikt neuron udtrykkes på følgende måde:

$$a_n = \sigma(a_1 * w_1 + a_2 * w_2 + a_3 * w_3 \ldots + b)$$

Dette er blot beregningerne for et neuron i det kunstige neurale netværk, hvor flere parametre allerede indgår. Denne aktiveringsværdi skal i realiteten beregnes for alle neuroner i netværket. For at gøre dette anvendes matrixregning.

3.1 Matrixregning:

De ovenstående beregninger for et neurons aktiveringsværdi kan opstilles på en mere kompakt måde, hvilket også gør det nemmere for computeren at arbejde med. Her anvendes matricer, hvilket er skemaer bestående af rækker og kolonner. Antallet af rækker m og antallet af kolonner n danner tilsammen en matrix med dimensionerne m*n.

$$M_1 = \begin{matrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{matrix}, \quad n = 2 \; og \; m = 2$$

Matricer er en samling af elementer, der hver især kan refereres til ved brug af indekserne *i* og *j*. Indekset *i* betegner hvilken række som elementet befinder sig i. Tilsvarende betegner indekset *j* hvilken kolonne elementet befinder sig i. Det er derfor muligt at referere til et specifikt element i en matrix således: $element = a_{ij}$. Som det kan ses i matrixen ovenfor, starter indekserne *i og j* ved 0. I et kunstigt neuralt netværk placeres alle aktiveringsværdierne fra det foregående lag i en kolonnevektor eller en m*1 matrix a, hvor m betegner antallet af neuroner i laget. Dernæst placeres alle vægtene mellem neuronerne i det pågældende og foregående lag også i en matrix W. I denne matrix er alle vægtene i hver række forbundet med den tilsvarende række (altså en aktiveringsværdi) i aktiveringsvektoren. Til sidst opstilles alle bias for hvert neuron også i en m*1 matrix. Den følgende notation følger det der bliver beskrevet i denne kilde[7]

$$W = \begin{matrix} w_{0.0} & \cdots & w_{0.n} \\ \vdots & \ddots & \vdots \\ w_{k.0} & \dots & w_{k.n} \end{matrix}$$

$$a^0 = \begin{matrix} a_0^{(0)} \\ \vdots \\ a_n^{(0)} \end{matrix}$$

$$b = \begin{matrix} b_0^{(0)} \\ \vdots \\ b_n^{(0)} \end{matrix}$$

[7] (Sanderson, But what is a neural network? | Chapter 1, Deep learning, 2017)

Afsnittet Matrixregning er skrevet ud fra (Ebbesen, u.d.) side 2-8

Beregningen for alle aktiveringsværdier i et givent lag kan dermed beskrives som følgende, hvor a^1 betegner den matrix der indeholder alle aktiveringsværdierne for det 2. lag (der anvendes nul indeksering):

$$a^{(1)} = \sigma(W \times a^{(0)} + b)$$

Den vægtede sum for hvert neuron kan dermed beregnes ved at finde produktet mellem matrix W og kolonnematrixen a, hvorefter kolonnematrixen b adderes. I følgende del vil matrixmultiplikation samt addition blive gennemgået[8]

For at 2 matricer kan multipliceres, skal de være kompatible i deres dimensioner. En matrix består af m rækker og n kolonner. Dermed kan størrelsen på en matrix beskrives som m*n. For at 2 matricer kan multipliceres skal følgende gælde:

$$M_1 = \ m_1 * n_1 \quad , \quad M_2 = \ m_2 * n_2$$

$$m_1 = \ n_2$$

Hvis ovenstående gælder for M_1 og M_2, vil produktet mellem dem give en matrix M_3 med dimensionerne $m_1 * n_2$. Dette vil altid være tilfældet i beregningerne, der indgår i et kunstigt neuralt netværk. Matrixmultiplikation minder meget om prikprodukter mellem 2 vektorer. Betragt 2 kompatible matricer M_1 og M_2, der begge har dimensionerne 3*3.

$$M_1 = \begin{matrix} 1 & 2 & 3 \\ 5 & 3 & -1 \\ 2 & 0 & 1 \end{matrix} \quad , \quad M_2 = \begin{matrix} 4 & 0 & 1 \\ 2 & -2 & 4 \\ 1 & -3 & 5 \end{matrix}$$

$$M_3 = \ M_1 \times \ M_2$$

For at beregne produktet mellem de 2 matricer $M_1 \times M_2$, findes prikproduktet mellem den første rækkevektor i M_1 og den første kolonnevektor i M_2. Dette bliver det første element med indekset 00*(i = 0, j=0)* i den nye matrix. Dernæst findes elementet med indekset 01*(i = 0, j=1)* ved at finde prikproduktet mellem den første rækkevektor og den anden kolonnevektor i M_2. Dette gentages for den tredje kolonnevektor. Denne proces gentages dernæst med den anden og tredje rækkevektor i M_1. Produktet mellem de 2 matricer vil dermed være:

[8] (Ebbesen, u.d.) side 2-8

$$M_3 = \begin{matrix} 1*4+2*2+3*1 & 1*0+2*(-2)+3*(-3) & 1*1+2*4+3*5 \\ 5*4+3*2+(-1)*1 & 5*0+3*(-2)+(-1)*(-3) & 5*1+3*4+(-1)*5 \\ 2*4+0*2+1*1 & 2*0+0*(-2)+1*(-3) & 2*1+0*4+1*5 \end{matrix}$$

$$M_3 = \begin{matrix} 11 & -13 & 24 \\ 25 & -3 & 12 \\ 9 & -3 & 7 \end{matrix}$$

Det er værd at bemærke, at matrixmultiplikation ikke er en kommutativ matematiks operation. $M_1 \times M_2$ vil dermed ikke give det samme resultat som $M_2 \times M_1$. En mere generel illustration af matrixmultiplikation kan ses i *Figur 3.*

$$\begin{matrix} & \vec{b_1} & \vec{b_2} \\ & \downarrow & \downarrow \\ \vec{a_1}\rightarrow & 1 & 7 \\ \vec{a_2}\rightarrow & 2 & 4 \end{matrix} \cdot \begin{bmatrix} 3 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} \vec{a_1}\cdot\vec{b_1} & \vec{a_1}\cdot\vec{b_2} \\ \vec{a_2}\cdot\vec{b_1} & \vec{a_2}\cdot\vec{b_2} \end{bmatrix}$$

$A \qquad B \qquad C$

Figur 3, denne figur illustrerer matrixmultiplikation[9]

Addition eller subtraktion mellem 2 matricer gøres ved at udføre den matematiske operation mellem hvert element i den ene matrix med det tilsvarende element i den anden matrix. Addition mellem de 2 matricer $M_1 og M_2$ vil dermed se således ud:

$$M_1 + M_2 = \begin{matrix} 1 & 2 & 3 \\ 5 & 3 & -1 \\ 2 & 0 & 1 \end{matrix} + \begin{matrix} 4 & 0 & 1 \\ 2 & -2 & 4 \\ 1 & -3 & 5 \end{matrix} = \begin{matrix} 5 & 2 & 4 \\ 7 & 1 & 3 \\ 3 & -3 & 6 \end{matrix}$$

En mere generel udledning vil se således ud:

$$M_1 = \begin{matrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{matrix}, \quad M_2 = \begin{matrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{matrix}$$

$$M_1 + M_2 = \begin{matrix} a_{00}+b_{00} & a_{01}+b_{01} \\ a_{10}+b_{10} & a_{11}+b_{11} \end{matrix}$$

I formel for aktiveringsmatrixen findes produktet mellem aktiveringsmatrixen a og vægtmatrixen W. Når produktet mellem de 2 matricer beregnes, svarer det til at finde den vægtede sum for alle neuronernes aktiveringsværdier. Herefter adderes lagets bias til summen. Da vægtmatrixen W har

[9] Kilde: Figuren er fundet på denne hjemmeside Multiplying matrices (article) | Matrices | Khan Academy

dimensionerne k*n, hvor k betegner antallet af neuroner i laget L, og aktiveringsmatrixen a^{L-1} har dimensionerne n*1, vil produktet mellem de to være en matrix a^{L} med dimensionerne k*1. Dette er dermed også en vektor. [10]

$$a^{(L)} = \sigma(W \times a^{(L-1)} + b^{L})$$

Når en matrix er et input i en funktion, svarer det til at hvert element i matrixen bliver input i funktionen

$$\sigma\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sigma(x) \\ \sigma(y) \\ \sigma(z) \end{pmatrix}$$

Det er nu muligt at beskrive alle neuronernes aktiveringsværdier i et givent lag L ved at indsætte produktet mellem vægtmatrixen og aktiveringsvektoren i laget L-1 i en aktiveringsfunktion. Resultatet bliver igen en aktiveringsvektor, der kan anvendes til at beregne aktiveringsværdierne for neuronerne i det næste lag.

3.2 Backpropagation algoritmen:

Som beskrevet tidligere (afsnit 2) er et kunstigt neuralt netværk en funktion. For at denne funktion kan lære at kortlægge det givne input til det korrekte output, skal de mange parametre i form af vægtene mellem neuronerne samt deres tilhørende bias optimeres og justeres. Dette sker gennem det, som kaldes backpropagation algoritmen. En algoritme er bare en række af matematiske operationer, som følges, hvilket dermed også kan udføres af en computer. Ved brug af Supervised Learning (afsnit 2.1) kan man ud fra de angivne labels estimere hvor dårligt/godt en model præsterer. Til dette anvendes en omkostningsfunktion. I lighed med aktiveringsfunktionerne, som beskrevet i (afsnit 3), findes der forskellige typer af omkostningsfunktionen. I dette afsnit vil "The mean squared error" blive anvendt[11].

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

I denne funktion betegner y_i det ith rigtige output, hvilket er det ønskede output for funktionen. $\hat{y}_i$ er det estimerede output for funktionen. Begge disse værdier beskrives ofte som vektorer, da et

[10] (Sanderson, But what is a neural network? | Chapter 1, Deep learning, 2017)
[11] (Barta, 2022)

neuralt netværk kan have mange output. I "The mean squared error" finder man kvadratet af forskellen mellem disse 2 værdier. På denne måde vil en stor differens give en endnu højere værdi, hvilket vil motivere netværket til at minimere fejlen. Når man træner et neuralt netværk, gøres det ofte ved, at man viser modellen en gruppe af data også kaldet et parti(batch). For hvert datapunkt i partiet beregnes den kvadratiske forskel, hvorefter alle forskellene bliver summeret. Den gennemsnitlige fejl findes dermed ved at dividere summen med antallet af datapunkter n. Herefter kan netværkets parametre opdateres for at minimere denne fejl. Denne proces gentages iterativt hvormed parametrene gradvist optimeres til at formindske omkostningsfunktionen.

3.2.1 Gradient Decent

For at optimere omkostningsfunktionen anvendes gradienter. Gradienten er en vektor, der beskriver den retning hvormed en funktion vokser mest. For at minimere en funktion anvendes gradienten dermed med omvendt fortegn. For en funktion med flere variabler beregnes den omvendte gradient ved at finde de partielt afledte funktioner for hver variabel. I tilfældet af et kunstigt neuralt netværk har omkostningsfunktionen C typisk flere tusinde variabler i form af netværkets parametre[12]

$$-\nabla C(x_0, x_1, \dots) = -\begin{matrix} \frac{C}{\partial x_0} \\ \frac{C}{\partial x_1} \\ \vdots \end{matrix}$$

Den beregnede vektor fortæller dermed, hvordan hver parameter skal ændres for at minimere funktionen. Hvis dette gentages nok gange, vil parametrene efterhånden konvertere til et lokalt minimum for funktionen. Følgende vektor kunne være et eksempel på en beregnet gradient efter en træningscyklus.

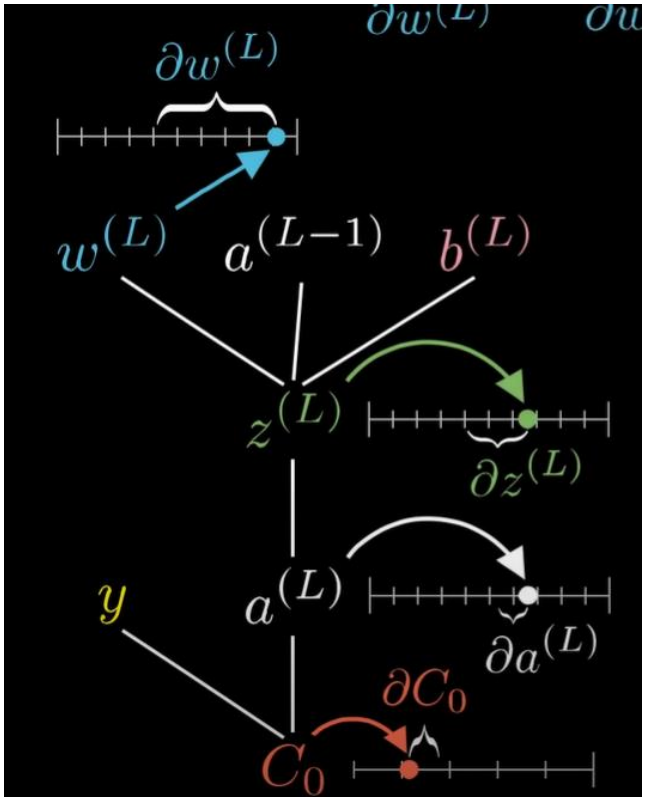
$$-\nabla C = \begin{matrix} -0.4 \\ 3.1 \\ 2 \end{matrix}$$

Denne vektor beskriver dermed, hvordan de 3 parametre i funktionen bør ændres for at omkostningsfunktionen kan minimeres hurtigst. Fortegnet indikerer om parameterens værdi skal sænkes eller hæves. Vektorkomponentens størrelse beskriver, hvor meget denne parameter skal ændres,

[12] (Sanderson, Gradient descent, how neural networks learn | Chapter 2, Deep learning, 2017)

men kan også indikere hvor stor en betydning netop denne parameter har for funktionen. For at beregne gradienten for omkostningsfunktionen anvendes backpropagation algoritmen.

Omkostningsfunktionen er reelt sammensat af flere funktioner, og for at beregne den afledte funktion for omkostningsfunktionen C i forhold til en bestemt parameter, anvendes kædereglen og partiel differentiering.



Figur 4, sammensætningen af omkostningsfunktionen[13]

I figur 4 kan det ses, hvordan omkostningsfunktionen er sat sammen af flere dele. Omkostningen for et specifikt datapunkt beregnes på følgende måde[14]:

$$C_0 = \sum_{j=0}^{n_L-1} (y_j - a_j^{(L)})^2$$

C_0 har følgende partielt afledte funktion i forhold til $a_j^{(L)}$. Denne afledte funktion vil blive anvendt i senere beregninger:

[13] Billede taget fra videon (Sanderson, Backpropagation calculus | Chapter 4, Deep learning, 2017)
[14] (Sanderson, Backpropagation calculus | Chapter 4, Deep learning, 2017)

$$\frac{\partial C_0}{\partial a_j^{(L)}} = 2\left(y_j - a_j^{(L)}\right)$$

Her beskriver y den ønskede værdi for funktionens output og $a_j^{(L)}$ beskriver funktionens estimerede output. *L* indikerer hvilket lag neuronet befinder sig i, og *j* indikerer den nøjagtige neuron i laget *L*. Omkostningen er summen af alle de kvadratiske forskelle. Selve aktiveringsværdien $a_j^{(L)}$ bliver beregnet ud fra $z_j^{(L)}$, hvilket er den vægtede sum af alle de tidligere aktiveringsværdier, som beskrevet i afsnittet om matrixmultiplikation (afsnit 3.1). Bogstavet k indikerer den specifikke neuron i laget L-1. Dermed bliver vægten mellem neuronet $a_k^{(L-1)}$ og $a_j^{(L)}$ refereret til som $w_{jk}^{(L)}$. $z_j^{(L)}$ er den vægtede sum af alle de tidligere aktiveringsværdier i laget L-1, hvorefter en bias bliver tilføjet[15]:

$$z_j^{(L)} = + b_j^{(L)} + \sum_{i=0}^{k} w_{ji}^{(L)} * a_i^{(L-1)}$$

Aktiveringsværdien for et neuron bliver dermed aktiveringsfunktionen med $z_j^{(L)}$ som input:

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

Målet med backpropagation er at beregne de afledte funktioner for omkostningsfunktionen i forhold til alle parametrene i det neurale netværk.

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

Ovenfor kan det ses, hvordan kædereglen anvendes til at beregne den afledte funktion for omkostningsfunktionen i forhold til en specifik vægt $w_{jk}^{(L)}$. Omkostningsfunktionen afhænger af aktiveringsværdien. Aktiveringsværdien afhænger af værdien for z, som afhænger direkte af de parametre, som skal optimeres. Forholdet mellem funktionerne og parametrene kan ses i figur 4. Den afledte funktion for omkostningsfunktionen i forhold til en specifik vægt kan beregnes på følgende måde:

[15] (Nielsen, 2019)
Afsnittet Gradient Decent er skrevet ud fra (Sanderson, Backpropagation calculus | Chapter 4, Deep learning, 2017)
Afsnitter Gradient Decent er også inspireret af (Nielsen, 2019)

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = 2\left(y_j - a_j^{(L)}\right) * \sigma'\left(z_j^{(L)}\right) * a_k^{(L-1)}$$

Når udtrykket for C_0 differentieres i forhold til aktiveringsværdien, differentieres den ydre funktion først, og den indre funktion differentieret ganges på. Da den indre funktion $a_j^{(L)}$ i sig selv også er en sammensat funktion anvendes kædereglen igen. Dermed fås ovenstående udtryk. Denne formel beskriver dermed hvordan omkostningen C_0 ændrer sig når vægten $w_{jk}^{(L)}$ ændres. Dette er dermed blot en komponent i gradienten. Samme beregninger kan udføres for bias og aktiveringsværdierne.

$$\frac{\partial C_0}{\partial b_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

Ud fra den beskrevne aktiveringsfunktion (se afsnit 3) og omkostningsfunktion kan dette udtryk skrives mere specifikt.

$$\frac{\partial C_0}{\partial b_j^{(L)}} = 2\left(y_j - a_j^{(L)}\right) * \sigma'\left(z_j^{(L)}\right) * 1$$

Ovenstående beskriver hvordan omkostningen C_0 ændrer sig, når bias $b_j^{(L)}$ ændres. For at beregne den afledte funktion for C_0 i forhold til en specifik aktiveringsværdi $\partial a_k^{(L-1)}$, er det nødvendigt at tage højde for, at denne aktiveringsværdi påvirker alle aktiveringsværdierne i laget L. Dette skyldes, at aktiveringsværdien i laget L bliver beregnet ved at finde den vægtede sum af alle de tidligere aktiveringsværdier i L-1.

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L - 1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial z_j^{(L)}} \frac{\partial C_0}{a_j^{(L)}}$$

For at finde den afledte funktion i forhold til $a_k^{(L-1)}$ summeres alle de afledte funktioner for aktiveringsværdierne i laget L. Dette udtryk kan igen skrives mere specifikt som:

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L - 1} 2\left(y_j - a_j^{(L)}\right) * \sigma'\left(z_j^{(L)}\right) * w_{jk}^{(L)}$$

Selvom det ikke er muligt direkte at ændre på aktiveringsværdierne i det tidligere lag L-1, kan hele algoritmen udføres igen rekursivt, hvor værdien for $\frac{\partial C_0}{\partial a_k^{(L-1)}}$ bliver anvendt som $\frac{\partial C_0}{\partial a_k^{(L)}}$ i beregningerne for det næste lag.

På denne måde kan de afledte funktioner i forhold til hver enkelt variable (alle parametrene i det neurale netværk) i omkostningsfunktionen beregnes. Hvis værdier for parametrene indsættes, fås retningen for gradienten $\Delta C(\dots)$, der med omvendt fortegn beskriver, hvordan parametrene skal ændres for at minimere omkostningsfunktionen.

3.2.2 Læringsrate og optimering

For at minimere risikoen for at overskride det lokale minimum for omkostningsfunktionen skaleres gradienten ned med en parameter kaldet læringsraten. Læringsraten forsøger at mindske størrelsen af de "skridt" som gradienten angiver vil minimere omkostningen mest muligt[16]. Læringsraten bliver angivet med symbolet α.

$$step = \alpha * (-\Delta C)$$

Læringsraten bliver ofte sat relativt lavt, men den konkrete værdi kan blive eksperimenteret med i modellen.

Selve beregningerne bag gradienten for omkostningsfunktionen kan være meget tidskrævende. Derfor kan man indføre nogle teknikker for at minimere antallet af beregningerne i backpropagation algoritmen. Det kan være meget effektivt at opdele dataene i mindre grupper også kaldt batches[17]. På denne måde kan den samlede gradienten beregnes for alle datapunkter i en batch, men netværkets parametre bliver først opdateret på baggrund af den gennemsnitlige gradient for en hel batch. Dette kan minimere tidskompleksiteten for læringsprocessen drastisk, dog vil hvert trin være mindre effektivt. Ideelt burde gradienten beregnes efter hvert datapunkt, men denne tilgang har ofte en alt for høj tidskompleksitet i praksis. Derfor anvendes mini-batch gradientnedstigning som beskrevet. Andre tekniker såsom stokastisk gradientnedstigning[18] kan også anvendes til at minimere tidkompleksiteten. Kort beskrevet indfører denne teknik noget tilfældighed i processen, hvormed datapunkter udvælges tilfældigt, og gradienten beregnes ud fra dette.

[16] (Srinivasan, 2019)
[17] (Brownlee, 2018)
[18] (Srinivasan, 2019)

4. Metoden bag konstruktionen af en ML model

Konstruktionen af en Machine Learning model kan opdeles i 7 trin[19]. Denne trinbaseret opdeling af en Machine Learning model vil blive anvendt i næste afsnit til at programmere et kunstigt neuralt netværk fra bunden.

1. Opsamling af data

   Generelt er data yderst vigtig i Machine Learning. Det er dermed nødvendigt at have en stor nok mængde af repræsentativt og velfordelt data. Kvaliteten af den indsamlet data er i direkte korrelation med modellens anvendelighed og præcision.

2. Forberedelse af data

   For at den indsamlede data kan anvendes, skal det ofte forbedredes således, at modellen kan anvende det. Først skal dataene gennemgås, og uønskede datapunkter fjernes eller modificeres hvis nødvendigt. Ofte er det nødvendigt at omdanne dataene til andre datatyper, da modellen kun kan anvende tal. Der findes blandt andet mange metoder til at repræsentere tekst eller billeder som tal. Typisk bør dataene også standardiseres for at sikre, at de forskellige datapunkter bliver vægtet lige højt. Herefter splittes datene i to forskellige datasæt. Træningssættet vil blive anvendt under træningsprocessen, hvorefter modellens præcision kan bestemmes ud fra testsættet. Det er meget vigtigt, at en model ikke bliver præsenteret for testsættet under træningen. Dette skyldes at modellen muligvis kan lære at memorere dataene i stedet for at finde mønstre og sammenhænge. Dette trin vil blive diskuteret mere i dybden i et senere afsnit 7

3. Udvælge en model

   Når dataene er blevet indsamlet og forberedt, skal en model kontureres. Dette kan gøres på mange måder, dog vil der i denne opgave være fokus på de kunstige neurale netværk. Dertil skal aktiveringsfunktioner, omkostningsfunktionen og læringsraten udvælges, alt afhængig af problemet. Selve strukturen af netværket er også vigtigt. Antallet af lag samt mængden af neuroner skal udvælges, og dette gøres ofte i en iterativ proces. Modeller bliver derfor ofte rekonstrueret flere gange for at finde den bedste løsning.

[19] (The Complete Guide to Machine Learning Steps, 2021)

4. Træning af modellen

   Træningsprocessen er hvor modellen forsøger at konverterer mod et minimum for omkostningsfunktionen, som beskrevet i afsnit 3.2.1. Dette gøres ved at præsentere modellen for træningssættet gennem flere iterationer også kaldet epochs[20]. Træningsprocessen kan visualiseres ved at plotte værdierne for omkostningsfunktionen over iterationerne. Dette kan give et indblik i hvor hurtigt omkostningsfunktionen konverterer mod et lokalt minimum. Det kan også ske at modellen ikke konverteret, hvilket også vil kunne ses på grafen.

5. Evaluer modellen

   Efter træningsprocessen udføres en evaluering af modellen. Dette gøres ved at anvende det hidtil uanvendte testsæt, til at vurdere hvor præcis modellen er. Det er vigtigt at evalueringen sker på baggrund af uset data og ikke træningssættet, som beskrevet i step 2. For at vurdere modellens præcision kan forskellige målinger anvendes. Under evalueringen analyseres omkostningsfunktionens værdi, og andre målinger såsom nøjagtigheden bliver beregnet.
   I tilfældet af en klassifikationsmodel kan nøjagtigheden beregnes på følgende måde[21]:

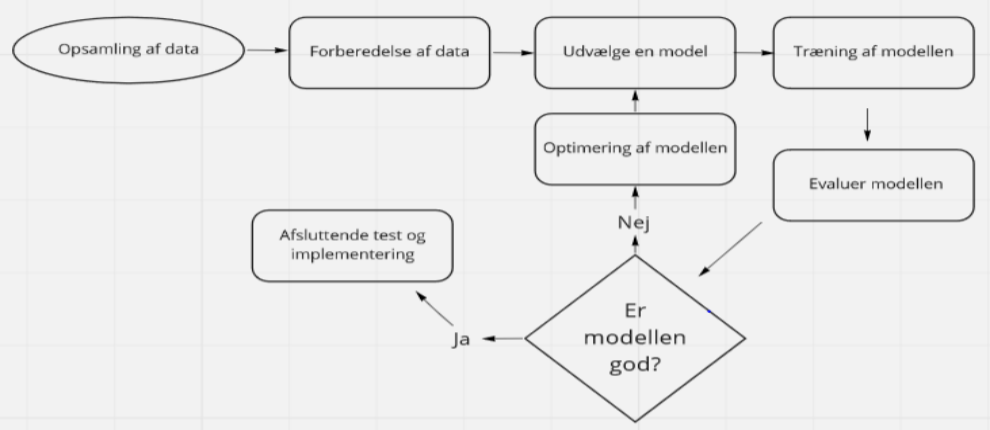
$$Accuracy = \frac{True\ Positive + True\ Negativ}{True\ Positive + True\ negative + True\ Positive + \ False\ Positive}$$

   Ud fra disse målinger vurderes det om modellen er præcis og god nok til det ønskede formål.

6. Optimering af modellen

   Hvis modellen evalueres til ikke at være præcis nok, skal strukturen af modellen genovervejes og ændres. Step 3 til 5 bliver dermed gentaget, og modellens parametrene bliver optimeret. Denne iterative proces kan blive illustreret som set i nedenstående flowdiagram (figur 5)

[20] (Brownlee, 2018)
[21] (Amor, 2020)



Figur 5, Flowdiagram der beskriver de 7 trin samt den iterative proces

7. Afsluttende test og implementering

   Hvis modellen evalueres til at være præcis nok i step 5, udføres en afsluttende test på data, der ikke er blevet anvendt under træningsprocessen. Dette er med til at give et afsluttende tjek på om modellen fungerer. Herefter kan modellen implementeres og anvendes.

5. Konstruktion af et kunstigt neuralt netværk

Koden for det selvvalgte projekt kan findes i den vedhæftede zip-fil samt på følgende link: kode

I dette afsnit vil de gennemgået formler og algoritmer blive anvendt i praksis. Selve modellen, som der vil blive kontureret, er tænk som et simpelt indblik i kunstige neurale netværk og Supervised Learning. Modellen vil blive programmeret i programmeringssproget Python helt fra bunden, og blot nogle få biblioteker (ekstern kode som kan importeres i et program) vil blive anvendt. Selve koden er inspireret af denne guide (Aflak, 2015). Den anvendte data kommer fra et velkendt datasæt kaldt fashion-MNIST. Datasættet består af 70000 labelled sort-hvid pixelbilleder af 10 forskellige typer tøj. Det kunstige neurale netværk vil blive trænet ved brugen af backpropagation algoritmen, som beskrevet i afsnit 3, til at kunne klassificere disse billeder. Selve algoritmen i programmet vil dog ikke beregne den optimale ændring for hver vægt og bias enkeltvist, men som en samlet vektor. For at udføre de matematiske beregninger såsom matrixmultiplikation (afsnit 3.1) effektivt, anvendes et bibliotek kaldt ”numpy”(NumPy). Numpy bliver ofte forkortet til ”np” i kode.

For at programmere modellen er Objekt orienteret programmering (OOP) blevet anvendt[22]. OPP er et programmeringsparadigme, der er baseret på klasser og objekter. Disse objekter kan have

[22] (Gillis, u.d.)

attributter (data) samt metoder i form af funktioner. I dette projekt er der blevet kreeret 4 forskellige klasser, der gennem deres metoder og attributter kan interagere med hinanden. Fælles for alle klasserne der repræsenterer de forskellige lag i det neurale netværk er deres metoder. De har alle en metode kaldt ”forward”, der beregner en vektor for alle lagets aktiveringsværdier

$$a^{(L)} = \sigma(W \times a^{(L-1)} + b^L)$$

Denne formel bliver beskrevet i afsnit 3.1. Den tilsvarende kode for et givent lag kan ses i figur 5.

```
def forward(self,input):
    self.input = input

    self.values = np.dot(input, self.weights) + self.bias
    return sigmoid(self.values)
```

Figur 6, kode fra produktet under filen HiddenLayer.py

I filen HiddenLayer.py defineres en metode i ”HiddenLayer” klassen, der beregner lagets aktiveringsværdier. I denne funktion angives en attribut ”self.input”, hvilket er en vektor/array med alle aktiveringsværdierne fra det tidligere lag. Dette anvendes til at beregne den vægtede sum:

$$z^L = W \times a^{(L-1)} + b^L,$$

Matrixen W angives som ”self.weights” i koden. For at beregne produktet mellem matrixen W og aktiveringsvektoren $a^{(L-1)}$, anvendes den importerede funktionen ”np.dot()” til at udføre matrixmultiplikation som beskrevet i afsnit 3.1. Herefter indsættes den beregnede vektor i sigmoidfunktionen. Denne funktion returnerer dermed aktiveringsvektoren for dette lag. De andre klasser har tilsvarende metoder til at beregne aktiveringsvektoren.

For at beregne gradienten har hver af disse klasser også en metode kaldt ”backward”. Koden for denne funktion kan ses i figur 6.

```
def backward(self,output_error, learning_rate):
    output_error = output_error * sigmoid_prime(self.values)

    input_error = np.dot(output_error, self.weights.T)
    weights_error = np.dot(self.input.T, output_error)
    bias_error = output_error
    self.weights -= learning_rate * weights_error
    self.bias -= learning_rate * bias_error
    return input_error
```

Figur 7, kode fra produktet under filen HiddenLayer.py

I selve koden bliver gradienten for lagenes parametre beregnet som separate vektorer. Den fulde gradient for omkostningsfunktionen er dermed alle disse separate vektorer samlet. Denne funktion har 2 parametre ”output_error” og ”learning_rate”. Læringsratens værdi angives i filen Main.py som en konstant, hvorfra den bliver videregivet til alle objekterne.

For at simplificere den nedenstående forklaring tildeles følgende afledte funktioner navne. Dertil bliver de anvendte matematiske formel gennemgået i afsnit 3.2.1.

$$B0: \frac{\partial C_0}{\partial a_k^{(L-1)}} \quad B1: \frac{\partial C_0}{\partial a_j^{(L)}} \quad B2: \frac{\partial a_k^{(L-1)}}{\partial z_j^{(L)}} \quad B3: \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}}$$

”output_error” indikerer værdien for den afledte funktion B1. I koden vil dette blot være en komponent i vektorerne ”output_error”. B1 er en del af udregningen for B0 (se afsnit 3.2.1). For at beregne hele den afledte funktion ganges B2 og herefter B3 på. I koden (figur 6) svarer dette til omdefineringen af ”output_error”, hvor den tidligere vektor udfører komponentvis vektormultiplikation (Hadamard produktet)[23] med vektoren, der bliver returneret fra hjælpefunktionen ”sigmoid_prime” (B2 på vektorform). Implementeringen af denne funktion kan findes i filen functions.py og følger matematikken beskrevet i afsnit 3. Herefter udføres der i linje 3 (se figur 6) matrixmultiplikation for at beregne B0 i form af variablen ”input_error”. Denne værdi bliver dermed værdien for B1 i det næste lag som beskrevet i slutningen af afsnit 3.2.1. For at beregne den optimale ændring af ”self.weights” anvendes følgende udtryk:

[23] (Nielsen, 2019)

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = 2\left(y_j - a_j^{(L)}\right) * \sigma'\left(z_j^{(L)}\right) * a_k^{(L-1)}$$

I koden (figur 6, linje 4) beregnes $\frac{\partial C_0}{\partial w_{jk}^{(L)}}$ ved at finde produktet mellem aktiveringsværdierne fra det tidligere lag L-1 ”self.input” og matrixen ”output_error”. Herefter skaleres den beregnede matrix ”weights_error” ned med læringsraten som beskrevet i afsnit 3.2.2, og lagets attribut ”self.weights” opdateres. Tilsvarende gøres for vektoren ”self.bias”. Klassen for outputlaget har en tilsvarende metode til at beregne ændringerne til dette lags parametre. I dette lag anvendes en anden aktiveringsfunktion og dermed bliver udtrykket for B2 anderledes i koden. Her kunne sigmoidfunktionen også været blevet anvendt, og denne implementering kan ligeledes ses i koden.

Det er relevant at bemærke, at en prætrænet models parametre kan findes i undermappen ”Modelparametre”. Denne model kan anvendes ved at kalde ”Network” klassens metoder ”load_params” og ”setup_network”. Den prætrænet model vil blive analyseret i næste afsnit. Koden er også blevet dokumenteret i selve programmet, hvor de 7 trin (se afsnit 4) er blevet fremhævet og forklaret.

6. Analyse af modellen

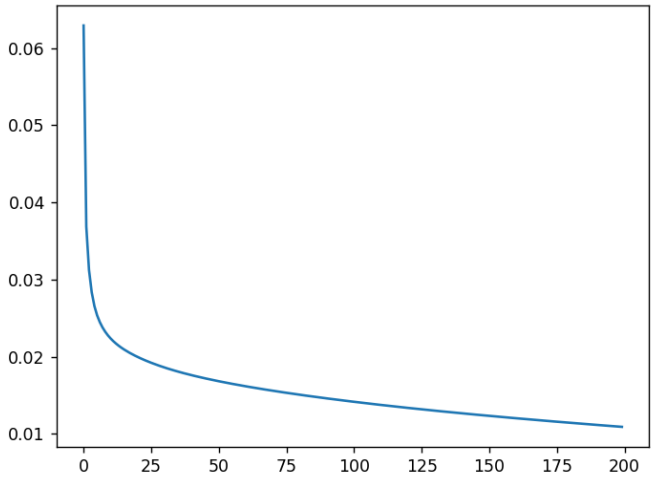
Den prætrænet model er kontureret gennem de 3 forskellige lag, med hver deres antal af neuroner. Det første lag er inputlaget, hvor netværket modtager data. Inden dette lag modtager datene, er den blevet forbedret på en specifik måde. Dette er trin 2, forberedelse af data, som beskrevet i afsnit 4. Selve det anvendte datasæt består af billeder, hvor hver pixel bliver repræsenteret som et element i en matrix. For at netværket kan anvende dette omdannes denne matrix til en vektor. Hver pixel i denne vektor bliver repræsenteret gennem et tal i form af gråskalaværdien. Gråskalaen går fra 0 til 255. Alle disse værdier bliver dernæst skaleret ned til et interval mellem 0 og 1. Dette gøres ved at dividere med den maksimale værdi på 255. Denne matematiske manipulation kan ses som en standardisering af dataene. Da billederne består af 28*28 pixels, vil antallet af komponenter i vektoren være 784. Der vil dermed altid være 784 neuroner i inputlaget. Modellen har også et skjult lag (hidden layer) med i alt 128 neuroner. Dertil er sigmoidfunktionen (se afsnit 3) blevet anvendt som dette lags aktiveringsfunktion. Antallet af neuroner og skjulte lag kunne potentielt ændres for at give en bedre model. Det skal dog bemærkes at hvis disse strukturparametre bliver for store vil træningsprocessen tage endnu længere tid, og modellen vil potentiel ikke blive bedre. Generelt forsøger man i Machine Learning at lave de simpleste modeller muligt, der stadig kan udføre den givne

opgave. Dette har også været tilgangen til udarbejdelsen af denne model. Den valgte aktiveringsfunktion er egentlig blevet valg på baggrund af dens simple implementering. Den er dog meget typisk anvendt i klassifikationsmodeller. Dette skyldes at sigmoidfunktionen kan konvertere tal til en sandsynlighed[24]. Modellen har 10 output neuroner, hvor hver deres aktiveringsværdi beskriver sandsynligheden for, at billedet repræsenterer et givent stykke tøj. Dermed bliver modellens forudsigelse den højeste af disse sandsynligheder. Sigmoidfunktionen tilføjer også noget ikke-linearitet til modellen, hvilket ofte er nødvendigt for at beskrive komplekse sammenhænge mellem data. En anden aktiveringsfunktion såsom ReLU er også meget almindeligt set anvendt i kunstige neurale netværk. ReLU er defineret som:

$$f(x) = \max(0, x)$$

ReLU er dermed identitetsfunktion for alle x-værdier over 0, og alle negative input har en funktionsværdi på 0. ReLU har nogle forskellige fordele overfor sigmoidfunktionen, men den største er at den er meget hurtig at beregne. Dertil har ReLU en konstant hældning, og dens afledte funktion er dermed meget hurtigere at beregne. For meget store og komplekse modeller der anvender sigmoidfunktionen som aktiveringsfunktion, kan træningsprocessen ofte være meget tidkrævende. Dette skyldes at gradienten meget hurtigt konverterer mod nul, og netværkets omkostningsfunktion dermed optimeres meget langsomt[25]. Da modellen anvendt i denne opgave ikke er specielt kompleks, har dette dog ikke været et problem. Man kunne med fordel forsøge at implementere den samme model blot med ReLU som aktiveringsfunktion i det skjulte lag. Herefter kunne de 2 modellers grafer for omkostningen sammenlignes (se trin 4, afsnit 4). Grafen for den prætrænet model kan ses herunder.

[24]Dette afsnits analyse er skrevet ud fra (Wood, u.d.)
[25] (Wood, u.d.)



Figur 8, graf af omkostningen

Det kan ses på figur 8, at omkostningen efter træningsprocessen sluttede på 0.011. På trods af den lave omkostning havde modellen endnu ikke konverteret imod et lokalt minimum. Når en model er konverteret, vil det ses som en lige linje på grafen. Dette kunne også ses under selve træningsprocessen, da omkostningen stadig var aftagende selv under de sidste iterationer gennem datasættet. Modellen blev trænet med hele fashion-MNIS datasættet gennem 200 iterationer(epochs) og opnåede under evalueringen (trin 5, afsnit 4) en nøjagtighed på 88.25% samt en gennemsnitlig omkostning på 0.02. Dette er et ganske udmærket resultat. Det ville være fordelagtigt fortsat at træne modellen indtil at den konverterer mod et lokalt minimum.

7. Diskussion af muligheder og begrænsninger for neurale netværk

Kunstige neurale netværk kan trænes og modelleres til at udføre mange forskellige opgaver på en effektiv måde. Supervised Learning (se afsnit 2) kan anvendes til at udarbejde simple klassifikationsmodeller som beskrevet i afsnit 5. Denne slags modeller kan udføre billede og objektgenkendelse, hvilket også anvendes til ansigtsgenkendelse dog med noget mere kompleksitet. Regressionsmodeller kan beskrive sammenhænge mellem uafhængige og afhængige variabler[26], hvilket hyppigt ses anvendt i store firmaer ud fra kunders data. Neurale netværk er blot en type model inden for

[26] (Education, Supervised Learning, 2020)

Afsnittet om Diskussion af muligheder og begrænsninger er inspireret af (7 Major Challenges Faced By Machine Learning Professionals, 2021)

Machine Learning. Machine Learning algoritmer er kraftigt afhængige af data, hvilket til tider kan give nogle problematikker. Opsamling af data er det første trin for at udarbejde en ML model, og store mængder af højkvalitets data er ofte nødvendige, for at modeller kan trænes og senere anvendes i praksis. For små mængder af data vil resultere i tendentiøse forudsigelser, da modellen ikke er blevet præsenteret for et bredt nok spektrum af data. Dertil vil lavkvalitetsdata, såsom datapunkter med forkerte labels også give dårlige resultater. Denne problematik bekræfter også vigtigheden bag forberedelsen af et datasæt som forklaret i afsnit 4.

I modellen udarbejdet i denne opgave er et velkendt datasæt blevet anvendt (afsnit 5). Under konstruktionen af et kunstigt neuralt netværk er det dog ikke altid nemt at komme i besiddelse af et præeksisterende og relevant datasæt. Dertil er det i Supervised Learning en nødvendighed at datasættet er labelled. Dette er i sig selv ofte en stor udfordring og tidskrævende proces. For at bearbejde denne udfordring er forskere rundt omkring i verden gået sammen for at tilbyde ordinære mennesker penge for at give disse datasæt labels. Ideelt er dette en god løsning, dog kan det resultere i menneskelige fejl og dermed sænke kvaliteten af et datasæt. Generelt er opsamlingen og forberedelsen af data yderst vigtige dele af kunstige neurale netværk. Ofte kræver det også ekstra arbejde for at konvertere den opsamlet data til anvendelige datatyper for en model.

	Country	Age	Salary	Purchased
0	France	44.0	72000.000000	No
1	Spain	27.0	48000.000000	Yes
2	Germany	30.0	54000.000000	No
3	Spain	38.0	61000.000000	No
4	Germany	40.0	63777.777778	Yes

Figur 9, en tabel med forskellige datatyper[27]

I figur 9 ses en model med forskellige datatyper, der kunne findes i et datasæt. Disse datapunkter skal hver især standardiseres og muligvis konverteres til andre datatyper. Dette er et helt separat felt inden for Machine Learning. Datapunkter såsom ”Purchased” kan nemt repræsenteres binært som 1(yes) eller 0(nej). Det er dog mere besværligt at præsentere tekst såsom ”France” på talform. Dette er dog nødvendigt for at træne kunstige neurale netværk til at udføre opgaver såsom at genkende tekst eller endda skrive tekststykker selv. For at forbedrede denne slags data anvendes algoritmer

[27] Kilde: https://miro.medium.com/max/592/1*Bqb-yfE-VNLhBQTee0Xa3g.png

såsom One-Hot Encoding[28], hvor tekst kan repræsenteres binært. Denne tilgang medfører dog dets egne udfordringer. Samme udfordringer kan opstå når datasættet indeholder billeder. For helt simple sort-hvid pixelbilleder er forberedelsen af daten ikke kompliceret, da det ofte er tilstrækkeligt at anvende hver pixels gråskalaværdi som beskrevet i afsnit 6. Det er dog nødvendigt at anvende andre teknikker for at behandle mere komplekse billeder. Modellen skal i disse scenarier også kunne forstå sammenhængen mellem de forskellige pixels i billedet, hvilket ikke er muligt ved blot at anvende gråskalværdien. Inden for kunstige neurale netværk bliver dette problem ofte løst med convelutional neurale netværk[29], der kort sagt implementerer nogle filtre på billederne.

7.2 Begrænsninger ved træningsprocessen:

Træningsprocessen for et kunstig neuralt netværk kan også være en stor udfordring. Som beskrevet i afsnit 4 deles et datasæt typisk i 2 sæt. Et træningssæt der bliver anvendt under træningsprocessen til at justere netværket parametre, og et testsæt der avendes under evalueringen af modellen. Under træningsprocessen kan der opstå det, som kaldes overfitting. Overfitting betyder at modellen tilpasser sig træningssættet for meget. Modellen formår simpelthen at memorere datene i stedet for at finde generelle sammenhænge og mønstre som ønsket. Dette er et meget normalt problem i Machine Learning, og der findes teknikker såsom dataaugmentation, hvilket er en gren inden for dataanalyse, der anvendes til at forbygge overfitting. Overfitting kan ofte opstå ved for små datasæt. Tegn på overfitting kan blandt andet ses under evalueringsprocessen. Hvis den gennemsnitlige omkostning for testsættet er for høj i forhold til træningssættet, er dette et tegn på overfitting. En forskel kan forventes, men den bør ikke være for stor, hvis begge datasæt følger den samme sandsynlighedsfordeling. Problemet kan eksempelvis genskabes ved at begrænse størrelsen af det anvendte træningssæt i denne opgaves projekt.

Træningsprocessen for kunstige neurale netværk kan også være meget tidkrævende. Dette skyldes de mange parametre, lag og neuroner, der indgår i de matematiske beregner i computeren. Nogle modeller kan have billioner af parametre. For at beregne gradienten for så mange parametre kan optimeringsalgoritmer anvendes såsom mini-batch gradientnedstigning (afsnit 3.2.2). Dette kommer dog på bekostningen af mindre effektive trin mod det lokale minimum. Hertil opstår endnu et problem under optimeringsprocessen, da det ikke kan garanteres at det lokale minimum, som modellen

[28] (towardsdatascience, 2018)
[29] (Education, Convolutional Neural Networks, 2020)

konverterer imod, er det bedste. Dette kan ofte kun løses ved at gennemføre træningsprocessen forfra med andre initiale værdier for modellens parametre. Dette gør processen endnu mere tidkrævende.

10. Konklusion

Dette studieretningsprojekt har redegjort for matematikken samt teorien bag kunstige neurale netværk. Kunstige neurale er komplekse funktioner sammensat af neuroner og lag samt massevis af parametre og variabler. For at regne med alle disse parametre anvendes matrixer, der nemt kan repræsentere alle disse parametre som elementer i et skema. For at beregne aktiveringsværdierne i et kunstigt neuralt netværk anvendes matrixmultiplikation, hvilket er en matematisk operation, der gør det muligt at beregne den vægtede sum for alle neuroner i et lag samtidigt. Under træningsprocessen for en model kan et kunstigt neuralt netværk lære at finde sammenhænge og mønstre i store datasæt. Til dette anvendes backpropagation algoritmen til at beregne gradienten for omkostningsfunktionen. Målet med dette er, at omkostningsfunktionen over tid skal konvertere mod et lokalt minimum.

På baggrund af de 7 trin for konstruktionen af en Machine Learning model er et kunstigt neuralt netværk blevet udarbejdet. Denne model kan med høj nøjagtighed klassificere sort-hvid pixelbilleder af tøj. Selve modellen er blevet trænet med et stort datasæt gennem flere iteration, og netværkets struktur er konstrueret ved brug af den iterative arbejdsmetode. Sammen med matematikken har faget informatik implementeret de matematiske formler for backpropagation algoritmen og matrixmultiplikation i kode.

Kunstige neurale netværk kan anvendes til at udføre mange relevante opgaver, dog har de også nogle begrænsninger. Begrænsningerne er i særdeleshed relateret til opsamlingen og forberedelsen af data. For at konstruere og træne en præcis og velfungerende Supervised Learning model, kræves store mængder af labelled og højkvalitets data. Dette kan ofte være svært at komme i besiddelse af. Dertil kræver forskellige datatyper ofte en del forberedelse inden, det kan blive brugt. Overfitting er også et hyppigt opstående problem for kunstige neurale netværk, hvor modeller memorerer et datasæt frem for at finde forbindelser og mønstre. Dertil kan træningsprocessen ofte være meget tidskrævende for store modeller.

11. Litteraturliste

Hjemmesider:

7 Major Challenges Faced By Machine Learning Professionals. (2021, 10 13). Retrieved 04 07, 2022, from geeksforgeeks: https://www.geeksforgeeks.org/7-major-challenges-faced-by-machine-learning-professionals/#:~:text=Data%20plays%20a%20significant%20role%20in%20the%20machine,data%20can%20make%20the%20whole%20process%20extremely%20exhausting.

Aflak, O. (2015, 11 18). *Neural Network from scratch in Python*. Retrieved from towardsdatascience: https://towardsdatascience.com/math-neural-network-from-scratch-in-python-d6da9f29ce65

Amor, E. (2020, 02 03). *Machine Learning Evaluation Metrics*. Retrieved 04 03, 2022, from Medium: https://medium.com/ml-cheat-sheet/machine-learning-evaluation-metrics-b89b8832e275

Barta, A. (2022, 02 25). *What is Mean Squared Error?* Retrieved 04 04, 2022, from study: https://study.com/learn/lesson/mean-squared-error-formula.html

Brownlee, J. (2018, 07 20). *Difference Between a Batch and an Epoch in a Neural Network*. Retrieved 04 03, 2022, from Machine Learning Mastery: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

Ebbesen, G. R. (n.d.). Retrieved 04 07, 2022, from uvmat: https://www.uvmat.dk/paradig/not/n242-1.pdf

Education, I. C. (2020, 10 20). *Convolutional Neural Networks*. Retrieved from IBM: https://www.ibm.com/cloud/learn/convolutional-neural-networks

Education, I. C. (2020, 08 19). *Supervised Learning*. Retrieved 04 07, 2022, from IBM: https://www.ibm.com/cloud/learn/supervised-learning

Gillis, A. S. (n.d.). *object-oriented programming (OOP)*. Retrieved 04 05, 2022, from techtarget: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

Nielsen, M. (2019, 12). *Neural Networks and Deep Learning*. Retrieved 04 07, 2022, from neuralnetworksanddeeplearning: http://neuralnetworksanddeeplearning.com/chap2.html

Reinforcement learning. (2021, 10 18). Retrieved 04 07, 2022, from Geeks for geeks: https://www.geeksforgeeks.org/what-is-reinforcement-learning/#:~:text=Reinforcement%20learning%20is%20an%20area%20of%20Machine%20Learning.,path%20it%20should%20take%20in%20a%20specific%20situation.

Sanderson, 3. -G. (2017, 11 03). *Backpropagation calculus | Chapter 4, Deep learning*. Retrieved from Youtube: https://www.youtube.com/watch?v=tIeHLnjs5U8

Sanderson, 3. -G. (2017, 10 05). *But what is a neural network? | Chapter 1, Deep learning*. Retrieved 04 07, 2022, from youtube: https://www.youtube.com/watch?v=aircAruvnKk

Sanderson, 3. -G. (2017, 10 16). *Gradient descent, how neural networks learn | Chapter 2, Deep learning*. Retrieved 04 07, 2022, from Youtube: https://www.youtube.com/watch?v=IHZwWFHWa-w

Srinivasan, A. V. (2019, 09 07). *Stochastic Gradient Descent — Clearly Explained !!* Retrieved 04 03, 2022, from towardsdatascience: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31

The Complete Guide to Machine Learning Steps. (2021, 11 12). Retrieved 04 03, 2022, from simplilearn: https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-steps?source=sl_frs_nav_user_clicks_on_previous_tutorial

towardsdatascience. (2018, 12 25). *Introduction to Data Preprocessing in Machine Learning*. Retrieved 04 07, 2022, from https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d

Wood, T. (n.d.). *Sigmoid Function*. Retrieved 04 07, 2022, from DeepAI: https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function#:~:text=A%20sigmoid%20function%20is%20a%20type%20of%20activation,output%20to%20a%20range%20between%200%20and%201.