LBS™ LJUD & BILDSKOLAN KREATIVA GYMNASIET

Deep Neural Networks i praktiken

En studie på DNN-arkitekturens förmåga att lösa komplexa problem

Namn: Isak Lefèvre **Klass:** SU18A

2020-04-09

Handledare: Christoffer Janbris

Abstract

The Deep Neural Network (DNN) architecture is based on the brain-inspired Artificial Neural Networks which have increasingly become more common throughout AI research and development. In this essay I will examine the potential of this architecture and find an answer to how a DNN should be developed to solve real world problems and help with everyday life.

The simplest design of a DNN from Michael Nielsen's book was used to create a functioning system. The network was trained to identify handwritten digits of 28 by 28 pixels in grayscale. Several tests were performed during the training of DNN:s with varying sizes, start values and other parameters, where the importance and disadvantages of its different parts were revealed. The tests showed that the assumption that bigger DNN:s can handle a greater amount of complexity was correct. It was also confirmed that more practice leads to a better performance. Surprisingly a smaller amount of training examples per learning cycle ended up training more accurate DNN:s than bigger amounts.

Due to the fact that the tests were computationally heavier than expected the quantity of results were insufficient to make any concrete conclusions. Though it was still determined that it was good enough to serve as general guidelines for the importance of the different network parts. The fact that the best performing DNN got an accuracy of around 93% while only being a simpler variant of the architecture bodes very well for its potential uses in real world applications.

Innehållsförteckning

1. Introduktion

Artificiell intelligens är ett enormt ämne som innefattar allt mellan algoritmen som bestämmer vilken reklam den visar dig och den legendariska schackdatorn IBM Deep Blue som besegrade Garry Kasparov 1997. Gränsen för vad som går att åstadkomma med teknologin dras dock inte vid automatisering, utan AI har idag lyckats med bedrifter som till synes hade varit omöjliga för en människa. En av de ledande arkitekturerna idag är Deep Neural Networks (DNN) vars design är baserad på den mänskliga hjärnan. Varianter av arkitekturen har bland annat lyckats bemästra utmaningar som japanska Go och det 50 år gamla proteinvikningsproblemet inom biologin. I arbetet testas potentialen hos arkitekturen på en av de äldre utmaningarna inom bildigenkänning, sifferklassificering. Detta är för att få en förståelse för hur dessa system utvecklas samt hur rimligt det är att de fortsätter vara en trend inom maskininlärning.

Det visar sig snabbt and indexeringen är den mest utmanande delen av arbetet. Ett DNN består av ett inmatningslager som tar in data, ett aktiveringslager som ger nätverkets svar och en obestämd mängd “gömda” lager som bearbetar datan till resultatet. Datan som bearbetas placeras i vektorer och matriser som förs—lager för lager—från nätverkets början till dess slut, vilket innebär att det förekommer variabler med upp till tre index. För att ta reda på hur systemet ska utvecklas optimalt testas beståndsdelarna som verkar ha en större påverkan på pricksäkerheten. Vilka delar av DNN-arkitekturen är det egentligen som bidrar till deras avsevärda förmåga att hantera komplexitet?

2. Bakgrund

Artificiell Intelligens är ett begrepp som på senare tid har blivit alltmer aktuellt inom en mängd olika sammanhang. Dessa algoritmer och systems förmåga att på en relativt kort tid bemästra komplexa ämnen är eftersökt och kan användas inom en stor bredd branscher.

2.1 Vart kommer AI ifrån?

Synen på artificiell intelligens har genom åren förändrats drastiskt. Det är häpnadsväckande att den framsteg och den potential som ses idag mer eller mindre var förknippat med science fiction för mellan tjugo och trettio år sedan. I takt med att mänskligheten rört sig djupare in i en ny informationsålder har AI inte bara blivit större som ämne utan det har också blivit mer välkänt. Det talas om självkörande bilar, ryska nättroll och till och med robotar som med sin överlägsna intelligens fruktas en dag ta över världen. Svårt är det att tänka sig att ett sådant nytt och avancerat koncept skulle ha sina rötter i det antika grekland. Glymour menade 1992 att några av Aristoteles tankar om ontologi och kunskapsbaser än idag är centrala inom skapandet av AI. Även filosofen och matematikern Descartes uttryckte en teori inom ämnet långt innan den första datorn var uppskissad:

"If there were machines which bore a resemblance to our body and imitated our actions as far as it was morally possible to do so, we should always have two very certain tests by which to recognise that, for all that, they were not real men. The first is, that they could never use speech or other signs as we do when placing our thoughts on record for the benefit of others. For we can easily understand a machine's being constituted so that it can utter words, and even emit some responses to action on it of a corporeal kind, which brings about a change in its organs; for instance, if it is touched in a particular part it may ask what we wish to say to it; if in another part it may exclaim that it is being hurt, and so on. But it never happens that it arranges its speech in various ways, in order to reply appropriately to everything that may be said in its presence, as even the lowest type of man can do. And the second difference is, that although machines can perform certain things as well as or perhaps better than any of us can do, they infallibly fall short in others, by which means we may discover that they did not act from knowledge, but only for the disposition of their organs. For while reason is a universal instrument which can serve for all contingencies, these organs have need of some special adaptation for every particular action. From this it follows that it is morally impossible that there should be sufficient diversity in any machine to allow it to act in all the events of life in the same way as our reason causes us to act. (Descartes 1637, p. 116)"

Descartes diskuterade filosofin kring ett ämne som kom till först tre hundra år senare. Han menar i det här textstycket att återskapandet av en mänsklig hjärna är omöjligt på grund av de otaliga komponenter som krävs för att ta beslut baserade på ett helt liv av erfarenheter. Dock så säger han också att en maskin mycket väl kan bli bättre än en människa på en specifik uppgift. Efter att AI lyckats besegra världsmästarna i Schack, Jeopardy, Go och många andra spel bekräftas hans påstående (Bringsjord, S 2018).

Den allmänna uppfattningen om AI:ns eventuella uppkomst är på 1950-talet, då utvecklingen tog fart som en restprodukt av Andra världskriget. Då var det främst Alan Turing som i skapandet av sitt "Turing Test" blev en känd profil inom AI:n. Men bakom kulisserna var det först år 1956 under en konferens på Dartmouth College i Hanover, New Hampshire som termen "artificiell intelligens" uppkom. Där samlades tio på den tiden stora tänkare och diskuterade ämnet tills de kom fram till olika betydelsefulla slutsatser (Bringsjord, S 2018).

2.2 Vad är AI?

Trots den explosiva ökningen i intresse kring artificiell intelligens finns det ingen allmänt accepterad definition. Det är någonting som många har försökt göra, och i vissa fall kommit väldigt nära att lyckas med, men någon exakt definition har aldrig bestämts. Peter Norvig och Stuart J. Russell menar i sin bok *Artificial Intelligence: A Modern Approach* (2002) att definitionen för AI ska baseras på dess syfte. I följande tabell ställde de upp de i princip enda förekommande syftena som uppkommer när en AI ska designas.

	Human-Based	**Ideal Rationality**
Reasoning-Based:	System that think like humans	System that think rationally
Behaviour-Based:	System that act like humans	System that act rationally

Russell och Norvig sa alltså att det fanns två dimensioner till en AI:s syfte. En av dimensionerna till AI:n handlar om huruvida den ska vara en tänkande eller en agerande maskin. Den andra dimensionen fungerar då som en beskrivande term till den första, alltså hur den agerar eller tänker? Även där menar de att det bara finns två alternativ, ifall den ska utföra sina instruktioner så likt en människa som möjligt eller om de ska utföras så rationell som möjligt (Bringsjord, S 2018).

Men som sagt så är meningarna kring definitionen skilda, och även Europaparlamentet har sin egen version. Deras variant är centrerad kring dess användningsområden och AI:ns förmåga att återskapa mänskligt beteende. Den är alltså betydligt mer snäv än Russell och Norvigs och har ett större fokus på dess praktiska användningsområden.

"AI är en maskins förmåga att visa människoliknande drag, såsom resonerande, inlärning, planering och kreativitet.

AI möjliggör för tekniska system att uppfatta sin omgivning, hantera vad de uppfattar och lösa problem, med syfte att uppnå ett specifikt mål. Datorn mottar information (redan förberedd eller insamlad genom sina egna sensorer, t.ex. via en kamera), behandlar den och svarar.

AI system är kapabla till att anpassa sitt beteende, till en viss grad, genom att analysera effekterna av tidigare åtgärder, och att arbeta självständigt." (Europaparlamentet - Nyheter 2020).

2.3 Olika typer av AI

Det finns ett nästan oändligt antal mängder av syften som en AI kan ha. Det kan gå från någonting så simpelt som att spela Pong till att navigera trafiken på ett bättre och mer säkert sätt än en människa. När syftena är så pass olika kommer även programmens struktur variera väldigt mycket. Detta innebär att arkitekturen och även språket programmet skrivs i varierar beroende på syftet.

2.3.1 Den moderna AI-utvecklingens tre delmål

Det brukar sägas att Artificiell Intelligens har tre delmål: ANI, AGI och ASI. Artificial Narrow Intelligence är vad som förkortas till ANI, dessa fokuserar på att bemästra ett snävt område, till exempel ett spel eller en specifik arbetsuppgift. De är oerhört kraftfulla och har hittills primärt använts för att dominera spel som Space Invaders och shack. Anledningen till att det har varit just spel som AI har utvecklats för svarar Google Deepminds VD Demis Hassabis på i ett uttalande i The Deepmind

Podcast. Han menar där att artificiella världar som är avskärmade från omvärldens oförutsägbarhet är de perfekta startmiljöerna för utveckling av AI. Men idag har utvecklingen kommit tillräckligt långt för att inte längre vara begränsad av spelens världar. ANI har idag kommit så långt att de börjat utvecklas i form av självstyrande bilar, röstigenkänning och mycket mer. Många AI-utvecklare menar dock att ANI endast är ett steg i riktningen av nästa utmaning: AGI (Tegmark, M 2017).

AGI står för Artificial General Intelligence och är det koncept som många tänker på när de hör termen AI. Man skulle kunna se det som steget efter ANI, det handlar om att AI:n ska, likt en människa, kunna lära sig en stor bredd olika nya saker på egen hand. På grund av att det är en så komplex uppgift som är väldigt långt från den nuvarande förståelsen av AI så är det svårt att riktigt konkretisera delmål. Men de allra flesta ser mänsklig intelligens som det slutgiltiga målet. I sin rapport *Mind* resonerar Alan Turing kring en del av konceptet AGI och ställer sig frågan "Can a machine be linguistically indistinguishable from a human?.". Detta gav upphov till vad som har kommit att kallas för Turing-testet som ska avgöra ifall en maskin kan ge intrycket av mänsklighet (Bringsjord, S 2018). För specifikt det här steget av AI uppkommer det en oenighet mellan Descartes teorier som visades i citatet i 2.1 och faktiska AI-utvecklare. Descartes menade att på grund av hur en maskin agerar olika beroende dispositionen av dess organ så är en generell intelligens omöjlig, att den lär sig något nytt skulle innebära att dess gamla förmågor påverkas drastiskt. Samtidigt finns där en bred konsensus bland de främsta utvecklarna inom teknik- och AI-branscherna att det bara är en tidsfråga innan en generell intelligens uppnås (Tegmark, M 2018).

ASI, Artificial Superintelligence, är det mest okonkreta konceptet inom AI. Det sägs vara steget efter AGI och kommer i grund och botten från spekulativa resonemang om konsekvenserna av en generell intelligens. Hjärnforskaren och filosofen Sam Harris menade i sitt TED Talk (2016) att det finns tre antaganden som tillsammans garanterar att ASI kommer att uppnås. Det första antagandet handlar om att intelligens som vi ser det endast är produkten av bearbetning av information. Med detta menar han att det är hastigheten och effektiviteten i processen som avgör någon eller någots intelligens. Det andra påståendet menar han är att hela resonemanget är beroende av är att vi, mänskligheten, kommer att fortsätta utveckla intelligenta maskiner. Det sista antagandet är till skillnad från de andra ett som vi inte säkert kan svara på, det är antagandet om att vi för tillfället inte är i närheten av toppen av intelligens. Strax innan detta (2014) definierade en annan filosof på Oxford University, Nick Bostrom, i sin bok *Superintelligence: Paths, Dangers, Strategies* termen ASI som:

"any intellect that greatly exceeds the cognitive performance of humans in virtually all domains of interest"

Det är helt enkelt en form av intelligens som har kunnat börja förbättra sig själv, vilket enligt spekulationer ska leda till en explosionsartad utveckling i intelligens. Detta är även varför ASI anses komma ganska kort efter AGI, då en generell intelligens ska kunna hålla på med datavetenskap och därmed kunna ändra på sig själv.

2.3.2 Olika AI-arkitekturer

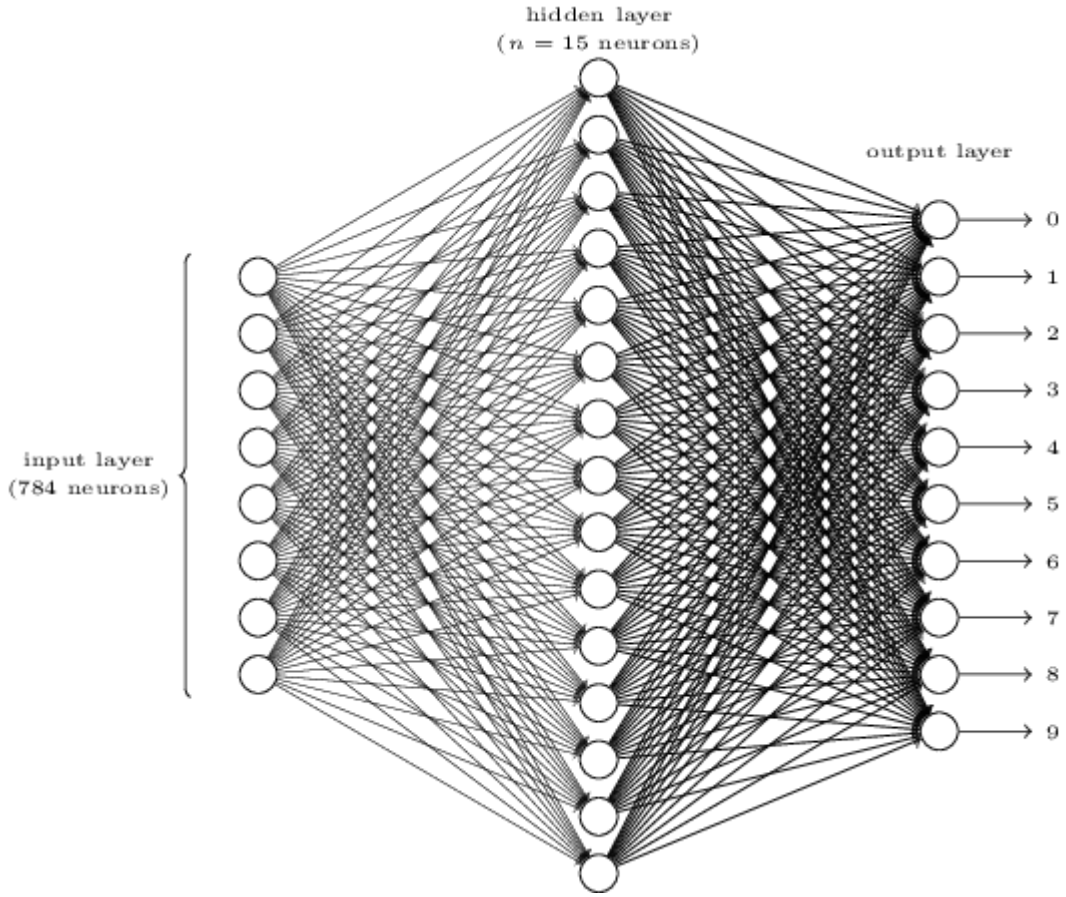
Den "optimala" AI-arkitekturen har länge ansetts vara den som blivit hårdkodad av en människa, alltså en AI vars alla erfarenheter och principer blivit bestämda av någon. Dessa system har ofta väldigt snäva användningsområden där utvecklarna och experter på ämnet har utvecklat metoderna och algoritmerna i decennier. Detta var fallet med IBMs ANI Deep Blue som besegrade den dåvarande världsmästaren Gary Kasparov i schack. På senare tid så har detta däremot kommit att bli motbevisat av så kallade självlärande system. Dessa system har det senaste decenniet tagit världen med storm och visat sig överlägsna de gamla. DeepMinds ANI AlphaZero var banbrytande i sin

förmåga att bemästra Schack, Shogi (japanskt schack) och Go med samma AI (Hassabis et al. 2018). Denna relativt nya arkitektur har tagit inspiration från delar av den mänskliga hjärnan och kallas för Artificial Neural Networks (ANN). På grund av att man vet så lite om dem så är det hos dessa system som det största hoppet om att uppnå AGI ligger.

2.4 Artificiella Neurala Nätverk

Ett artificiellt neuralt nätverk (ANN) är som nämnt ovan gjort för att likt en människa kunna lära sig saker själv. Detta gör teknologin oerhört kraftfull då hur snabbt den lär sig saker endast blir beroende av datorns hastighet. De är inspirerade av den mänskliga hjärnan och använder sig av tre lager med artificiella neuroner för att bearbeta information. Den stora nackdelen med ANN är att de kräver ohyggliga mängder data för att ge förväntade resultat, exempelvis så brukar datan krävd för att få en AI att kunna känna igen handskrivna siffror ligga på åtminstone 50.000 exempel (Nielsen, M 2019).

Första lagret är det som tar in information, och har därför neuroner proportionerligt till den mängd information AI:n kräver. Vanligtvis kallas det andra lagret för det "gömda lagret", vilket är för att informationen som bearbetas inte syns utanför systemets kod. Tredje och sista lagret är det som ger resultat, där antalet neuroner är relativt till hur många svarsalternativ AI:n har. Om den till exempel har fått i uppgift att identifiera siffror brukar den generellt sett ha tio neuroner, en för varje siffra (förekommer även med fyra för ett binärt resultat).



Figur 1: Ett "shallow" Artificial Neural Network designat för att klassificera handskrivna siffror. Det har en neuron för varje möjligt resultat i output-lagret och en neuron för varje inmatning i input-lagret (krympt i det här exemplet för ökad tydlighet).

2.4.1 Hur fungerar ett ANN?

Byggstenarna som tillsammans blir det neurala nätverket kallas för artificiella neuroner, dessa kommer av förtydligande skäl hänvisas till som noder. De kan ses som egna funktioner som tar in ett antal inmatningar och returnerar ett resultat. Resultatet som en nod returnerar brukar kallas för dess aktivering och är ett decimaltal mellan noll och ett. Det är värt att notera att en nods inmatningsvärden består av noderna i det tidigare lagrets aktiveringar, men för ökad tydlighet kommer dessa hänvisas till som inmatningsvärden.

Eftersom att nodens aktivering används av ett större system är ingen sida av svarsspektrat nödvändigtvis bättre än det andra. Nodernas aktiveringar kan ses som representationer av datan som givits, vilket gör att en aktivering kan indikera existensen—eller bristen på den—av någonting AI:n lärt sig leta efter.

Vad som bildar en nods aktivering är i grunden inmatningarna från det föregående lagret. Vart och ett av dessa värden har tilldelats ett så kallat "vikt-värde" som ska representera hur stor påverkan varje inmatning har på systemet. Produkten av vikt-värdet och aktiveringen avgör detta (se ekvation 1).

$$a_i * w_i$$

Ekvation 1: Den "vägda" aktiveringen.

Då varje inmatning ska påverka noden är det sedan summan av alla dessa produkter som blir resultatet av inmatningarna (se ekvation 2).

$$a_0 * w_0 + a_1 * w_1 + \ldots + a_x * w_x = \sum_{i=0}^{x} (a_i * w_i)$$

Ekvation 2: Summan av alla x st kopplingar till noden.

För att kunna avgöra vad som är en bra summa läggs ytterligare en variabel till, ett så kallat "bias". Bias-variabeln adderas till summan och har i uppgift att fungera som ett gränsvärde, där en summa större än noll indikerar en positiv aktivering och vice versa (se ekvation 3).

$$z = \sum_{i=0}^{x} (a_i * w_i) + b$$

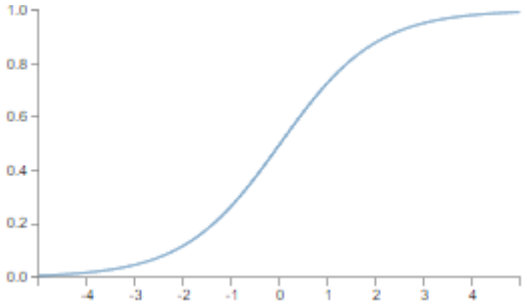
Ekvation 3: Summan av aktiveringarna adderade med ett bias-värde.

Detta är neuronens slutgiltiga resultat, men problemet här är att det inte är begränsat mellan noll och ett. Lösningen är den ickelinjära funktionen som ofta kallas för en sigmoidfunktion eller en logistisk funktion, betecknad σ. Positiva värden som skickas in ger ett värde över 0.5 som går mot ett och negativa värden på *z* ger ett värde under 0.5 som går mot noll.

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

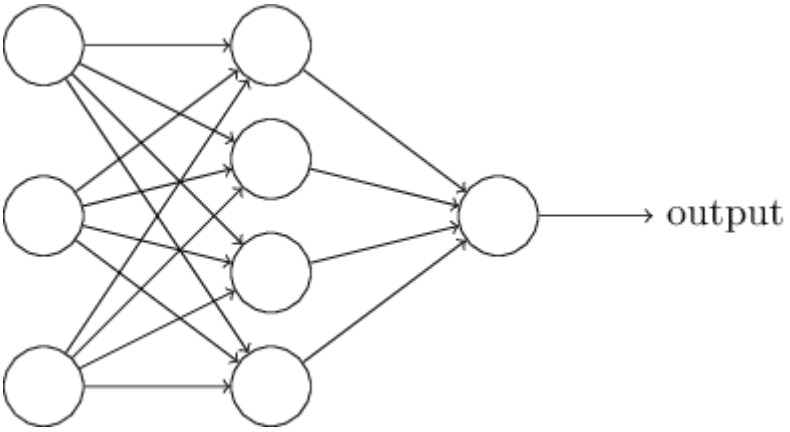
Ekvation 4: Sigmoidfunktion.

Resultatet körs sedan genom funktionen och ett decimalt värde mellan noll och ett returneras (se figur 2). Kurvans släta övergång bidrar till en bättre inlärningskurva och är varför den är så vanligt förekommande inom utvecklingen av artificiella neurala nätverk.



Figur 2: En sigmoidfunktion, även känd som en logistisk funktion.

Detta är alltså hur en ensam nod tar ett beslut baserat på de givna inmatningsvärdena. När de sitter i ett ANN fungerar de likadant och är då kopplade till noderna i intilliggande lager. Om noden i fråga sitter i lagret L så kommer dess inmatningsvärden att vara aktiveringarna i lagret L-1 och dess aktivering kommer att vara en del av inmatningsvärdena i lagret L+1. Detta är med undantag för input-lagret som istället för att ta in tidigare noders aktiveringar bara tar in ett decimaltal mellan noll och ett, vilket kan ses som en färdig aktivering (se figur 3).



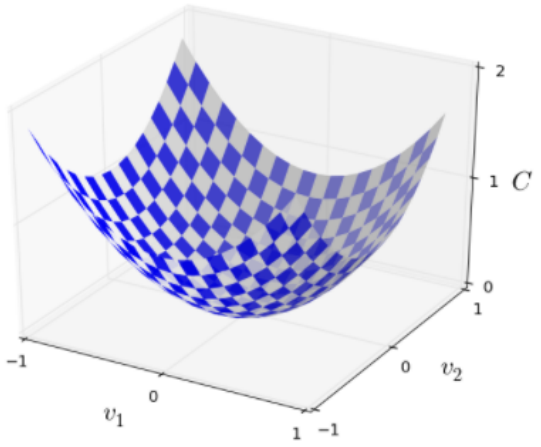
Figur 3: Ett simpelt ANN med tre input-noder och ett gömt lager med fyra noder som returnerar ett output-värde.

Systemet tar på det här sättet sina beslut, men frågan återstår fortfarande om hur det sen kan lära sig av misstagen som oundvikligen kommer att göras. Detta görs med en metod som kallas "Gradient Descent". Metoden utnyttjar faktumet att det föredragna resultatet *y* är känt och räknar med hjälp av det ut vilka förändringar som behöver göras för att komma närmare det. En sådan funktion brukar kallas för en "Cost Function" (*C*) och returnerar ett lägre värde vid ett bättre beslut. För ändamålet valdes en funktion som kallas för "Quadratic Cost Function" (se ekvation 5) och är en mer generell Cost Function som funkar för de flesta syften. För resterande delar av texten kommer den att hänvisas till som *C* eller Cost Function.

$$C(x) = \sum_{i=0}^{i^L} (x - y_i)^2,\ x = \sigma(z_i),\ i^L = |L| - 1$$

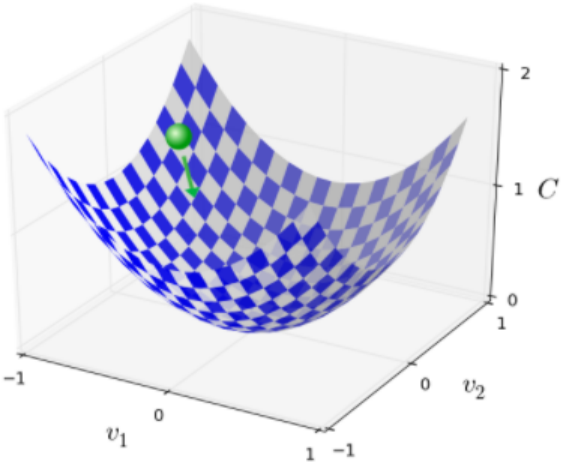
Ekvation 5: Quadratic Cost Function.

Trots att detta är en funktion som ofta är beroende av tusentals värden så visualiseras det bäst med en tredimensionell graf (se figur 4).



Figur 4: Visualisering av Gradient Descent.

Vad som utgör självinlärningen är när systemet försöker röra sig ner för grafen och nå en minimipunkt. Det försöker alltså sänka resultatet på funktionen *C* och därmed öka kvalitén på dess beslut. Processen kan visualiseras med en kula som ligger på grafen och rullar ned till dess lägsta lokala punkt (se figur 5).



Figur 5: Visualisering av ett självlärande system.

För att kunna få kulan att rulla nedför grafen krävs beräkningar för att avgöra vilket håll som leder nedåt. Målet är då att beräkna vilka modifikationer som behöver göras på nätverkets olika vikt- och bias-värden för att bollen ska rulla nedåt i det visuella exemplet. Detta ges av derivatan av funktionen *C* med hänsyn till värdet i fråga, vilken ges efter en applicering av kedjeregeln (se ekvation 6). Då alla variabler i systemet behöver modifieras för att hela nätverket ska bli bättre, är det inte bara bias- och vikt-värdena som blir väsentliga. Även inmatningarna *a* från föregående lager blir viktiga då de kommer att representera en förändring som behöver göras till föregående lagers aktiveringar.

$$\frac{dC}{dx} = \frac{dz_k^L}{dx} * \frac{d\sigma_k^L}{dz_k^L} * \frac{dC}{d\sigma_k^L},\ x \in \{ w_{k,n}^L,\ a_n^{L-1},\ b_k^L \}$$

Ekvation 6: Derivatan av funktionen C med hänsyn till variabeln x.

Ekvation 6 visar att derivatan av funktionen *C* med hänsyn till variabeln *x* ger produkten av de tre funktionernas derivator. Här är det värt att notera att derivatan av funktionen *z* med hänsyn till variabeln *x* varierar beroende på vilken variabel den representerar. Det blir därför en derivata för *w* (se ekvation 7), en för *a* (se ekvation 8) och en för *b* (se ekvation 9).

$$\frac{dz_k^L}{dw_{k,n}^L} = a_n^{L-1}$$

Ekvation 7: Derivatan av funktionen z i nod k, lager L med hänsyn till ett vikt-värde.

$$\frac{dz_k^L}{da_n^{L-1}} = \sum_{i=0}^{i^L} (w_{i,n}),\ i^L = |L| - 1$$

Ekvation 8: Derivatan av funktionen z i nod k, lager L med hänsyn till en aktivering.

$$\frac{dz_k^L}{db_k^L} = 1$$

Ekvation 9: Derivatan av funktionen z i nod k, lager L med hänsyn till ett bias-värde.

Fortsättningen i ekvation 6 är att derivera sigmoidfunktionen med hänsyn till *z*. Även detta genomförs med hjälp av en applicering av kedjeregeln (se ekvation 9).

$$\frac{d\sigma_k^L}{dz_k^L} = \frac{e^x}{(e^x+1)^2},\ x = z_k^L$$

Ekvation 10: En sigmoidfunktions derivata med hänsyn till en funktion z.

Kedjeregeln kommer även till användning vid derivatan av kostnadsfunktionen med hänsyn till sigmoidfunktionen:

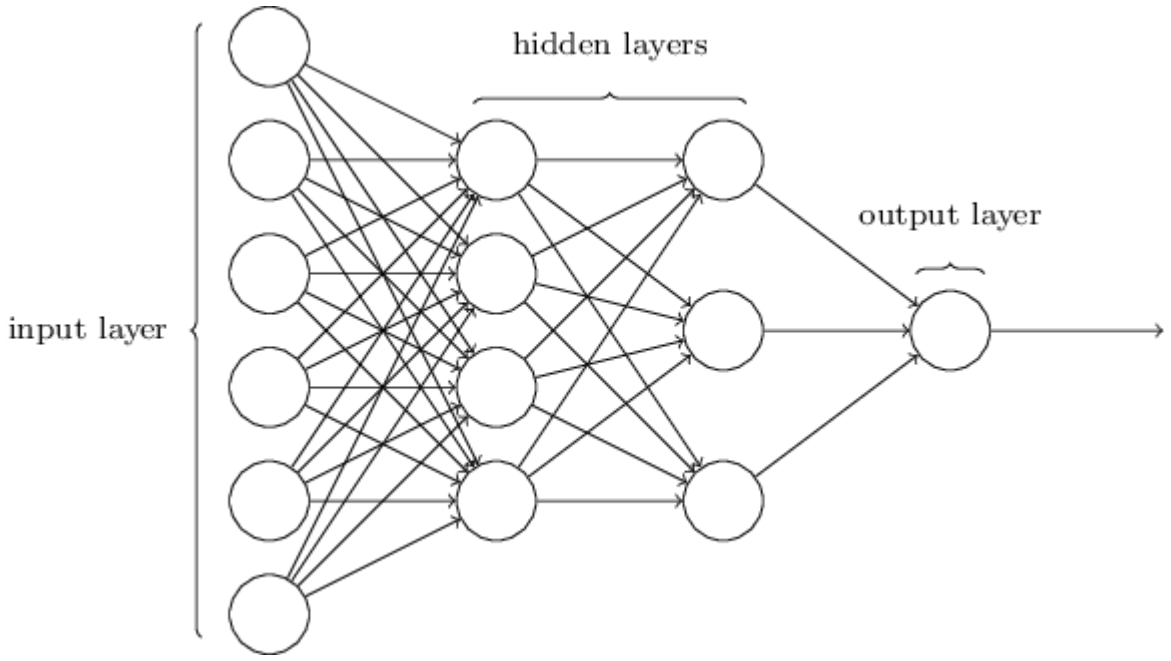
$$\frac{dC}{d\sigma_k^L} = 2 * (x - y),\ x = \sigma(z_k^L)$$

Ekvation 11: Derivatan av kostnadsfunktionen C med hänsyn till en funktion σ.

Processen görs rekursivt för kopplingarna mellan varje lager och ska i optimala fall göras efter varje exempel, efter vilka modifikationernas medelvärden läggs ihop och subtraheras från nätverket. Då det är meningen att kulan i visualiseringen ska rulla nedåt krävs att modifikationerna subtraheras istället för att adderas. Detta kräver dock enorma mängder beräkningar och blir väldigt långsamt för en dator att hantera. Istället används en version där exemplena delas in i så kallade “mini-batches” som kan ses som behållare för en liten andel av exemplena. Nätverket tränas på en mini-batch i taget och får modifikationerna ihoplagda efteråt (Sanderson, G 2017; Nielsen, M 2019). Denna metod kallas för “Stochastic Gradient Descent” och kan visualiseras som att en slalomåkare ersätter kulan i det tidigare exemplet: istället för att långsamt beräkna den kortaste vägen till botten rör sig slalomåkaren snabbt men med stora svängar nedför backen.

2.4.2 Varianter

Till en början var ANN endast en av många metoder att skriva en AI på. Efter att de visade sig vara otroligt effektiva har fler och fler kategorier bildats under termen ANN. Den minst annorlunda varianten kallas för Deep Neural Network (DNN) och har alltid fler än ett gömt lager (se figur 3). På senare tid har detta visat sig mer effektivt än vanliga ANN vilket anses vara på grund av ett DNN:s system att hantera mer komplex data.

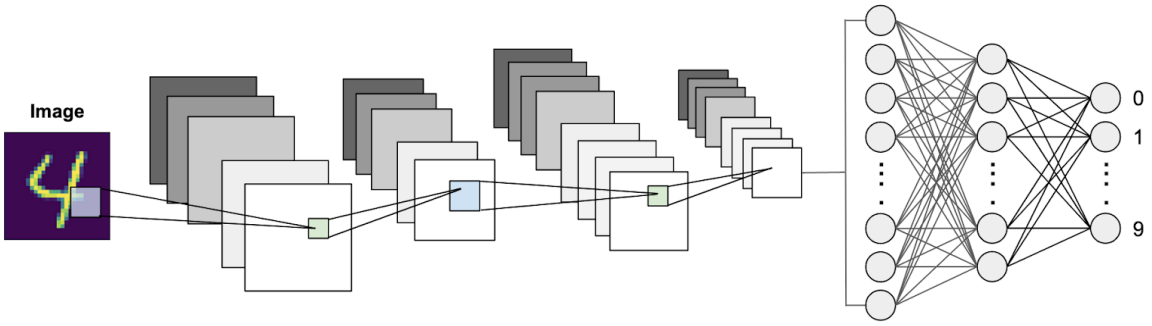


Figur 6: Ett väldigt simpelt Deep Neural Network med två gömda lager.

Både ANN och DNN har gemensamt att information endast förs rakt genom dem, de är så kallade “feed-forward neural networks” som inte har något minne. Detta kan bli begränsande då minne är centralt för att AI:n ska kunna utföra även den simplaste formen av planering. För att lösa detta problemet har så kallade Recurrent Neural Networks (RNN) skapats. Återkoppling eller användandet av minne låter systemen komma ihåg tidigare händelser av vikt. Detta hjälper inte bara dem med planering, utan har även visat sig effektivt på processer som bildkategorisering och röstigenkänning (Dettmers, T 2016).

Den sista varianten av ANN som är viktig nog att nämna är ett så kallat Convolutional Neural Network (CNN). Dessa system använder sig av en process kallad “convolution” som genom

appliceringen av olika filter simplifierar bilder flera gånger för att kunna ta ett beslut (se figur 7). De är otroligt bra på att identifiera vad de ska leta efter, är det till exempel vanlig bildidentifikation lär den sig snabbt att det är kanter och föremålets form som avgör saken. Inte bara ska de här maskinerna vara väldigt bra på att hantera bilder och tal, utan de används även mycket inom robotteknik och självkörande bilar (Dettmers, T 2015).



Figur 7: Exempel på ett Convolutional Neural Network med syfte att känna igen handskrivna siffror.

2.4.3 Potentialen hos ANN

Som nämnt tidigare var 2010-talet decenniet då de artificiella neurala nätverken hamnade i rampljuset på grund av sina många framsteg. Trots dessa fanns där problemet att majoriteten av vad som åstadkommits skett i spelvärldar. Skulle AI någonsin röra sig från spelens artificiella världar till verkligheten? Sent år 2020 fick Google Deepmind resultatet från det årets CASP (Critical Assessment of protein Structure Prediction) att deras AI, AlphaFold, hade uppnått en över 90% pricksäkerhet i förutsägning av proteinstrukturer (Deepmind 2020). Proteiner är kedjor av aminosyror som är nödvändiga för i princip allt liv. Dessa kedjors tredimensionella struktur avgör deras funktioner, vilket gjort det till ett centralt problem inom biologin. Faktumet att ett ANN löst ett problem så långt ifrån spelens värld pekar på en stor potential hos systemen.

Nvidia är en av många utvecklare av självkörande bilar och pekar på vikten av olika typer av ANN i dessa system. I ett inlägg på deras blogg skriver Automotive Content Marketing Manager Katie Burke att deras system är beroende av användandet av flera olika typer av ANN för att skapa en mer generell expertis. Exempelvis har de separata ANN för småsaker som igenkänning av trafikljus och identifiering av vägskyltar (Burke, K 2019). Även inom andra kategorier som taligenkänning, robotmotorik och språköversättning utforskas just nu möjligheterna för liknande användning av ANN.

3. Problemformulering

Igenkänning av text och bilder är någonting som vi människor gör åtminstone hundratusentals gånger om dagen utan att tänka på det. Men när den här förståelsen ska återskapas så att en AI kan göra detsamma blir det hela utmanande. Det visar sig att en förståelse för koncept som text och bilder är svårt att återskapa för en maskin som inte har några förkunskaper utanför de som givits den. En väg runt detta problemet och ett sätt att undvika den annars oundvikliga filosofiska aspekten är att använda teknologi som ANN. Dessa systems förmåga att genom otaliga iterationer lära sig saker är avgörande för att kunna uppnå eftersökta resultat på en rimlig tid. Det är ju bra och så, men nu uppstår ett helt nytt problem: hur utvecklas ett artificiellt neuralt nätverk? Bara ett simpelt ANN utformat för detta syfte har sällan under hundra noder och är beroende av stora mängder så kallade “weights” och “biases”. Detta lägger grunden till ett komplext nätverk av kopplingar som tillsammans ska kunna lära sig ta vettiga beslut. Därför kommer det huvudsakliga problemet vara komplexiteten av AI:n som ska byggas.

Forskaren Michael Nielsens bok *Neural networks and deep learning* handlar om neurala nätverk och förklarar konceptet “Deep learning”. Det löpande exemplet genom texten är hanteringen av utmaningen att få en AI att lära sig känna igen handskrivna siffror. Exemplet kommer att vara projektets utgångspunkt och stegen Nielsen tar kommer att vara grunden för den AI som kommer utvecklas i det här projektet.

I stort så är lösningen av ett ändå relativt grundläggande problem nödvändigt för att vidare utveckling ska kunna ske inom området. Bara i denna simpla form finns det många användningsområden inom automatisering, där mycket tid kan sparas på datorer som kan läsa in siffror. Nästa steg är skulle kunna ses som bildigenkänning, då många liknelser kan dras mellan de båda processerna. Det är i lösningen på detta problemet som den riktiga potentialen hos AI visar sig. Detta är nämligen ett tydligt steg mot att kunna simulera våra sinnen, vilket kan ses som ett nödvändigt steg för att någon gång uppnå AGI som fungerar likt oss.

3.1 Frågeställning

Hur utvecklas ett DNN för att kunna lösa verkliga problem och hur kan det användas som hjälpmedel i vardagen?

3.2 Metodbeskrivning

Målsättningen med AI:n är att den med relativt hög pricksäkerhet ska kunna känna igen handskrivna siffror, med en nedre gräns på 80% korrekta svar på testdatan. För att uppnå detta kommer fokus läggas på den faktiska träningen. Efter att rätt struktur för AI:n implementerats är det optimeringen av antalet lager och noder som ska hamna i fokus. Detta är för att det i teorin finns en optimal uppsättning beståndsdelar för specifikt detta problemet. Någonting som gör det nödvändigt att åtminstone finna en relativt optimal uppsättning för att en accepterad pricksäkerhet ska kunna uppnås.

3.2.1 Implementeringen av AI:n

Ett DNN är verktyget som kommer att användas för att besvara frågeställningen. Som nämnt i kapitel 2.4.2 så är egentligen både CNN och RNN bättre alternativ för syftet. Men på grund av hur de endast bygger vidare på komplexiteten i ett vanligt ANN / DNN så får här ske en avgränsning. Studier säger även att trots faktumet att ett DNN är suboptimalt för uppgiften så har tidigare nätverk uppnått mer än 95% pricksäkerhet om det optimeras rätt (Yann LeCun, 2021).

När AI:n skrivs blir startpunkten att implementera strukturen och sätta upp de listor med värden som kommer koppla ihop systemets lager. Därefter ska det ske en implementering av matrismultiplikation som ska ha i uppgift att föra datan framåt genom lagrena. Med detta så kommer AI:n att kunna ta egna beslut, men inte kunna lära sig av sina misstag. Det är viktigt att själva systemet görs först, innan självinlärningen, då den behöver kunna ta beslut för att kunna lära sig från dem. Självinlärningen är det sista och mest utmanande steget i dess design. Genom att köra systemet i motsatt riktning - med resultatlagret som inmatning och inmatningslagret som resultat - kan det lära sig vad som gjordes fel och hur dess värden kan förändras för att förbättra resultatet.

3.2.2 Träning och optimering

I det här delmomentet kommer det färdiga ANN att tränas på den tillgängliga datan. Datan kommer från “MNIST handwritten digit database” som är en öppen databas för handskrivna siffror i upplösningen 28x28 och kommer ihopparade med ett svar på vilken siffra de föreställer. Den har ett träningsset på 60.000 exempel och ett testset på 10.000 exempel som båda kommer vara mycket användbara för att träna det neurala nätverket.

Efteråt kommer AI:n att värderas efter sin pricksäkerhet, alltså antal gånger den haft rätt visat i procent. Efteråt görs optimeringar på dess struktur i syfte att ge bättre igenkänningsförmåga. Dessa görs på de gömda lagren och kommer främst vara mindre modifikationer som tillägget eller borttagandet av en artificiell neuron, men kan även inkludera förändringar i antalet gömda lager. De yttre lagren blir orörda då en förändring i dem hade blivit en förändring i systemets hela funktion.

4. Genomförande

4.1 Implementering

4.1.1 Struktur

Alla typer av artificiella neurala nätverk kan i grund och botten ses som en samling vikt- och bias-värden (se kapitel 2.4). Systemet består alltså av en mängd vektorer och matriser som innehåller variablerna för dessa värden. När dessa implementeras är det mycket viktigt att de hålls ordnade, då minsta lilla felindexering leder till fullständigt irrelevanta resultat. Mellan två lager finns där lika många vikt-värden som produkten av de två lagrens nodantal. Detta placeras i en matris där y-axeln är nodantalet i det tidigare lagret *L-1* och där x-axeln är nodantalet av det mottagande lagret *L* (se tabell 1).

$$\begin{bmatrix} w^{L}_{0,0} & w^{L}_{0,1} & \cdots & w^{L}_{0,n} \\ w^{L}_{1,0} & w^{L}_{1,1} & \cdots & w^{L}_{1,n} \\ \cdots & \cdots & \cdots & \cdots \\ w^{L}_{k,0} & w^{L}_{k,1} & \cdots & w^{L}_{k,n} \end{bmatrix}$$

Tabell 1: Matris som innehåller vikt-värdena mellan ett lager L-1 och ett lager L.

Antalet bias-värden mellan de båda lagren är betydligt färre då antalet är samma som nodantalet i lagret *L* (se tabell 2). Detta är på grund av att varje nod blivit tilldelad ett bias-värde som adderas till summan av vikt-värdena och inmatningsvärdena (se kapitel 2.4.1).

$$\begin{bmatrix} b^{L}_{0} \\ b^{L}_{1} \\ \cdots \\ b^{L}_{k} \end{bmatrix}$$

Tabell 2: Vektor som innehåller bias-värdena mellan ett lager L-1 och ett lager L.

En matris och en vektor förvarar datan mellan varje lager, vilket innebär att antalet par är antal lager minus ett. När ett lager aktiveras är det första som sker en multiplikation av varje vikt-värde $w_{k,n}$ och ett inmatningsvärde a_{n} (se tabell 3).

$$\begin{bmatrix} w^{L}_{0,0} * a^{L-1}_{0} & w^{L}_{0,1} * a^{L-} & \cdots & w^{L}_{0,n} * a^{L-}_{n} \\ w^{L}_{1,0} * a^{L-1}_{0} & w^{L}_{1,1} * a^{L-} & \cdots & w^{L}_{1,n} * a^{L-}_{n} \\ \cdots & \cdots & \cdots & \cdots \\ w^{L}_{k,0} * a^{L-1}_{0} & w^{L}_{k,1} * a^{L-} & \cdots & w^{L}_{k,n} * a^{L-}_{n} \end{bmatrix}$$

Tabell 3: Matris som innehåller de värderade aktiveringsvärdena (se ekvation 1, kapitel 4.2.1)

De vägda inmatningarna för alla *k* stycken noder i lagret *L* läggs ihop och bildar en vektor bestående av summorna (se tabell 4).

$$\begin{bmatrix} \sum_{i=0}^{i^{L-1}} (w^{L}_{0,i} * a^{L-1}_{i}),\ i^{L-1} = |L-1| \text{ -} \\ \sum_{i=0}^{i^{L-1}} (w^{L}_{1,i} * a^{L-1}_{i}),\ i^{L-1} = |L-1| \text{ -} \\ \cdots \\ \sum_{i=0}^{i^{L-1}} (w^{L}_{k,i} * a^{L-1}_{i}),\ i^{L-1} = |L-1| \text{ -} \end{bmatrix}$$

Tabell 4: Alla k stycken noders vägda inmatningar summerade var för sig.

Denna vektor adderas med bias-vektorn för att bilda en ny vektor av samma storlek, men som kan skrivas om till att innehålla variabeln *z* enligt ekvation 3 (se tabell 5).

$$\begin{bmatrix} z^{L}_{0} \\ z^{L}_{1} \\ \cdots \end{bmatrix}$$

$$\begin{bmatrix} z_k^L \end{bmatrix}$$

Tabell 5: Summan av de vägda aktiveringarna för noderna adderas med ett bias-värde och bildar variabeln z enligt ekvation 3.

För att få fram en vektor av aktiveringsvärden krävs att varje tal i vektorn körs genom sigmoidfunktionen. Efter detta ges den så kallade aktiveringsvektorn som är av samma storlek och vars värden är decimaltal mellan noll och ett. Aktiveringsvektorn blir för lagret efteråt inmatningsvektorn och samma process upprepas tills att det sista lagret har nåtts. Vid det sista lagret blir aktiveringsvektorn systemets "svar" på den data som matades in i input-lagret.

4.1.2 Självinlärning

Självinlärningen är en process där nätverket går igenom sina variabler för att ta reda på vilka som behöver förändras för att ge bättre svar. I motsats till aktiveringarna—som kör igenom systemets variabler från början till slut—görs inlärningsprocessen baklänges. Den sker rekursivt genom var lager och går först igenom vad som gick fel i aktiveringarna av det föregående lagret *L*, varpå kopplingarna från lagret *L-1* värderas och modifieras. Processen görs för en nod i taget. Vikt-värdena är de första att modifieras och är till antal lika många som nodantalet i lagret *L-1*. Enligt ekvation 6 beräknas först känsligheten för en vikt som sedan görs negativ för att värdet av funktionen *C* ska gå ner för de inmatningsvärdena (se ekvation 12).

$$-\frac{dC}{dw_{k,n}^L} = -a_n^{L-1} * \frac{e^{z_k^L}}{(e^{z_k^L}+1)^2} * 2 * (\sigma(z_k^L) - y)$$

Ekvation 12: Derivatan av den negativa kostnadsfunktionen med hänsyn till vikt-värdet av index k och n.

Bias-värdena är färre till antalet och blir totalt lika många till antal som noderna i lagret *L*. Enligt ekvation 6 så sker endast en förändring i den första faktorn, där *x* i derivatan av *x* med hänsyn till *z* blir till *b* (se ekvation 13).

$$-\frac{dC}{db_k} = -1 * \frac{e^{z_k^L}}{(e^{z_k^L}+1)^2} * 2 * (\sigma(z_k^L) - y)$$

Ekvation 13: Derivatan av den negativa kostnadsfunktionen med hänsyn till bias-värdet av index k.

Här uppstår ett problem efter första iterationen då vektorn *y* endast är anpassad för sista lagret *L*. För att lösa detta används derivatan av kostnadsfunktionen med hänsyn till inmatningsvärdet, alltså aktiveringen från det föregående lagret (se ekvation 14).

$$-\frac{dC}{da_n} = -\sum_{i=0}^{i^L}(w_{i,n}^L) * \frac{e^{z_k^L}}{(e^{z_k^L}+1)^2} * 2 * (\sigma(z_k^L) - y),\ i^L = |L| - 1$$

Ekvation 14: Derivatan av den negativa kostnadsfunktionen med hänsyn till den föregående inmatningen av index n.

Varje nod i lagret *L* ger genom denna ekvationen sin bild av hur aktiveringarna i det föregående lagret *L-1* borde förändras. Dessa föreslagna modifikationer adderas sedan för varje nod i lagret *L-1* och ger den nya vektorn y^{L-1} (se tabell 6).

$$\begin{bmatrix} -\frac{dC}{da_0^{L-1}} \\ -\frac{dC}{da_1^{L-1}} \\ \cdots \\ -\frac{dC}{da_n^{L-1}} \end{bmatrix}$$

Tabell 6: Vektorn y^{L-1}.

Detta tillåter systemet att fortsätta använda kostnadsfunktionen genom hela inlärningen, då den förnyas efter varje rekursion. När input-lagret är nått ges alla systemets föreslagna modifikationer som tillsammans med det ursprungliga nätverket kan bilda ett (oftast) bättre nätverk.

4.2 Träning

4.2.1 Inlärningsprocess

Som tidigare nämnt så är det optimalt att ta medelvärdet från summan av de föreslagna modifikationerna från varje träningsexempel. Men på grund av dess långsamhet görs samma sak istället för en grupp som består av en bråkdel av träningsexemplena, en så kallad "mini-batch". Då tränas systemet på en mini-batch i taget och lär sig istället stegvis. Detta ökar snabbheten på algoritmen och ger rum för generering av mer data.

4.2.2 Optimering

Det finns ett antal olika delar i ett DNN som alla har sin påverkan på nätverkets prestation. Vid optimeringsfasen är det viktigt att dessa delar hittas och optimeras. Därför kommer tester göras på alla de större delarna av nätverket, vilka är följande fem:

- Antalet noder i de gömda lagren.
- De framslumpade startvärdena.
- Antal gömda lager.
- Antal träningsobjekt i varje mini-batch.
- Hur många iterationer nätverket får på sig att träna på träningsdatan.

För att ta reda på betydelsen av alla dessa delarna tränas ett antal nätverk för varje test där delen i fråga förändras men inte de andra.

5. Resultat

5.1 Data

Test 1: Slumpade startvärden

Seed	Prestation
“bxreeo”	82%
“cnythe”	91%
“grnvgv”	90%
“gxtypl”	74%
“mycqhf”	91%
“ppwfdm”	91%
“rkwyum”	91%
“tpcgtp”	92%
“ukcalp”	91%
“xiqbuo”	90%

Tabell 1: Resultaten från tränade nätverk med startvärden från olika “seeds”.

Test 2: Noder

Nätverksstruktur	Prestation
784-3-3-10	53%
784-6-6-10	77%
784-9-9-10	87%
784-12-12-10	73%
784-15-15-10	91%
784-18-18-10	83%
784-21-21-10	92%
784-24-24-10	93%
784-27-27-10	85%
784-30-30-10	93%

Tabell 2: Resultaten från tränade nätverk med varierande antal noder.

Test 3: Gömda lager

Antal gömda lager	Prestation
1	83%
2	83%
3	10%
4	90%
5	80%

Tabell 3: Resultaten från tränade nätverk med varierande antal gömda lager.

Test 4: Storlek på mini-batch

Storlek	Prestation
20	92%
40	84%
60	83%
80	83%
100	83%

Tabell 4: Resultaten från nätverk tränade med mini-batches av varierande storlek.

Test 5: Iterationer

Iterationer	Prestation
1	66%
2	78%
3	80%
4	81%
5	82%
10	83%
15	84%
20	84%
21	83%
22	83%
23	83%

24	83%
25	92%

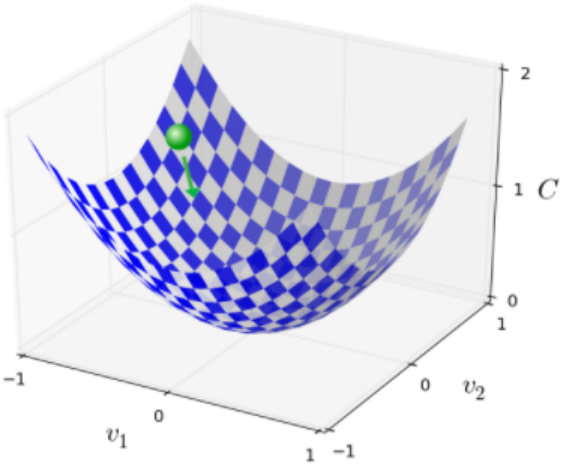
Tabell 5: Resultaten från nätverk tränade efter olika antal iterationer, notera att iterationer två till fyra och 21 till 24 lagts till av relevansskäl.

5.2 Analys

5.2.1 Test 1: Slumpade startvärden

Den överväldigande majoriteten av startvärdena gav i slutändan resultat med över 90% pricksäkerhet. Detta var förvånande då tidigare testkörningar av systemet pekade på att startvärdena skulle ha en massiv påverkan på systemets prestation. Avvikelser uppstod dock fortfarande i detta testet men skilde som mest 18 procentenheter mellan de bäst- och sämstpresterande nätverken. Det tyder på att de slumpade startvärdena—åtminstone vid ett litet antal iterationer som tio—har en stor påverkan på ett tränat systems slutgiltiga prestation.

Detta problemet är ganska vanligt förekommande och kan tydligt visualiseras bland annat med hjälpa av exemplet i kapitel 2.4.1. Ifall gropen som kulan rullar ned i (se figur 8) befinner sig på toppen av en stor kulle kommer inte nätverket att ta hänsyn till det och istället ta sig direkt till gropens botten. Detta kallas för en “lokal minimipunkt”. Då kostnadsfunktionen C egentligen består av tusentals värden är det sällan som dessa lokala minimipunkter märks av med mer än några procent som värst. Det händer dock att de slumpade startvärdena är så dåliga att systemet fastnar direkt och därmed inte längre lär sig under träningsfasen.



Figur 8: Figur 5 från kapitel 2.4.1.

5.2.2 Test 2: Noder

I andra testet observerades en tydlig trend där systemets prestation ökade med antalet noder. Detta var väntat då fler noder i ett system antogs resultera i en bättre förmåga att hantera komplexitet. I testet sjönk aldrig ett nätverks pricksäkerhet under 80% när det hade fler än 12 noder. Den högsta pricksäkerheten uppmättes på systemen med 24 och 30 noder, som båda nådde ungefär 93%. Däremot fanns där undantag i trenden då nodantal som 18 och 27 nästan sjönk med 10% vardera i pricksäkerhet i jämförelse med sina föregångare. Detta antogs bero på att dåliga lokala minimipunkter hade nåtts och hindrat nätverket från att lära sig mer. Men på grund av den ändå relativt lilla

skillnaden kan det inte säkerställas. Trots det är det dock tydligt att ett större antal noder leder till en bättre prestation från systemets sida.

5.2.3 Test 3: Gömda lager

Av samma princip som i hypotesen för test två antogs ett större antal gömda lager resultera i en bättre förmåga att hantera komplexitet. Detta stämde till viss del på resultatet då fyra gömda lager gav betydligt bättre pricksäkerhet än ett, två eller tre lager. Dessvärre så visade sig detta inte stämma vid tre gömda lager då detta resulterade i en prestation på ungefär 10%, vilket kan jämföras med att gissa på samma alternativ genom hela testet. Det observerades sedan från resultatet från varje iteration av inlärningen att nätverket fastnat på 10% pricksäkerhet. Detta gjorde det tydligt att det dåliga resultatet med högsta sannolikhet berodde på att en dålig lokal minimipunkt stötts på. Även nätverket med fem gömda lager avvek från hypotesen med en oväntad minskning på tio procentenheter i jämförelse med nätverket med fyra gömda lager. Detta gav testet ett tvetydigt resultat som antingen pekade på dåliga lokala minimipunkter som tog bort möjligheterna för ett rimligt resultat eller på att fyra lager är det optimala.

5.2.4 Test 4: Storlek på mini-batch

För det här testet var hypotesen att nätverkens prestationer skulle förbättras i takt med att varje mini-batch blev större. Detta var baserat på idén om att större mini-batches är bättre än mindre på grund av att de är mer lika vanlig Gradient Descent. Detta visade sig vara ett falskt antagande då den minsta storleken gav den bästa prestationen. Sedan följdes en tydlig trend där resultatet försämrades i takt med att storleken blev större. Någonting som tydde på att mini-batches—när de väl används i inlärningsprocessen—genererar bäst resultat vid mindre storlekar.

5.2.5 Test 5: Iterationer

Mer iterationer antogs leda till en bättre prestation från nätverkets sida då det även kan appliceras på en människas inlärning, där mer övningstillfällen tenderar att leda till bättre prestation i slutändan. Resultatet följde detta antagandet och systemet blev långsamt bättre i takt med att det fick fler tillfällen på sig att öva. Efter iteration 20 skedde dock en oväntad avvikelse där systemets prestation gick ned med en procentenhet. Systemets inlärning verkade då ta stopp för de kommande fyra iterationerna fram till iteration 25 då trenden gjorde ytterligare en oväntad vändning. Denna förbättrade systemet drastiskt med nio procentenheter, vilket placerade nätverket bland de bästa från alla testen. Förändringar i prestation av denna storleken förekom vanligtvis annars i de fem första iterationerna av nätverkets träning. Detta var en massiv indikation på att mer iterationer leder till ett bättre nätverk. Däremot noterades det även att ett stopp vid iteration 24 hade lett till ett sämre resultat än tidiga iteration 15, någonting som pekar på vikten i att varje iteration testas för att en tydlig helhetsbild ska kunna ges. Då kan en kvalificerad gissning göras om huruvida nätverket borde tränas för fler iterationer.

5.3 Slutsatser

Frågeställningen lyder: *"Hur utvecklas ett DNN för att kunna lösa verkliga problem och hur kan det användas som hjälpmedel i vardagen?"*

Första delen av frågeställningen besvaras utan svårighet, det visar sig nämligen att ett DNN kan tränas till att ge en betydelsefull aktivering så länge inmatningsvärdena och svarsalternativen presenteras på ett effektivt sätt. Vad som gjordes i det valda exemplet med sifferigenkänning var att den inmatade siffran krymptes ned till en vektor bestående av decimala gråskalvärden mellan noll och ett för varje pixel som blev systemets inmatningsvärden. Nätverkets aktiveringar var tio noder, där en representerar varje svarsalternativ. För att ett DNN ska ha möjligheten att lösa verkliga problem måste datan—både från inmatningen och i aktiveringen—alltså kunna simplifieras till en vektor bestående av tal mellan ett och noll. Detta ändrar problemet från hur en AI ska *designas* för ett verkligt problem till hur det verkliga problemet ska *formuleras* för att den ska förstå.

För att ett DNN ska kunna användas som ett hjälpmedel i vardagen måste det effektivt kunna lösa de problem det ställs inför. Om det ska finnas en anledning för dem att användas måste de prestera så bra och effektivt att de blir att föredra över mänskliga konkurrenter. De tester som genomförts på sifferexemplet pekar på att vissa större delar i ett DNN bör hålla sig till en viss struktur. För att undvika det vanligt förekommande problemet med dåliga lokala minimipunkter pekar resultaten bland annat på att ett större antal iterationer av träning på träningsdatan generellt sett gynnar nätverkets prestanda. Även ett större antal noder och möjligen även lager verkar ge bättre nätverk i slutändan. I träningsfasen verkar även en mindre storlek på mini-batches gynna nätverkets slutgiltiga prestanda.

6. Avslutande Diskussion

6.1 Sammanfattning

Bildklassificering är något som de flesta människor gör åtminstone flera hundratusentals gånger dagligen. Trots det är metoden som används av hjärnan för att åstadkomma denna otroliga klassificeringsförmåga väldigt svår att konkretisera. För att undvika utmaningen att formulera denna metod för en dator utan förkunskaper används Artificiella Neurala Nätverk (ANN). En mer kraftfull variant av ANN, Djupa Neurala Nätverk, består av en mängd olika delar vars parametrar tillsammans har en stor påverkan på resultatet. För att göra denna AI-arkitektur till en rimlig kandidat för att lösa stora, verkliga problem krävs det att dessa delar utvecklas rätt och att parametrarna är optimala för uppgiften. Efter dessa krav definieras arbetets frågeställning som:

"Hur utvecklas ett DNN för att kunna lösa verkliga problem och hur kan det användas som hjälpmedel i vardagen?"

Metoden som används för att uppnå detta är en implementeringsfas där systemet skapas och en optimeringsfas där de olika delarnas påverkan på nätverkets prestation testas. I första fasen formuleras koncepten och ekvationerna som introducerades i kapitel 2.4.1 om för att få de använda matematiska operationerna att fungera i vektor- och matrisform, en uppgift där indexering blir väldigt centralt. För att ta reda på betydelsen av nätverksdelarna i den senare fasen genomförs ett antal test. En bestämd mängd nätverk tränas för vart och ett av testen där delen som testas förändras men inte de andra. Resultaten på dessa test förväntas ge en bild av hur viktiga de testade delarna är och ifall någon dels påverkan är obetydlig nog att den kan försummas.

Testernas resultat gav en tydlig bild av arkitekturens potential inom specifikt bildklassificering. Det bekräftade att större nätverk presterar bättre än mindre i majoriteten av fallen. Dock så var resultatet mer oväntat vad gäller träningen av nätverket. Vid användning av Stochastic Gradient Descent tydde resultatet på att mindre storlekar på mini-batches gynnade systemets prestation i slutändan. Det pekade även på att nätverkets startvärden har betydligt mindre betydelse när det får mer tillfällen på sig att träna på träningsdatan.

6.2 Diskussion

Det huvudsakliga problemet med resultatet är att det ur ett kvantitativt perspektiv är i minsta laget för att jag med säkerhet ska kunna dra några helt säkra slutsatser. Det ger dock fortfarande vaga riktlinjer för hur de olika delarna fungerar och påverkar nätverket. Anledningen till denna brist är att det krävdes större mängder datorkraft än förväntat att få tag i resultatdatan. Detta innebar att mängden data genererad på den givna tiden blev mindre än väntat. Med en bättre bild av den krävda datorkraften hade antalet iterationer för alla tester åtminstone kunnat dubblas. I testerna visade det sig att dåliga lokala minimipunkter motverkas av fler iterationer, någonting som hade ökat resultatets trovärdighet drastiskt. Det hade även hjälpt om varje test gjorts för åtminstone två olika seeds för startvärden, till skillnad från det enda seed som faktiskt användes i testen (med undantag för seed-testet). Dock är det värt att notera att tillägget av ett nytt seed och en dubbling i iterationer till alla tester hade resulterat i att tiden för testerna ökade med en faktor av fyra.

Trots resultatets brister kvarstår faktumet att vårt DNN vid sina ställen i testen uppnådde över 90% pricksäkerhet på testdatan. Efter att det bekräftats att DNN kan tränas för sifferigenkänning anser jag att det inte är orimligt att de även kan användas inom andra områden. Bara inom bildigenkänning

öppnas en bred mängd alternativ som arkitekturen i så fall kan appliceras på. Jag tror dock endast att dessa system kommer att ta upp en liten del av programmet då de förmodligen fungerar bäst tillsammans med andra algoritmer. Om man exempelvis hade applicerat denna teknologin på de maskiner och processer som vi har automatiserat idag är det inte omöjligt att detta hade lett till stora framsteg inom områdena. Ett område som använder sig av en sådan här blandning av teknologier är självkörande bilar. Företaget Nvidia forskar bland annat inom detta området och uttrycker sig i ett av sina blogginlägg att de använder sig av flera olika typer av ANN för att uppnå en bra körförmåga (Burke 2019).

6.3 Framtida arbete

Givetvis så hade planen för en kortsiktig fortsättning på examensarbetet inkluderat tillägget av fler iterationer för varje test. Förmodligen hade de ökats till runt 25 för de vanliga testen och kanske täckt tiotalen mellan noll och hundra för iterationstesten. Utöver det hade algoritmen i sig setts över och tillägg hade gjorts på de ställen där den för tillfället inte är mycket mer än grundläggande. Bland annat hade en så kallad “learning rate”—en faktor till inlärningens modifikationsvärden—lagts till för att ge möjlighet att experimentera med inlärningshastigheten. Kostnadsfunktionen hade även blivit ersatt av en så kallad “Cross Entropy cost function” och sigmoidfunktionen av en så kallad “Softmax function”. Dessa två är mer anpassade än de tidigare för nätverkets syfte och hade därmed uppnått bättre resultat samt snabbare inlärning. Förutom dessa mindre modifikationer hade även systemets helhet setts över i syfte att uppnå en mer effektiv kod som potentiellt kunnat minska kravet på datorkraft. Målet hade alltså varit att bättra nätverkens prestationer samt bredda tillgången på data i resultatet.

Vid en utveckling av examensarbetet hade AI:n kunnat förbättras drastiskt. Först och främst hade CNN-arkitektur ersatt den simpla DNN-arkitekturen. Eftersom den är anpassad för att hantera tvådimensionella bilder har den potentialen att uppnå betydligt högre precisioner inom sifferklassificering, där en av de bästa systemen hittills lyckats uppnå 99.61% (LeCun et al. 1998). Mer tid hade även låtit mig bli mer bekant med nätverket och dess upplärning vilket hade lett till en bättre tidsplan för datagenereringen. I stort hade målsättningen varit att uppnå en betydligt bättre pricksäkerhet för nätverken samt se till så att det skulle finnas nog med relevant data för att jag skulle kunna dra säkra slutsatser.

Ifall arbetet blivit finansierat skulle det kunnat breddas betydligt. Utöver sifferigenkänning hade projektet också fått möjligheten att innefatta bildklassificering av exempelvis trafikljus eller vägskyltar. Finansieringen hade låtit mig hyra datorkraft i stil med Google Clouds “Cloud GPU”-tjänst. På grund av detta stora lyft i datorkraft skulle resultatet få en mycket bättre bredd som hade tillåtit mig dra säkra slutsatser om systemens olika delar. Det finns ganska mycket träningsdata tillgänglig på internet men i det fallet att jag inte hittar data av den kvalitén jag söker hade Amazons “Mechanical Turk”-tjänst kunnat vara till nytta. Det är ett ganska enkelt sätt att få tag i märkt data då den går ut på att man betalar någon annan för att göra datamärkningen åt en. Med en mängd olika system med olika syften och resultat hade jag kunnat dra en slutsats utifrån ett mycket bredare perspektiv.

Källförteckning

Källor:

Tegmark, M (2018). *Sommar & Vinter i P1: Max Tegmark.*
https://sverigesradio.se/avsnitt/1077305 [2021-04-05].

Nielsen, M (2019). *Neural Networks and Deep Learning.*
http://neuralnetworksanddeeplearning.com/ [2021-03-27].

Europaparlamentet - Nyheter (2021). *Vad är artificiell intelligens och hur används det?*.
https://www.europarl.europa.eu/news/sv/headlines/society/20200827STO85804/vad-ar-artificiell-intelligens-och-hur-anvands-det [2020-11-10].

Bringsjord. S (2018). *Artificial Intelligence.*
https://plato.stanford.edu/entries/artificial-intelligence/#HistA [2021-02-16].
LeCun et al (1998). *The MNIST database of handwritten digits.*

http://yann.lecun.com/exdb/mnist/ [2021-03-26].

Dettmers, T (2016). *Deep Learning in a Nutshell: Sequence Learning*.

https://developer.nvidia.com/blog/deep-learning-nutshell-sequence-learning/ [2021-02-16].

Dettmers, T (2015). *Deep Learning in a Nutshell: Core Concepts*.

https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/ [2021-02-16].

Hassabis, D. Schrittwieser, J. Hubert, T. Silver, D (2018). *AlphaZero: Shedding new light on chess, shogi, and Go.*

https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go [2021-04-05].

Sanderson, G (2018). *Neural networks.*

https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

[2021-04-03].

Deepmind (2020). *AlphaFold: a solution to a 50-year-old grand challenge in biology.*

https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology [2021-02-11].

Burke, K (2019). *How Do Self-Driving Cars Make Decisions?*.

https://blogs.nvidia.com/blog/2019/05/07/self-driving-cars-make-decisions/ [2021-02-18].

Bildkällor:

Figur 1 - 6: (Nielsen, M 2019)