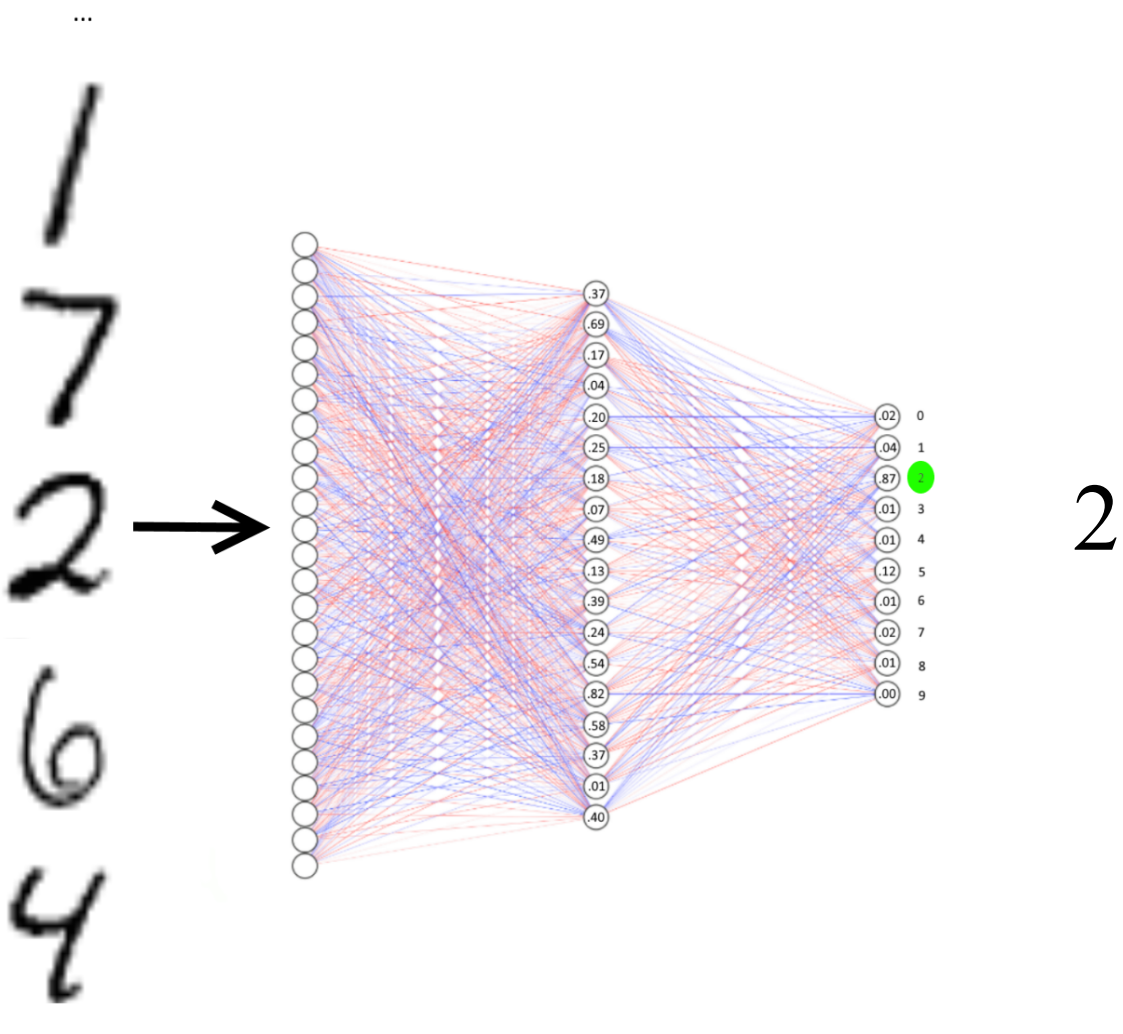
Innovativt Studieretningsprojekt

Neurale netværk til genkendelse af håndskrift

Albert Lenz Bærentsen, 2017s

Ved vejledning fra Mads Peter Hornbak Steenstrup



forstørrelsesbobbel

Matematik A, Informatik C

Afleveret 2020-03-24

Resumé

Dette projekt fokuserer på neurale netværk, nærmere bestemt flerlags-perceptron modellen. Formålet med opgaven er, at undersøge anvendeligheden af modellen til optimering af bestillingsblanket-systemet, hvilket en virksomhed som Ry-Fotos stadig benytter. Indledningsvis følger en redegørelse for flerlags-perceptron-modellen som klassificeringsmetode, efterfulgt af en brugerundersøgelse af Ry-Fotos som basis for casen. Herefter vil matematikkens rolle i neurale netværk undersøges, og hvorfor infitesimalregning er relevant her. Denne relevans underbygges af et deduktivt bevis for gradientens retning mod lokalt maksimum. Herefter vil løsningen til casen, der skal aflæse telefonnumre og postnumre, blive kodet samt trænet fra bunden i programmeringssproget C#, ved brug af MNIST-Datasættet. Løsningen vurderes slutteligt ved brug af binomialmodeller og regressionsanalyse, og der perspektiveres til fremtidige neurale netværk. Opgaven konkluderer, at løsningen var uimplementerbar på casen, da succesraten på 88% pr ciffer er langt fra optimal til casen. Det gav en succesrate på 36% og 60% for henholdsvis telefonnumre og postnummer-aflæsning. En optimering mod mere implementerbare resultater ville kræve et øget kompleksitetsniveauet, en større processorkapacitet, og allermest en længere tidshorisont.

Indholdsfortegnelses:

Indledning

Vi mennesker har i mange årtier overvejet, om vores teknologi en dag i fremtiden vil inkludere bevidste, kognitive væsener. Vi portrætterer det i film som Wall-E og Terminator, hvor teknologiens fremskridt bliver til en dystopisk affære. Så langt er vi dog ikke -endnu. Men vi har dog forsøgt at genskabe den menneskelige hjerne til at løse problemstillinger. Neurale netværk er netop dette forsøg på at skabe intelligens med afsæt i den menneskelige hjerne.
Projektet her vil fokusere på en mindre del af det det kolossale emne, som neurale netværk er. Nærmere bestemt, flerlags-perceptron modellen, som egner sig bedst til billedklassificering. Jeg vil implementere netop sådanne neurale netværk på en case som Ry-fotos, en enkeltmandsvirksomhed, der aflæser og klassificerer cifrene på en bestillingsblanket for skolefotos. Denne optimeringscase vil blive kodet og trænet i programmeringssproget C# ved brug af MNIST-datasættet, fuldkommen fra bunden, og herefter vurderet ved brug af binomialmodeller, regressionsanalyse, og test. Før dette vil der dog blive redegjort for, hvorfor og hvordan neurale netværk bruges til klassificering.
Ydermere vil jeg undersøge, hvordan flerlags-perceptron modellen fungerer matematisk ved matematisk modellering, samt hvordan denne trænes ved brug af en gradient. Gradientens egenskab indenfor monotoniforhold vil heri også blive bevist ved deduktion.
Til sidst vil der blive perspektiveret til, hvad man kan forvente af neurale netværk i fremtiden.

Klassificering

Klassificeringprocedurer er måder eller mekanismer, hvorpå man forsøger at finde de egenskaber ved et input, der definerer hvilken klasse, det burde tilhøre. En god klassificeringsprocedure, er én, hvor denne sammenhæng er fundet og brugt mest hensigtsmæssigt. For mennesker er denne proces ofte nem og implicit i vores måde at opfatte verden på. Eksempelvis vil langt de fleste mennesker kunne genkende og klassificere et håndskrevet 8-tal ved dets karakteristiske udseende i form af to cirkler oven på hinanden trods det faktum at der er utallige variationer i udformningen af et håndskrevet 8-tal. Prøver man at modellere denne tilsyneladende simple procedure matematisk, støder man på problemer med genkendelsen af de mange forskellige måder at skrive et 8-tal på. Her kommer de neurale netværk til anvendelse.

Klassificering er en af de mest brugte anvendelser af neurale netværk, og regnes for at være bedre end konventionelle klassificeringsmetoder[1]. Dette kan underbygges teoretisk fra flere forskellige perspektiver. Først og fremmest har neurale netværk muligheden for at tilpasse sig data selvstændigt, uden underliggende menneskelige specifikationer. Foruden dette kan neurale netværk bruges på hvilket som helst klassificeringsproblem, og kan approximere forholdene i dataene til hvilken som helst præcision. Dette gælder også komplekse, realistiske sammenhænge, da neurale netværk ikke blot finder lineære sammenhænge.

Klassificerende neurale netværk bruges ofte til billedgenkendelser, som da typisk ikke har nemme klassificeringsprocedurer. Det kunne f.eks. være, når googles captcha spørger, hvor der er et bord på et billede. Dette bliver formentlig træningsdata til et neuralt netværk, der skal kunne genkende bestemte objekter på billeder. Google er i vid udstrækning en data-virksomhed, men andre virksomheder kunne også tage brug af neurale netværk. F.eks. kunne et neuralt netværk klassificere tøjartikler i en modevirksomhed, eller aflæse bestillingssedler til skolefotografer. Det sidste vil jeg tage fat i som min case.

[1] Abhishek, D. (2013)

Ry-Foto

Én af de sidste steder i samfundet, hvor digitaliseringen af den ene eller den anden grund ikke har sat ind fuldgyldigt endnu, er bestillingen af skolefotos. Blanketterne, hvorpå man skal skrive fornavn, efternavn, postnummer, adresse & telefonnummer og selvfølgelig hvilke fotos man vil have, er stadig en fysisk affære, hvilket umiddelbart kræver menneskelig aflæsning. Skønt at det havde været nemmere for disse virksomheder blot at acceptere bestillinger elektronisk, holdes der fast i denne tradition. Virksomheden jeg vil tage udgangspunkt i her er Ry-foto[2], som har en blanket der ser ud således:


Alle billederne har et 4 cifrede nummer som skal bruges i forbindelse med bestilling.
NB: Det er ikke tilladt at bruge billederne fra bestillings albummet på Facebook eller lignede. Uretmæssig brug af billeder faktureres til barnets forældre.

Elev navn for og efternavn: _____ Gruppe: _____

Forældre for og efternavn: _____

Adresse: _____ By _____ Postnr: _____

Tlf: _____ mail:_____

Figur 1: Ry-fotos bestillingsseddel til skoleelever

Fotograf Per Bille sælger sin service til f.eks. restauranter, bryllupper, og konfirmationer, men også til skoleelever. Dvs., at mens Per Bille blot får en bestillingsseddel pr. restaurant, bryllup eller konfirmation, får han omkring 28 pr. skoleklasse. Det betyder, som bestillingsblanketten ovenfor viser, mange aflæsninger af forskellige variabler, mange gange. Denne aflæsningsproces må dermed hurtigt blive langtrukken og dyr i mandetimer, da Per Bille med fotografi som profession, må fotografere mange skolebørn på et givent år.

Ry-fotos har her tydeligt en udgift, der ikke er blevet optimeret til nutidens teknologi, der kunne spare dem mandetimer og dermed reducere deres udgifter. Derfor vil jeg forsøge at skabe et klassificerende neuralt netværk, der kan afhjælpe denne problematik. Dvs. at gøre aflæsningsprocessen minimal i antal mandetimer.

Hertil kommer der altså flere benspænd til udviklingen af dette netværk. Først og fremmest er bestillingssedler ikke nødvendigvis skrevet i de kønneste tegn. Informationen skal altså skrives i

let læselige tegn, hvis ikke netværket skal konstrueres til klassificering af skråskrift eksempelvis. Dette ville i så fald komme i karambolage med den profitmargin ry-foto står til at vinde ved et neuralt netværk, der har korrekt og hurtig aflæsning af flere skrifttyper og tal. Et sådanne netværk vil være dyrt at implementere, hvilket ikke er en realistisk løsning for en enkeltmandsvirksomhed. Et neuralt netværk til denne case vil altså fortrinsvis være realistisk at implementere, hvis tegn og tal er let læselige. Den billigste og nemmest implentérbare løsning ville være et netværk, der kunne aflæse tal & cifre, og derfor sætter jeg hovedkravene i løsningen ved at spare mandetimer ved aflæsning af postnumre og telefonnumre.
Hertil kommer den europæiske persondatabeskyttelseslov, der sætter grænser for, hvordan man må opbevare og behandle personlige oplysninger. Dette sætter altså grænser for, hvorvidt løsningen må sende data til uigennemsigtige tredjeparts programmer, hvor dataenes fortrolighed kan kompromitteres.

Programmet skal nærmere bestemt have hovedfunktionen:

1. Aflæsning af et tal på et givent billede. Dette billede skal i forvejen være manipuleret til bestemte dimensioner, hvilket kan lade sig gøre ved inddeling af cifrene i telefonnumre og postnumre i individuelle kvadrater på bestillingssedlen.

Hertil kommer de sekundære attributter, som løsningen i praksis skal kunne opfylde:

2. Ingen anvendelse af tredjeparts programmer, hvor personlig datas fortrolighed kan kompromitteres.
3. Hastighed af netværket, så programmet selvfølgelig er hurtigere end menneskelig aflæsning.

[2] Ry-foto.dk

Neurale netværks matematikbog

Et neuralt netværk er i sig selv blot en algoritme, der ligesom en énvariabelsfunktion har et input, og et output. Neurale netværk fungerer på forskellig vis, og ikke alle typer af neurale netværk computerer uden feedback til forrige neuroner som feedforward neurale netværk gør. Flerlags-perceptron modellen er et feedforward neuralt netværk, og det er med afsæt i denne model, at jeg vil redegøre for de centrale matematiske dele i et neuralt netværk som helhed.
Flerlags-perceptron modellen computerer dog ikke blot direkte fra et input til et output. Denne indeholder derimod flere "lag" af neuroner, der som udgangspunkt er definerede på baggrund af de forrige lags neuronværdier[3].
Neurale netværks indre komponenter er dybt forbundne, og hver definition af disse komponenter kræver en definition af en anden. Jeg vil altså forsøge at starte med de mest håndgribelige komponenter, inden de mere abstrakte komponenter forklares.
Jeg vil desuden i redegørelsen og resten af min opgave bruge udtrykkene vektor, liste og array som substitutioner for hinanden. Dette er tilfældet, fordi de definerer den selvsamme datastruktur, men har forskellige konnotationer, specielt rummeligt.

Neuron

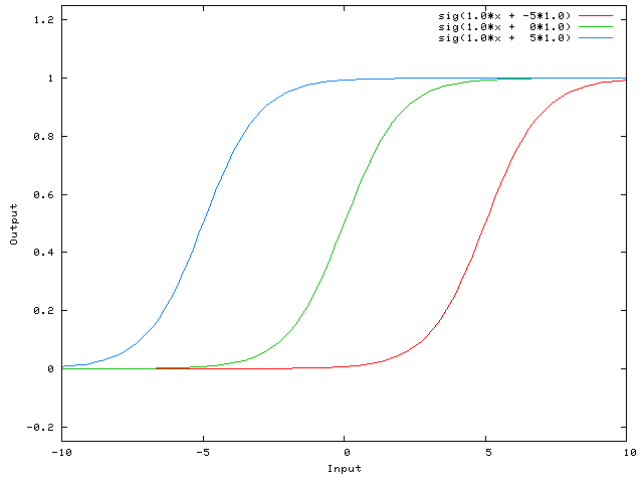
En neuron er i matematisk forstand blot en pladsholder for en værdi, hvilket altså giver neurale netværk deres navn[4]. En neuron kan dermed også blot benævnes en variabel.
Neuroners værdi bestemmes i neurale netværk på forskellig vis, men i flerlags-perceptron modellen bestemmes disse af perceptron-algoritmen[5], hvis ikke de ellers bruges som pladsholdere til input-værdier[6].

Vægte

En vægt i et neuralt netværk skal forstås som magnituden i forbindelsen mellem 2 givne neuroner. Det er blot en variabel, der indgår i perceptronen-algoritmen, hvor værdimængden kontra neuroner ikke er begrænset.
Antallet af vægte mellem 2 lag gives altså ved produktet af antallet af neuroner i det første lag og det næste. Dette lægger op til at bruge matricer til at holde vægte, frem for andre datastrukturer.
Vægte er sammen med bias de eneste variabelværdier i et neuralt netværk.

Bias

Bias er ekstra konstanter i perceptron-algoritmen, der adderes til en neurons sum, uanset hvad de forrige neuroner og vægtes værdier var. Bias forskyder altså den endelige værdi af neuronen, uanset hvilke input denne indføres. I figuren her ses de forskellige grafer for den samme aktiveringsfunktion og input, men med forskellige bias.



Figur 2: Sigmoid-funktion med forskellige bias

Som vi kan se på graferne, skal en endelig værdi af neuronen på eksempelvis 0.5 uden et bias, have en samlet vægtet sum på 0. Har neuronen derimod et bias lig 5, skal den vægtede sum af input være præcis -5 for at give det selvsamme output. Bias "skubber" altså aktiveringsfunktionen enten til højre eller venstre.

Et simplere eksempel kunne været konstantleddet b i en linear funktion[7]:

$$f(x) = ax + b$$

Er inputtet, eller x, lig 0, bestemmes funktionen i stedet blot ved:

$$f(x) = a * 0 + b$$

$$f(x) = b$$

[3] Sanderson, 2017, 1
[4] Sanderson, 2017, 1
[5] Se side 11
[6] Se Figur 10. Første lag af neuroner fra venstre mod højre har ingen input-værdier; disse er pladsholdere for materialet, der skal klassificeres.
[7] "Linear regressionsanalyse", Systime

Havde man ikke konstanten b, ville det altid resulterer i, at $f(0) = 0$, hvilket ville hæmme fremkomsten af en god regression. Man bruger altså konstantleddet b, til at "skubbe" funktionen op eller ned af 2. Aksen, hvilket gør regressionsmulighederne bedre. Det samme gør sig gældende i neurale netværks neuroner, der jo blot er funktioner af deres input.

Aktiveringsmetoderne

En aktiveringsfunktion en funktion brugt i perceptronen. Aktiveringsfunktionen sørger for at indføre "ikke linearitet" til netværket, der ellers blot havde været iterative prikprodukter af vægtmatricer og neuronvektorer fra input til output. Denne iteration af lineære operationer ville ikke kunne approximere et realistisk forhold mellem billede og klassificering[8].

En flerlags-perceptrons model som klassificeringsprocedure med flere outputneuroner, vil typisk ikke bruge en aktiveringsfunktion, der nødvendiggør et boolsk output.[9]

Dette er tilfældet da:

- Aktiveringsfunktionen ikke kan være differentiabel, og netværket kan dermed ikke trænes ved gradient descent.[10]
- Boolske output kan lade input blive klassificeret til flere klasser, hvis flere end 1 output deler klassificerende værdi.

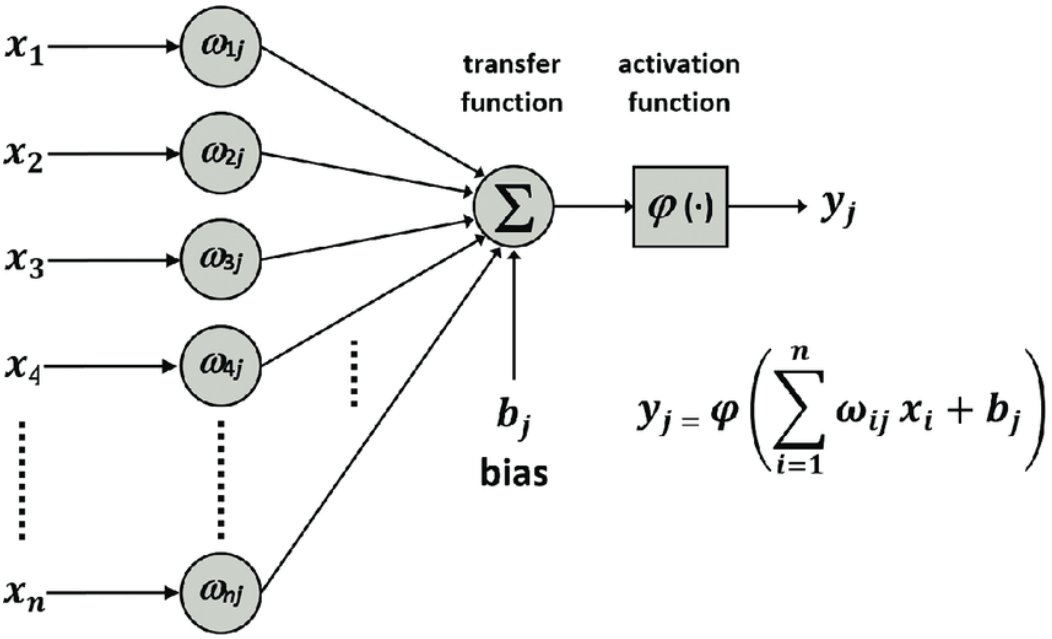
I flerlags-perceptron modellen bruges altså ikke nødvendigvis en boolsk funktion. Her kan en differentiabel funktion som sigmoid-funktionen benyttes i stedet.

Et problem ved sigmoid funktionen er, at dennes tangenthældning bliver tættere og tættere på 0, jo længere dennes input er fra 0. Har en vilkårlig neuron altid en positiv effekt på netværket, kan dennes værdi ikke stige til mere end 1. Derfor er flere neurale netværk nu bygget rundt om aktiveringsfunktionen ReLU, eller rectified linear unit[11]. Denne funktion sætter blot værdimængden af sit output til mindst 0.

[8] Oppermann, 2019
[9] Variabel som blot har 2 mulige værdier: korrekt eller falskt, 0 eller 1
[10] Se side 15.
[11] Oppermann, 2019

Perceptronen

Perceptronen er den grundlæggende binære klassificeringsprocedure i neurale netværk. Proceduren tager et vidst input, og outputter som udgangspunkt et boolsk variabel[12]. Outputtets værdimængde er dog fuldkommen afhængig af den anvendte aktiveringsfunktion. Perceptronen er algoritmen, der bruges til opdateringen af neuronværdierne i lagene på nær inputlaget[13]. Neuronværdierne defineres ved den vægtede sum af deres inputs, her forrige lags neuroner, som sammen med et bias bliver aktiveret i en aktiveringsfunktion. Denne process bliver udført iterativt for hver neuron i laget. I modellen her definerer x en neurons input, w disses vægte, b bias og Phi aktiveringsfunktionen. *y* bliver således neuronens værdi.



Figur 3: Perceptron-algoritmen

x kan her betegnes som en vektor af n-dimensioner. ”n” betegner her blot, hvor mange inputs der er til rådighed. Dette faktum afspejles i summationstegnet. Biasvektoren *b* kan ligeledes betegnes som en vektor til j længde, samme længde som neuronvektoren y. *w* er dog en rektangulær *array*, eller en matrix på dansk. Dette bliver nødvendigt i og med, at flerlags-perceptron modellen er fuldkommen forbundet, dvs. at hver neuron i et lag er forbundet med hver neuron i det forrige. Er der altså 4 neuroner i første lag, og 3 i det næste, vil der være i alt 12 forbindelser. Vægtmatricen vil således blive en matrix, af 4 kolonner og 3 rækker, der dermed indeholde vægtene af de 12 forbindelser. Dette er således ikke en vektor.

Da modellen her er forsimplet til et enkelt perceptronlag, har vi ikke at gøre med en flerlags-perceptron model. For at datastrukturerne w, x, b & y, skal kunne generaliseres til brug igennem hele modellen, skal vi altså udvide disse. Her vil jeg udelade vektorpile, da formlen hurtig bliver uoverskuelig:

$$y_j = \varphi(b_j + \sum_{i=1}^{n} w_{ij} \cdot x_i)$$

Dette gøres ved at lave en liste, med længden l, hvor l er antallet af lag. Hver vektor eller matrix bliver herefter indekseret efter, hvilket lag det tilhører. Dette vil vi bruge hævet skrift til at visualisere, som ikke henviser til en potens:

$$y_j^l = \varphi(b_j^l + \sum_{i=1}^{n} w_{ij}^l \cdot x_i^l)$$

Det noteres selvfølgelig, at neuronerne i et lag y^l ikke populeres med den vægtede sum fra sit deres lag, men det forrige:

$$y_j^l = \varphi(b_j^l + \sum_{i=1}^{n} w_{ij}^{l-1} \cdot x_i^{l-1})$$

Denne definition bruges nu i et flerlags-perceptron netværk, ved at lade første lags neuronvektor $x^{(l=1)}$ populeres med input, før næste lags neuronvektor defineres ved $l = 2$. Samlet set bruger vi altså funktionen:

$$x_j^l = \varphi(b_j^l + \sum_{i=1}^{n} w_{ij}^{l-1} \cdot x_i^{l-1})$$

til at populere alle neuronvektorer på nær første lag.

Perceptronen kan også beskrives ved prikproduktet[14] af forrige neuronvektor og vægtmatricen, adderet til bias. Dette er et markant kortere udtryk:

$$\vec{x}_l = \varphi(W_l \cdot \vec{x}_{l-1})$$

Hvor l er laget, W_l vægtmatricen til laget, og $\vec{x}_l$ neuronvektoren til laget. Her skal bias dog integreres i den vægtede sum af neuronvektorerne som en ekstra neuron, der har en konstant værdi.

[12] Variabel som blot har 2 mulige værdier: korrekt eller falskt.
[13] Sanderson, 2017, 1
[14] Hesselholt & Wahl, 2017, def. 2.1.4

Cost-funktionen

Cost-funktionen er et udtryk for, hvor meget netværkets output stemmer overens med, hvad det i en perfekt verden burde. Den beskriver altså hvor dårlig modellen er til at estimerer forholdet mellem 2 variabler, som et billede af et håndskrevet 8-tal og klassificeringen af et 8-tal som netop dette. Cost-funktionen altså et sidestykke til en matematisk regression, hvor træningseksemplerne er vores kendte værdier. Cost-funktionen defineres i et neuralt netværk ved[15]:

$$Cost(o,r) = \sum_{i=1}^{n} (o_i - r_i)^2$$

Hvor o er de reelle output, og r de perfekte output i forhold til kendte punkter. Dette er fuldkommen analogt til den omkringliggende proces angående regression og udtrykket man i en regression vil finde minimum af, for at finde den bedste tilpasning til dataene. Tager man udgangspunkt i en mere simpel lineær regression, vil dette også være summen af de kvadrerede residualer i anden, dog hvor baggrundvariablerne blot er a & b i modsætning til de utallige baggrundsvariabler tilstede i et neuralt netværk:

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \, , \hat{y} = ax_i + b$$

Linjen $\hat{y}$ betegner her én vilkårlig linje på baggrund af de 2 variable a & b, og udtrykkets samlede værdi, hvor godt eller dårligt denne linje passer på sin data[16]. Metoden man ville bruge i en lineær regression, for at finde den bedste linjeforskrift til dataen, hedder MKM, eller Mindste Kvadraters Metode. Man forsøger altså, at gøre summen af de kvadrerede residualer mindst muligt, ved at ændre på variablerne a & b. Det er selvsamme princip der bruges i neurale netværk ved gradient descent eller backpropagation, blot med langt flere variabler i form af vægtmatricer og bias-værdier.

Cost-funktionen er altså i sin essens et udtryk for, hvor dårligt et neuralt netværks algoritme-output passer på de givne data, defineret ved summen af de kvadrerede residualer til dataen.

[15] Sanderson, 2017, 2
[16] "Linear regressionsanalyse", Systime

Beviset for, at gradienten peger mod maksimum

En flervariabelsfunktion som:

$$f(x, y, \cdots, n)$$

Har en gradient defineret som så[17]:

$$\nabla f(x, y, \cdots, n) = \begin{bmatrix} \frac{df}{dx} \\ \frac{df}{dy} \\ \vdots \\ \frac{df}{dn} \end{bmatrix}$$

Gradienten er som bekendt opbygget af de partielt afledte til hver variabel i den originale funktion. Vi antager selvfølgelig, at funktionen f er differentiabel.

Gradienten her er en vektor, og dens længde definerer tangenthældningen i punktet, men f er en funktion af flere variable, der alle er med til at definere, hvor denne vektor peger hen. Jeg vil nu bevise, at gradienten til en hvilken som helst flervariabels funktion, peger mod et lokalt maksimum.

Vi starter med, at definere en retningsafledt funktion[18], som peger mod lokalt maksimum. Denne funktion tager udgangspunkt i et vilkårligt punkt i definitionsmængden til f A, og en enhedsvektor r. Denne retningsafledte funktion har egenskaben, at den kan omskrives til følgende udtryk:[19]

$$f'(A; \vec{r}) = \nabla f(A) \cdot \vec{r}$$

Sætningen siger, at den retningsafledte til et vilkårligt sted i definitionsmængden på grafen f, er det samme som prikproduktet mellem gradienten til f i punktet, og retningsvektoren r.

Vi skal nu bevise, at gradienten peger mod lokalt maksimum.

Til det tager vi fat i prikproduktet på højre side af lighedstegnet. Vi er interesserede i at se, hvornår dette prikprodukt er størst, da dette vil give os den retningsvektor, der har den stejleste hældning. Dette er tilfældet, fordi gradienten i punktet selvfølgelig er en konstant vektor, mens enhedsvektoren r's retning står udefineret. Prikproduktet mellem disse 2 vektorer defineres ved:

$$D_{\vec{r}} f(A) = |\nabla f(A)| \cdot |\vec{r}| \cdot \cos(v)$$

Vi ved desuden, at enhedsvektoren r en enhedsvektor. Den har altså en længde på 1:

$$D_{\vec{r}}f(A) = |\nabla f(A)| \cdot \cos(\mathrm{v})$$

Vi har altså blot produktet af vinklen mellem de to vektorer, og længden af gradienten tilbage. Max-værdierne af dette udtryk finder vi altså, når v, vinklen mellem vektor gradienten & enhedsvektoren r, er lig 0, da Cos(0)=1. Dette giver os altså 2 konklusioner:

$$\max\left(D_{\vec{r}}f(A)\right) = |\nabla f(A)|$$

$$\nabla f(A) \parallel \vec{r}$$

Den største værdi af den retningsafledte i punktet A er altså blot defineret ved magnituden af gradienten af f i selvsamme punkt. Desuden, fordi at vinklen mellem de 2 vektorer er 0, må de altså også være parallelle.
Den retningsafledte i punkt A i retning r definerede stejlheden af hældningen i selvsamme punkt og retning. Den største værdi af den retningsafledte i punktet A findes altså i gradientens retning. Gradienten peger altså i den stejleste retning i A, og dermed mod lokalt maksimum. Herved er beviset gennemført.

Backpropagation

Backpropagation er algoritmen hvorpå et kunstigt neuralt netværk trænes ved brug af minusgradienten af cost-funktionen. I teorien foregår dette, ved at tage gradienten til hvert træningseksempel på netværket; tage gennemsnittet af disse gradienter, og rette vægte & bias til på baggrund af denne værdi i minus[20]. Dette vil principielt matematisk altid give en lavere værdi af cost-funktionen, en mere præcis gradient, men vil give blot et ”trin” i den rigtige retning af lokalt minimum. I praksis vil man gerne have flere ”trin”, da træningstid også er en faktor at have

[17] Sanderson, 2017, 4
[18] Kro, T & Solovej, J. (2003)
[19] 1. Grøn, B, m.fl., 2019, 6.3, sætning 5
[20] Sanderson, 2017, 4

i mente. Et trin er her en billedlig fortolkning af den numeriske, langsomt tilnærmende metode. Dermed går man på kompromis med præcision for hastighed, ved at computere og anvende gennemsnitsgradienten på 100 træningseksempler, frem for 60.000 træningseksempler. Backpropagation i sig selv er bare den visuelt orienterede anvendelse af gradient descent, hvor cost-funktionens fejl propageres bagud til de forliggende vægte, fra output mod input; og dermed retter dem. Rettelsen her er typisk ikke én til én mellem gradienten og variablen, men ganges typisk med en læringsrate. Dette sørger for mindre "trin", og dermed en bedre endelig approximation af lokalt minimum.

Implementering af løsningen

Indledende overvejelser

For at skabe et neuralt netværk, der som udgangspunkt kan klassificere numrene fra 0-9, er der mange spørgsmål der skal svares på. Hvilken struktur skal det have? Feedforward, Recurrent, Radial bias, skal det være tilfældigt? Hvordan skal det trænes, ved gradient descent eller evolution? med supervision eller uden supervision?

Disse spørgsmål finder svar i hvilken data vi skal arbejde med. Vi vil med vores netværk bruge træningsmaterialet fra MNIST-datasættet[21], som indeholder 60.000 forskellige håndskrevne tal fra 0-9 i gråskala, i dimensionerne 28x28-pixels. Strukturen her er også blevet beskrevet med selvsamme datasæt af Grant Sanderson[22] i 2017, hvor træningsintuitionen blev fremlagt matematisk, men langt fra programmeringsmæssigt og implementeringsmæssigt. Herfra vil jeg i hvert fald trække inspiration til strukturen af det neurale netværk. Denne struktur, er af flerlags-perceptron modellen. I forhold til valget af programmeringssproget kunne jeg have sprunget over hvor gærdet var lavest, og kodet det i Python. Python indeholder utrolig mange nemt tilgængelige biblioteker, der gør strukturen, træningen, og testene til one-liners, hvilket havde gjort projektet overskueligt, hurtigt, og nemt. Det mangler dog gennemsigtighed samt hvilke processer der står

[21] Datasæt, LeCun, Y. & Cortes, C. (2010),
[22] Sanderson, 2017, 1-4

bag hvert trin i udviklingen af netværket. Derfor har jeg i stedet valgt C#, et sprog jeg er bekendt med.

Strukturen af løsningen

I C# benævner vi en class, som vi kalder NN. Denne class vil fungere som vores datastruktur, så vi kan referere til en initiation af den. I teorien, når vi initiere denne class, har vi objektiviseret vores neurale netværk.[23] Classen skal dog fyldes ud med forskellige datastrukturer. Disse er i

```
int[] layers;
float[][] neurons;
float[][] biases;
float[][][] weights;
```

Figur 4: Nødvendige datastrukturer i løsningen

praksis jagged arrays, der egentlig blot er et array af arrays, eller en vektor, hvor værdierne er andre vektorer[24].

Vektoren layers benævner antallet af lag, samt hvor mange neuroner hvert lag har. Vi vælger input laget med 28x28 neuroner, et mellemlag med 36 neuroner, og et output med 10 neuroner.

Vektoren layers vil altså se således ud:

$$layers = \begin{bmatrix} 784 \\ 36 \\ 10 \end{bmatrix}$$

Vektoren neurons bliver ikke defineret endnu, men får alligevel allokeret en datastruktur, baseret på layers-vektoren. Denne er altså en jagged vektor, en 3-dimensionel vektor, hvor hver af de 3 vektor-variable er en vektor i sig selv i dimensionerne givet i layers-vektoren. ”vektor[x]” betegner her en vektor med x dimensioner:

$$neurons[3][\quad] = \begin{bmatrix} vektor[layers[0]] \\ vektor[layers[1]] \\ vektor[layers[2]] \end{bmatrix} = \begin{bmatrix} vektor[784] \\ vektor[36] \\ vektor[10] \end{bmatrix}$$

Hertil kommer vektoren biases, som har fuldkommen sammen struktur som neurons, men hvor hver af dennes endelige værdier er et tilfældigt tal mellem 0 & 1.

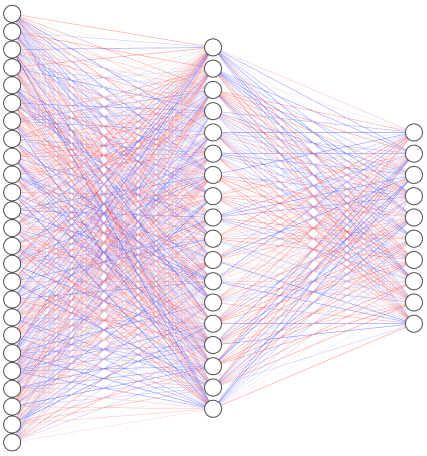
$$biases[3][\quad] = \begin{bmatrix} vektor[layers[0]] \\ vektor[layers[1]] \\ vektor[layers[2]] \end{bmatrix} = \begin{bmatrix} vektor[784] \\ vektor[36] \\ vektor[10] \end{bmatrix}, hvor\ alle\ værdier \in [0;1]$$

[23] Dette er grundet, at C# er et såkaldt objekt-orienteret programmeringssprog
[24] ”Jagged Arrays”, Microsoft .Net

Vektoren weights's struktur har en ekstra dimension, og er alle dennes endelige værdier er mellem -1 & 1. Denne jagged array skal nemlig definere hver vægt for alle neuronerne i laget, til hver neuron i forrige lag. Det er langt mere passende og forklarende at kalde dette for en vektor, der frem for værdier, indeholder andre vektorer, som selv indeholder andre vektorer, som længst ude forgreningstræet indeholder nogle værdier. Denne datastruktur kan minde om en matrix, men må ikke forveksles med pågældende. "vektor[a][b]" betegner altså en vektor af a dimensioner, hvor værdierne af denne vektor selv er vektorer af b længde.

$$weights[3-1][\quad][\quad] = \begin{bmatrix} vektor[36][784] \\ vektor[10][36] \end{bmatrix}, hvor\ alle\ værdier \in [-1;1]$$

Forskellen ligger i, at de arrays, der ligger inde i den jaggede array weights, ikke behøver være samme, uniforme størrelse. Men i realiteten er det blot en datastruktur, som indeholder de samme værdier, som hvis programmet havde brugt en matrix.



Figur 5: Forsimplet visualisering af løsningen

Feedforward mekanismen:

Alle disse datastrukturer vil vi forsøge at gengive grafisk. På billedet til højre ses en model af netværket, dog forsimplet, og uden inddragelse af alle neuronerne i de 2 første lag. Til venstre har vi input laget, til højre har vi outputlaget; Benævnt fra 0-9, og et mellemste lag, som vi kalder det skjulte lag[25]. Netværket's neuroner er som udgangspunkt 0, da vi ikke definerede dem ovenfor, ligesom vi gjorde vægtene og vores bias. Vi vil gerne have, at neuronerne kommer an på det forrige lags neuroner, ved en vægt defineret ved de tilhørende synapser dertil.
Deres værdi skal derfor ikke benævnes tilfældigt, men rettere ved formlen:[26]

[25] Abhishek, "A study of classification techniques using soft computing", Kapitel 4
[26] Tilsvarende til perceptron-algoritmen

$$a_j^{(l)} = \sigma(bias[l][j] + \sum_{n=0}^{neurons[l-1].length} weights[l-1][j][n] \cdot neurons[l-1][n])$$

Når vi computerer denne værdi laver vi først et for loop fuldkommen som ligningen ovenfor, for hver neuron i forrige lag, og inddrager snippeten nedenfor. "z" betegner summen af de vægtede neuroner i forrige lag, som sammen med biaset bliver aktiveret i aktiveringsfunktionen, for at populere vores lag af neuroner:

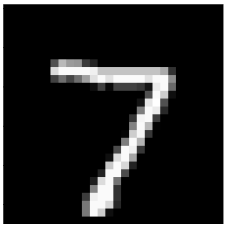
```
z += weights[i - 1][j][k] * neurons[i - 1][k];
```
[27]

Figur 6: Summering af vægtede sum

```
neurons[i][j] = activate(z + biases[i][j]);
```

Figur 7: Aktiveringsfunktion ved vægtede sum og bias

Formlen fungerer naturligvis ikke på første lag, da denne matematisk er lig nul, og kodningsmæssigt ville skabe en Index Out Of Bounds-exception, da $weights[0-1][j][n]$ ikke findes. Dette er nemlig input-laget, hvor vi indfører dataen fra MNIST-datasættet. Et sådanne billede kan ses på Figur 8, hvor de sorte pixels har værdier tættere mod 0, og de hvide har værdier mod 1.



Figur 8: Eksempel af MNIST-Træningseksempel

Vi kan nu "feede" vores netværk frem, fra inputlaget, til det mellemste lag, og til outputaget, mens vi populerere de forskellige vektorer af neuroner. Den største værdi i sidste lag bliver derfor, hvilket ciffer det neurale netværk tror, billedet forestiller. Herved bliver det neurale netværk til en klassificeringsprocedure. Den sidste neuronvektors værdier er dog tilfældige indtil videre, da alle værdier vi har defineret indtil videre har været tilfældige.

Træningen af netværket

Vores mål i denne del af processen er at sørge for, at indstille hver eneste vægt/synapse mellem neuronerne således, at output laget bliver korrekt. Gav vi altså netværket et billede af et 7-tal, ville vi regne med, at outputneuronerne der repræsenterer cifrene på nær 7 var lig 0, mens den der

[27] "+=" er en operation, der adderer en værdi til en variabel, her z.

repræsenterer 7 er lig 1. For at vise dette indfører vi en cost-funktion, der betegner, hvor langt vores output er fra det korrekte output. Denne cost defineres ved:

$$Cost = \sum_{n=0}^{9} (\mathrm{o[n]} - r[n])^2$$

hvor o er vektoren af output-laget, og r er den output-vektor, vi regner med fra et perfekt netværk[28].

Matematikken går altså nu ud på, at minimere denne funktion.

Dette gøres ikke ved at finde gradienten som helhed af denne funktion, da mine kodningsevner ikke rækker til computeringen grundet kompleksiteten og strukturen af netværket. Dog bruger vi stadig differentialregning til at rette på vægtene. Vi bruger nemlig backpropagation i stedet, der tager udgangspunkt i hvert lags enkelte vægte, hvilket kodningsmæssigt er en langt nemmere og intuitiv løsning. Det skal huskes, at disse værdier vi udregner nu er også en del af gradienten til Cost-funktionen. Dette foregår først på vægtene, der fører direkte ind i outputlaget, før vi ser på vægtene i laget før dette. Herved bliver det tydeligt, hvorfor metoden hedder backpropagation.

Vi gør dette i koden for hver neuron i outputlaget, j, og hver neuron i forrige lag, k.

```
//output = vektoren for sidste lag //expectedvalues = vektoren vi vil have den skal være // activatem = afledte aktivationsfunction
float vægt = 2 * (output[j] - expectedvalues[j]) * (neurons[1][k]) * activatem(invactive(neurons[2][j]));

weightchanges[1][j][k] += -vægt * 0.01f;
```

Figur 9: Opfangelse af gradientkomponenter

Matematisk er hvad der foregår her, kædereglen[29]. Vægt-variablen her stiller spørgsmålet om, hvordan en lille positiv ændring i vægten, ændrer en af cost-funktionens 9 elementer. Vægten indgår i en sum, som et produkt med sin foregående neuron. Denne sum blive aktiveret, og den aktiverede sum spiller en direkte rolle i elementet i cost-funktionen, som vi kalder C_j

Dette udtryk matematisk er derfor:

$$\frac{dC_j}{dw} = \frac{dC_j}{da_j} \cdot \frac{da_j}{dz_j} \cdot \frac{dz_j}{dw}$$

Hvor Cj er det element af cost-funktionen, som w henfører til; aj er defineret ovenfor; z er defineret som aj, men uden aktiveringsfunktionen (derfor har vi invactive. Det er den inverse

[28] Egentlig en regression; se redegørelsen af cost-funktionen
[29] 2. Grøn, B, m.fl., 3.5, sætning 5

funktion til vores aktiveringsfunktion. Activatem er her blot den afledte aktiveringsfunktion). Indsætter vi differentialerne med hensyn til deres respektive variabler/funktioner, får vi udtrykket:

$$\frac{dC_j}{dw} = 2 * (a[j] - r[j]) \cdot \ \sigma'(z_j) \cdot w\_neuron$$

Dette udtryk giver altså en numerisk værdi. Denne numeriske værdi beskriver altså, hvor meget denne Cj stiger, når vægten stiger med 1. Vi er ikke interesserede i, at Cj stiger, tværtimod vil vi jo minimere Cost-funktionen som helhed. Derfor tager vi i koden den negative værdi af udtrykket, og ganger denne med læringsfaktoren 0.01, tillagt weightchanges. Denne datastruktur er til start ens med weights, men indeholder til start kun værdier af 0. Med jævne mellemrum summeres weightchanges ind i weights, i takt med flere og flere træningseksempler.
I næste lag bruges samme princip. Der er blot 2 ekstra faktorer, men praktisk set påvirker denne vægt ikke direkte outputlaget, men det mellemste lag, hvilket betyder, at vægten har indvirkning på Cost-funktionen af 10 veje. De 10 måder vægten påvirker cost-funktionen på summeres altså. Denne sum indsættes herefter også i datastrukturen weightchanges.

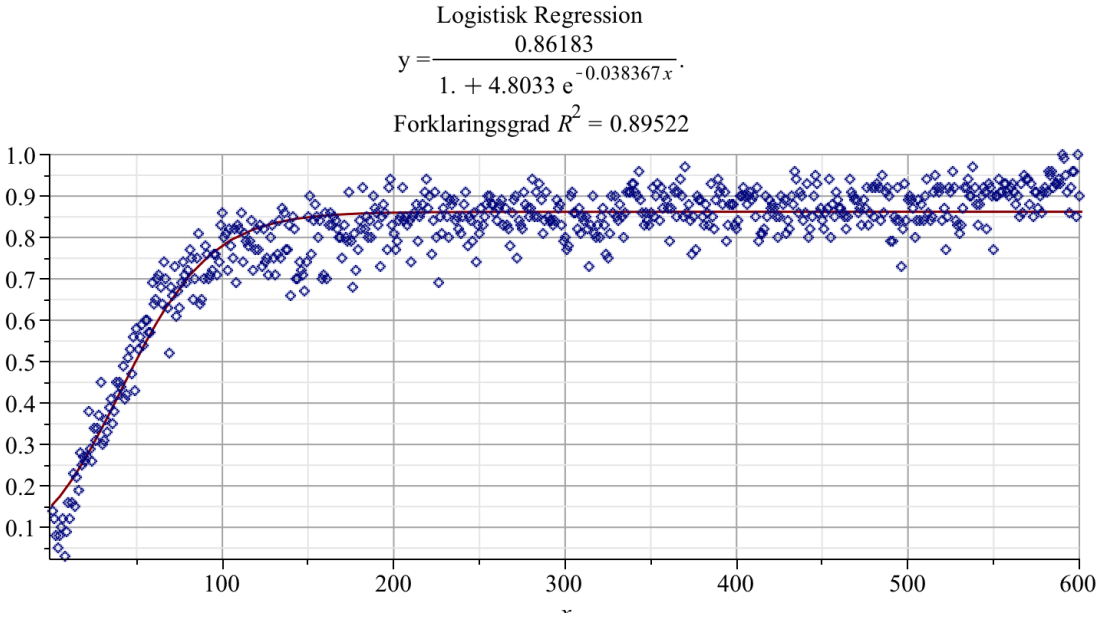
Nu er der blot træningen af netværket tilbage. MNIST-datasættet indeholder 60.000 træningseksempler, og jeg har brugt dem alle sammen til min træning. For hver af disse 60.000 eksempler udregnes først alle neuronerne i feedforward mekanismen, hvorefter vi udregner en cost-funktion, og lagrer ovenstående data for hver vægt i netværket. Blot udregning af vægtenes differentialer sker dermed i træningsforløbet:

$$(784 * 36 + 36 * 10) * 60000 = 1.715.040.000$$

-gange, hvilket dermed har taget flere timer. Classen NN, hvor alt dette er foregået, outputter sig selv til sidst i en tekstfil, hvor alle vægte & biases gemmes, samt et excelark over træningsforløbets succesrater.

Vurdering i forhold til kravspecifikationer

Test af netværket

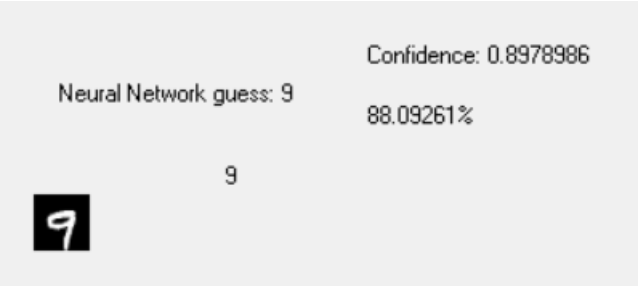


Figur 10: Løsningens succesrate & antal træningseksempler (100)

Succesraterne for netværket er her plottet over hele træningsforløbet, samt en passende regression hertil. Ud af førsteaksen har vi antal korrektioner i vægtene (én korrektion pr. 100 billeder), og anden aksen, hvilken succesrate netværket havde til den tid. Fra 0, hvor den gættede ingen af cifrene, til 1, hvor den gættede alle 100 cifre. Vi har lavet en logistisk regression på disse data, hvilket giver os et forhold mellem antal træningseksempler og succesrate, skønt forklaringsgraden er under 0.95. Vi aflæser "steady state" her[30] til 0.86, og regressionen siger os altså, at det neurale netværk vil have en succesrate på knap denne værdi, 86%.

[30] "Logistisk vækst", Webmatematik

Vi har dog faktisk ikke testet netværket på testmateriale endnu, så denne værdi er ikke nødvendigvis rigtig, specielt ikke med denne suboptimale forklaringsgrad. Derfor kører vi det endelige netværk igennem 500 billeder af testmateriale. Dette gøres fra et andet program, som inddrager classen NN og vores trænede netværk, og giver os mulighed for at diagnosticere netværkets præstation.



Figur 13: User Interface til testprogrammet

Figur 12: Eksempler på lav-konfidens eksempler

Figur 11: Eksempel på høj-konfidens fejl

Procenttallet til højre angiver succesraten på netværket, 88%, omtrent 2 procentpoint højere, end regressionen fra træningsprocessen.

Kan netværket implementeres i casen?

Skulle vi bruge dette neurale netværk til at aflæse alle cifrene i et telefon-nummer eller et postnummer, givent, at disse billeder er mainpuleret til formatet brugt i modellen, ville sandsynligheden for en korrekt aflæsning være:

$$binpdf(8, 0.88, 8) \qquad 0.3596345248$$

$$binpdf(4, 0.88, 4) \qquad 0.599695360$$

Ligning 1: Programmet Maple er her brugt til at udregne sandsynligheden for at få et komplet telefonnr. eller postnr. korrekt, ved sandsynlighedsparameteren 88% og antalsparameteren henholdsvis 8 & 4

Altså henholdsvis 36% sandsynlighed og knap 60% sandsynlighed for et telefonnummer eller et postnummer. Lever programmet op til sine kravspecifikationer, hvis ikke bare sin hovedfunktion, at kunne aflæse et ciffer på et billede?
Svaret er nej. I hvert fald ikke 12 % af tiden, hvilket er en ubrugelig stor fejlmargen i forhold til denne automatiseringscase. Men ikke blot, at programmet fejler 12 % af tiden er et problem; at identificere de 12 % lader heller ikke til at være muligt.
Programmet angiver også ”confidence”, eller rettere, hvor sikker netværket er på sit svar. Denne information er oplagt til at skille de usikre svar fra, som hvis et ciffer er skrevet som Figur 13. Der vil jo naturligvis opstå sporadiske usikkerheder om hvorvidt et tal er et 0, eller 6 på en skala som i casen, hvor hundredevis af mennesker årligt skriver telefonnumre ned. Et perfekt netværk ville angive disse eksempler ved lav confidence, og dermed ville man kunne tilkalde menneskelig hjælp til aflæsning. Dog ser vi i billedet ovenfor, at de usikre svar ikke kun er forkerte svar. Sætter vi grænsen for sikker og usikker ved 40%, opstår der altså også usikre rigtige positiver. Et usikkert, men rigtig svar, såvel som der opstår falske positiver, som set i figur 13. I casens anseende ville implementeringen af dette netværk altså betyde, at ikke nok med, at virksomheden skal bruge flere end nødvendigt mandetimer på at aflæse de usikre svar; så gætter det neurale netværk også med stor konfidens til tider forkert. Netværket er kort sagt, uimplementerbart på casen.

Forbedringer

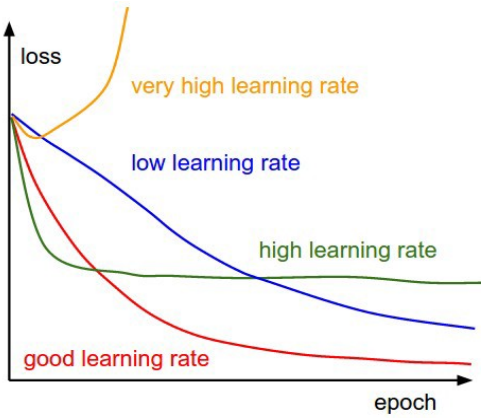
Et neuralt netværk til casen havde aldrig kunne have en konsekvent succesrate på 100%, da et fåtal af cifrene ville være uigenkendelige, da det er mennesker der har skrevet dem. Men således ville der også være tider, hvor et menneske heller ikke kunne genkende cifferet. Der vil altså altid være brug for double-checks, uanset om man bruger neurale netværk eller ej. Derfor kunne man argumentere for, at et neuralt netværk, der genkender 95% af cifrene og samtidig har lav confidence på den resterende procent ville være implementerbart, men altså aldrig fuldkommen automatiseret. Hvordan kunne min løsning have nået til dette krav?

Vi så i Figur 10 over træningsforløbet ifølge regressionskurven, at netværket ikke bliver nævneværdigt bedre om end vi bruger 20.000 eller 40.000 træningseksempler. Netværket når hurtigt sin steady state, og det lægger altså op til, at netværket har nok træningsdata, men ikke nok kompleksitet til at redegøre for alle relationerne fra billede til 10 kategorier. I min løsning er kun

brugt et enkelt lag på 36 neuroner fra billede til kategori, og dette faktum er ikke baseret på nogen form for analyse af kompleksitetsniveauet i klassificeringsproceduren. Der kunne sagtens have været 2, 3, eller 4 mellemlag, samt flere end 36 neuroner i hvert lag. 3 hidden layers havde været tættere på optimalt[31], og havde dermed sandsynligvis øget netværkets succesrate, på bekostning af overskuelighed og længere træningstid.

I kraft af flere mellemlag og flere neuroner, vil træningstiden stige mere og mere. Derfor ville større neurale netværk, dvs. med flere neuroner og lag, kræve mere processorkraft, hvis træningstiden ikke skal overstige få timer. Dog vil produktet til implementeringen, selve det neurale netværk, ikke skulle bruge meget mere computerkraft for at outputte et svar. Det er dog nødvendigt til produktionsfasen af implementeringen til casen.

Spørgsmålet kunne også stilles, om den brugte ”learning rate” til netværket var hensigtsmæssig. En learning rate er altså, hvor store ”skridt” der tages ved gradient descent. I netværket brugtes en learning rate på 0.01, hvilket ifølge regressionen fandt sit lokale minimum efter blot 1/3 af træningsmaterialet. Dette lader ifølge figuren her til at være en for høj learning rate. For at gøre træningsprocessen så hensigtsmæssig som muligt, at vægtene reelt konvergere til et minimum, kræver altså muligvis en lavere learning rate, end den brugte. Hvor lav en learning rate det er hensigtsmæssigt at bruge kommer også an på, hvor meget træningsdata netværket har til rådighed, da en lavere learning rate vil tage længere tid at konvergere til lokalt minimum.



Figur 14: Læringskurverne på baggrund af learningrate

På figurens 1. Akse henvises også til variablen epoker, som definerer, hvor mange gange netværket har trænet på hele datasættet. I netværket er der blot brugt 1 epoke, hvilket måske havde skabt en værre præstation, end hvis netværket havde trænet over datasættet flere gange. Dette ville

[31] Hornik, 1991

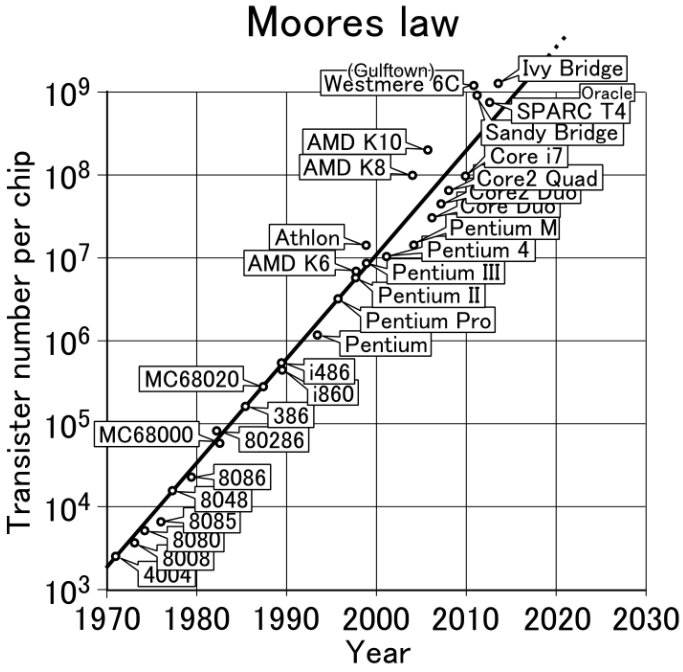
tilsyneladende skabe en bedre præstation, men dette er muligvis på bekostning af en anden variabel: overfitting. Hvis netværket havde trænet 100 gange på datasættet, costfunktionen lig 0, betyder det ikke nødvendigvis at netværket er ligeså godt i virkeligheden. Man risikerer, at netværkets variabler konvergere til et minimum i forhold til datasættet, som også passer perfekt på billeder som (figur 3), og dermed optager en forståelse af dette som et 8-tal. Netværket optager altså i høj grad "støj" i sine variabler, som ikke repræsenterer den reele sammenhæng mellem inputtet, billedet, og outputtet, klassificeringen. Havde netværket trænet for mange gange på datasættet, havde dette altså også skabt en suboptimal præstation, værre end hvad skal bruges til casen. I forhold til hvor hurtigt mit neurale netværk konvergerede til et minimum, var det altså ikke nødvendigt med flere epoker, men havde min learning rate været lavere, havde det muligvis været mere hensigtsmæssigt.
Der er mange faktorer der spiller ind i den løsning til den case jeg har fremført. Min løsning er ikke fyldestgørende, men det betyder næppe, at neurale netværk som et værktøj ikke kunne være en løsning til problemstillingen. Det kræver blot mere omsorg og omtanke, processorkraft, kompleksitet og allermest, tid. Udviklingen af denne løsning startede fra bunden, og der var forholdsvis lidt tid til tests med forskellige faktorer.

Fremtidens Neurale Netværk

Teknologien bag neurale netværk vil formentlig blive markant bedre i fremtiden. Løsningen til min case vil i fremtiden ikke blot kunne aflæse et tal eller ord, men et helt håndskrevet dokument. Desuden vil neurale netværk som værktøj finde langt flere anvendelsesområder, og de nuværende anvendelsesområder vil blive mere og mere styret af computerprocedurer. Dette gælder også områder som aktiehandel ved bedre forudsigelser om aktiemarkedet, bedre ansigtsgenkendelse, bedre selvkørende biler, og meget snart et program, der kan give private en diagnose på baggrund af symptomer og en sundhedsmæssig journal[32].

[32] Martin, "Artificial Intelligence Is Being Used To Diagnose Disease And Design New Drugs"

Det der lægger grundstenene for fremtidens neurale netværk er hovedsageligt øget processorkraft. Komplekse problemstillinger kræver komplekse neurale netværk, og kompleksiteten er direkte proportionelt med hvor mange neuroner, lag, og dermed forbindelser netværket indeholder. Flere forbindelser betyder flere dimensioner i gradienten, og det betyder længere træningstid, såvel som længere tid fra input til output. Komplekse neurale netværk er altså bundet af processorkraft, hvis de skal afløse et biologisk neuralt netværk. Moores lov, som har holdt stik siden 1965[33], illustrerer altså den trend, der gør fremtidens neurale netværk større, mere komplekse, og dermed mere nyttige på flere og flere problemstillinger. Teorien siger, at antallet af elektriske komponenter i et integreret kredsløb, som en processor, har en fordoblingshastighed på 18 måneder. Dette betyder ikke direkte, at der er et 1:1 forhold mellem elektriske komponenter og hastighed, men illustrerer altså bare, hvordan vores processorer bliver mere og mere sofistikerede; indrettede til mere og mere databehandling.



Figur 15: Moores lov. Sammenhæng mellem årstal og transistorer i chips

Nye applicationer::::

Når vi til et niveau, hvor processorkraften bare tilnærmelsesvis kan hamle op med hjernens 86 milliarder neuroner, vil neurale netværk i samspil med samtidig mekanisk teknologi formentlig have utroligt mange anvendelsesområder og nye cases. Brugen af neurale netværk til denne tid ikke nødvendigvist begrænset til en enkelt opgave som en klassificeringsprocedure, men egentlige tænkende væsener der tager input som mennesker gør; gennem sanser, med evnen ikke blot til at lære, men også evner til at rationalisere, have selvbevidsthed, og have kognitive menneskelige

[33] "Moore's Law", *Moore's Law; or how overall processing power for computers will double every two years*

tanker. Denne form for kunstig intelligens findes dog ikke endnu, men er allerede blevet teoretiseret som muligt[34]. Dette vil stort set gøre neurale netværk til et løsningsforslag til næsten hver eneste optimeringscase.

Konklusion

Redegør kort for brugen af neurale netværk til klassificering.
Neurale netværk bliver brugt til klassificering, da klassiske matematiske klassificeringsprocedurer har vist sig ineffektive på højere kompleksitetsniveauer. Neurale netværk optimerer selv deres klassificeringsprocedurer, hvilket gør dem brugbare i felter, hvor klassificeringsprocedurer er svære at modellere, som klassificering af billeder.
Foretag en brugerundersøgelse af en selvvalgte case, og kom frem til en kravspecifikation.

Ry-fotos, en enkeltmands virksomhed, bruger fysiske blanketter til at optage information som telefonnumre og postnumre. Disse informationer læses ved forbrug af mandetimer, og casen kunne altså afhjælpes vha. neurale netværk. Dette netværk vil altså have hovedfunktionen, at klassificerer håndskrevne tal til deres respektive klasse.

Redegør for centrale matematiske dele af et neuralt netværk. Kom herunder ind på optimering af funktioner med flere variable og bevis at gradienten peger mod lokalt maksimum.
Neurale netværk er egentlig et paraplyudtryk, og der findes flere forskellige matematiske modelleringer. Flerlags-perceptron-modellen bruger en lag-struktur, hvor 2 lag er henholdsvis input og output, med mindst ét lag i mellem. Flerlags-perceptron-modellen bruger feedforward mekanismen og er fuldt forbundede. Grundudtrykket til udregnelse af neuronernes værdier, med undtagelse af inputlaget, er:

$$a_j^{(l)} = \sigma(bias[l][j] + \sum_{n=0}^{neurons[l-1].length} weights[l-1][j][n] \cdot neurons[l-1][n])$$

[34] "A Holistic Approach to AI", *Strong AI,*

hvor en vilkårlig neuron defineres ved summen af en konstant og den vægtede sum af de forrige neuroner og disses vægte til den vilkårlige neuron. Dette aktiveres i en aktiveringsfunktion, således at man definerer en værdimængde for neuronen.
Netværket trænes herefter ved at minimere costfunktionen:

$$Cost(o,r) = \sum_{n=0}^{9} (\mathrm{o[n]} - r[n])^2$$

Som defineres efter samme princip som en regression, ved de kvadrerede residualer af det neurale netværks output, versus de kendte værdier.
Gradient descent er måden, hvorpå man minimerer costfunktionen, og dermed får et mere tilpasset netværk. Gradienten peger nemlig altid mod lokalt maksimum, fordi vinklen mellem denne og den retningsafledte mod maksimum skal være 0.
I implementeringsfasen af løsningen blev det hurtigt tydeligt hvor teoriorienteret matematikken er, kontra det langt mere praktiske informatiske, specielt i håndteringen af datastrukturer. Ud over denne kendsgerning foregik perceptron-algoritmen til neuronerne samt træningen af variablerne i løsningen, kerneelementerne i netværket, ved en direkte oversættelse fra den teoretiske matematik til den praktiske informatik.
Selvom løsningen opnåede en succesrate på 88% pr. ciffer, er det ikke godt nok til kravspecifikationerne og en realisering af løsningen på casen. Løsningen kunne have været realiseret, hvis kompleksiteten af netværket var højere, men det havde krævet mere processorkraft. Desuden havde læringskurven været bedre, hvis der havde været brugt en lavere learning rate, men dette havde kaldt for flere epoker, som i sidste ende kunne give netværket en værre præstation. I kraft af tidens forløb vil vi få flere og flere muligheder for at løse problemstillinger med neurale netværk, og løse de nuværende bedre end nu. Det sker på baggrund af den teknologiske udvikling, og vi vil bestemt se mere til neurale netværk i den nærmere fremtid.
Projektets løsning kan findes under:
https://github.com/abbi267/SRP

Litteraturliste

34 henvisninger

Bøger, afhandlinger & lign.:

Abhishek, D. (2013). ”A study of
classification techniques using soft computing” [Ph.d.-afhandling]. Guru Ghasidas University.

Grøn, B, m.fl. *Hvad er Matematik 3*, Lindhardt og Ringhof, 2019 (1)
Grøn, B, m.fl. *Hvad er Matematik A*, Lindhardt og Ringhof, 2013 (2)

Hesselholt, Lars & Nathalie Wahl. ”Lineær Algebra”. (2017). 2. Udgave.

Hornik, K. (1991). ” Approximation Capabilities of Muitilayer Feedforward Networks”, Wiens Tekniske Universitet.

Kro, T & Solovej, J. (2003) ”Matintro, Funktioner af flere variable”, Københavns/Oslo Universitet.

Videoer:

Sanderson, G. [3blue1brown], (5. Oktober 2017) ”Deep Learning chapter 1”, *Youtube*, https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi, [set 22/03-2020]

Sanderson, G. [3blue1brown], ”Deep Learning chapter 2”, *Youtube*, https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi, [set 22/03-2020]

Sanderson, G. [3blue1brown], ”Deep Learning chapter 4”, *Youtube*, https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi, [set 22/03-2020]

Artikler med forfatter:

Oppermann, Artem. ”Activation Functions in Neural Networks”, *Deep-Learning Academy*, https://www.deeplearning-academy.com/p/ai-wiki-activation-functions [set 28/03-2020]

Martin, Nicole. ”Artificial Intelligence Is Being Used To Diagnose Disease And Design New Drugs”, *Forbes*, https://www.forbes.com/sites/nicolemartin1/2019/09/30/artificial-intelligence-is-being-used-to-diagnose-disease-and-design-new-drugs/#597967e244db, [set 27/03-2020]

Artikler uden forfatter:

”Logistisk Vækst”, *Webmatematik*, https://www.webmatematik.dk/lektioner/matematik-a/differentialligninger/logistisk-vakst, [set 17/03-2020]

”Differentiation af sammensat funktion”, *Webmatematik*, https://www.webmatematik.dk/lektioner/matematik-b/differentialregning/differentiation-af-sammensat-funktion [set 18/03-2020]

”Linear regressionsanalyse”, Systime, (Egentlig pdf-bog) https://systime.dk/fileadmin/indhold/SupplerendeMaterialer/Markedskommunikation__1._udgave/LinearRegression.pdf, [set 24/03-2020]

”Moore’s Law”, *Moore’s Law; or how overall processing power for computers will double every two years, http://www.mooreslaw.org,* [set 27/03-2020]

”A Holistic Approach to AI”, *Strong AI,* https://www.ocf.berkeley.edu/~arihuang/academic/research/strongai3.html, [set 28/03-2020]

”Jagged Arrays”, *Microsoft .Net,*
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/jagged-arrays

Materiale:

LeCun, Y. & Cortes, C. (2010), ”MNIST handwritten digit database”