Metoder för regularisering av djupa neuronnät

Martin Rejström

30.03.2020

Innehåll

Introduktion

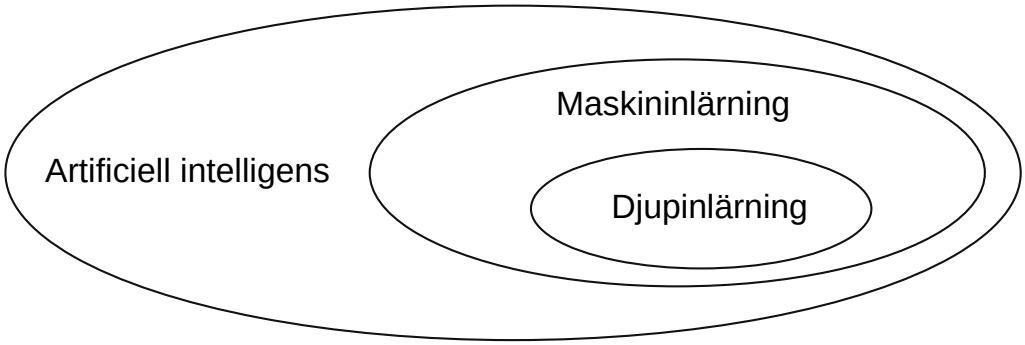
Djupinlärning är en specifik form av maskininlärning som är inspirerad av hjärnan och använder djupa neuronnät för att lära datorer lösa uppgifter med hjälp av en stor mängd exempeldata. Neuronnäten har en tendens att lära sig information ur exempeldatan som inte stämmer för de data de senare opererar på, och har därför svårt att göra rätta förutsägelser. För att lösa problemet har metoder utvecklats vars uppgift är att modifiera nätverket så att det endast lär sig den viktigaste informationen ur exempeldatan och på så sätt blir bättre på att utföra uppgiften. Dessa metoder går under namnet regularisering och presenteras i detta arbete. Regularisering är en av de viktigaste delarna inom maskininlärning. För att neuronnät skall uppnå en hög prestanda måste den som designar nätet förstå sig på hur regularisering fungerar och kunna tillämpa den valda regulariseringsmetoden på ett korrekt sätt.

Arbetet är uppdelat i fyra delar, första delen beskriver vad maskininlärning och djupa neuronnät är och redogör viktiga termer och koncept inom området. Den andra delen beskriver teorin över hur djupa neuronnät fungerar för att kunna beskriva vad regularisering är, och hur den fungerar i den tredje delen. Utöver det presenteras också de vanligaste regulariseringsmetoderna. Den sista delen är en kort sammanfattning och diskussion av de olika regulariseringsmetoderna och framtiden för regularisering av djupa neuronnät.

Bakgrund och kontext

Maskininlärning

Målet med maskininlärning är att skapa maskiner som kan lösa uppgifter genom att göra förutsägelser utgående från data. Förutsägelserna görs med hjälp av algoritmer för datorinlärning som tränas med en stor mängd exempeldata [13]. Djup maskininlärning är en avancerad form av maskininlärning som blev populär under början av 2000-talet. Metoden använder djupa artificiella neuronnät som är inspirerade av neuronnäten i hjärnan. Ett venndiagram över hur artificiell intelligens, maskininlärning och djup inlärning relaterar till varandra kan ses på figur 1.



Figur 1: Venndiagram över de olika delarna inom artificiell intelligens

Maskininlärningens grunder ligger i forskningsområdet artificiell intelligens som skapades på 1950-talet genom försök att automatisera intellektuella uppgifter som vanligtvis utförs av människor [2]. Med hjälp av algoritmer och formella regler kunde forskare lösa väldefinierade problem. Det är dock svårt att skriva regler för problem som kräver informell kunskap om världen, till exempel för igenkännandet av objekt eller språkförståelse. Vilka regler måste definieras för att en dator ska känna igen en siffra på en bild? En lösning är att låta datorer lära sig reglerna från en stor mängd relevanta data. Med hjälp av algoritmer för datorinlärning hittar programmet mönster i datan och lär sig reglerna. Med hjälp av dessa regler kan algoritmen sedan göra förutsägelser från aldrig tidigare sedda data. Detta möjliggör att artificiell intelligens kan fungera i komplicerade situationer i den verkliga världen [5].

Mitchell [16] definierar begreppet inlärning inom maskininlärning som alla datorprogram som blir bättre på att lösa en klass av uppgift T (*task*), om programmets prestanda mätt med prestandamåttet P, förbättras med hjälp av erfarenhet E. För att förklara dessa begrepp används här en känd datamängd som heter MNIST [12]. Den består av 70000 skannade bilder på handskrivna siffror från 0 till 9. Till varje bild finns en sammanhörande etikett som fungerar som facit och berättar vilken siffra bilden visar.

I Mitchells definition är uppgiften T den uppgift algoritmen försöker lära sig att utföra. Vid användning av datamängden MNIST är uppgiften traditionellt att känna igen handskrivna siffror och kategorisera dem i rätt grupp (1 - 9). I detta exempel är själva datamängden

MINST den erfarenhet, E, som programmet använder för att lära sig att utföra uppgiften. Under träningsfasen är målet att hitta en funktion som beskriver sambandet mellan en bild på en siffra och rätt kategori. Efter träningsfasen, när algoritmen används för att utföra uppgiften, ska den kunna bestämma vilken siffra som finns på en aldrig förr sedd bild [19]. För att mäta hur effektiv algoritmen är krävs ett prestandamått P. I detta fall kan prestandan mätas med noggrannhet som är ett procentuellt mått på hur många av exemplen som kategoriserats rätt[5].

Beroende på typen av datamängd som programmet lär sig av, kan maskininlärning delas in i övervakad eller oövervakad inlärning. Övervakad inlärning är det som sker i exemplet ovan. Datamängden som används består då av en mängd exempel som i sin tur består av en rad egenskaper (*features*) med en sammanhörande etikett (*label*). Målet i detta fall är klassificering [19]. Förutom klassificering kan övervakad inlärning användas för regression. Målet är då att förutspå ett numeriskt värde utgående från exempel. Uppgiften kan till exempel vara att förutspå försäljningspriset för ett hus utgående från information om huset och området där huset är beläget [4]. Vid oövervakad inlärning finns det ingen etikett. Programmet lär sig då egenskaper hos datamängden [5]. Ett exempel på oövervakad inlärning är klusteranalys där målet är att hitta grupper i datan som hör ihop. Det kan till exempel vara kundgrupper som köper liknande produkter [7].

Alla maskininlärningsalgoritmer erfar inte endast en förutbestämd datamängd. Inom förstärkningsinlärning interagerar programmet med en miljö [5]. Algoritmen får en belöningssignal om den utför den specificerade uppgiften korrekt och försöker maximera chansen att få en belöningssignal i fortsättningen [7]. Ett känt exempel är AlphaGo [26] som lärt sig spela bordspelet go med hjälp av en kombination av övervakad inlärning och förstärkningsinlärning. Först tränas ett neuronnät som använder övervakad inlärning med hjälp av drag från människor som är experter på spelet. Därefter tränas ett neuronnät som använder förstärkningsinlärning för att förbättra resultatet från det första nätverket genom att spela partier mot sig själv och optimera algoritmen för att vinna spel, inte förutspå experternas drag.

Djupinlärningens historia

Djupinlärning är en del av maskininlärning där inlärningen sker med djupa nätverk av artificiella neuroner. Skillnaden mellan neuronnät och djupa neuronnät är endast antalet lager av neuroner som finns i nätverket. Termen djupinlärning började användas runt år 2000 men teorin som ligger bakom neuronnät går bakåt till 1940-talet. [5]

Den första artificiella neuronnätsarkitekturen introducerades 1943 av McCulloch och Pitts [14]. Det är en modell över hjärnaktivitet och beskriver hur neuroner i hjärnan kan samverka för att utföra komplexa beräkningar. Med hjälp av metoden kan data delas in i två olika kategorier utgående från vikter som bestäms av en människa. Rosenblatt [23] utvecklade modellen på 1950-talet till en modell han kallade för en perceptron. Det är den första modellen som kan lära sig vikterna själv utgående från exempeldata [5].

Intresset för artificiella neuroner stannade av under 1970-talet eftersom Minsky och Papert [15] bevisade 1969 att en perceptron inte kan representera simpla funktioner som till exempel exklusiv disjunktion och det fanns inte en effektiv algoritm för att träna nätverk med många lager [16]. Trots att algoritmer för träning av nätverk med många lager uppfanns på 1980-talet ansågs andra maskininlärningsmetoder mer lovande. Stödvektormaskiner och beslutsträd hade högre prestanda än neuronnät eftersom stora nätverk ännu var svåra att träna. Genom att visa hur djupa bayesiska nätverk (*belief networks*) effektivt går att träna skapade Hinton m.fl. [8] ett nytt intresse för neuronnät år 2006. Denna högkonjunktur som populariserade termen djup inlärning fortsätter idag tack vare genombrott som gör att metoden har bättre prestanda än andra maskininlärningsmetoder. [5]

Användningsområden

Problem som lämpar sig bra för maskininlärning är att känna igen mönster i data, känna igen abnormiteter och göra förutsägelser. Till mönsterigenkänning hör igenkännandet av objekt och språkförståelse, till att hitta abnormiteter hör att hitta kreditkortsbedrägeri bland legitima transaktioner och till förutsägelser hör att förutspå vad ett hus kostar utgående från data om huset och omgivningen [7].

Tidigare maskininlärningsmetoder fungerade bra för att analysera strukturerade data. Det är till exempel data ur en databas där varje värde har en väldefinierad definition. Djupa neuronnät har högre prestanda än tidigare maskininlärningsmetoder vid användning av strukturerade data men är mycket bättre på användningsområden där ostrukturerade data används. Ostrukturerade data är data där det inte finns en definition för varje värde. Det kan till exempel vara bilddata där varje värde beskriver en pixel. Den mycket högre prestandan vid användning av ostrukturerade data har möjliggjort nya användningsområden [19].

Djupa neuronnät kan tillämpas inom många olika områden och för olika uppgifter. Sensorer runt självkörande bilar fångar information om omgivningen som neuronnät analyserar för att ge en styrsignal till bilen. Neuronnät används också för att bestämma vilka användare som visas vilken reklam på internet för att maximera klickningar. Andra användningsområden är identifiering av objekt på bilder, transkribering av talat språk och översättning från ett språk till ett annat [19].

För olika uppgifter används olika typer av neuronnät. Standard framåtmatande (*feedforward*) nätverk används för strukturerade data, faltningsnätverk (*convolutional neural network, CNN*) används för objektigenkänning och återkommande neuronnät (*recurrent neural networks, RNN*) används för data i en följd. Exempel på data i en följd är språkförståelse där tiden är följden eller översättning där orden kommer i en följd [19]. Detta arbete fokuserar på standard framåtmatande nätverk som använder övervakad inlärning.

Orsaker till populariteten av djupa neuronnät i dag

Orsaken till att djupa neuronnät har dominerat maskininlärningen sedan början av 2000-talet är att metoden erbjuder mycket bättre prestanda vid problem där det finns tillgång till en stor mängd data [2]. Vid lösning av problem där det endast finns tillgång till en mindre mängd träningsdata är traditionella maskininlärningsmetoder ungefär lika effektiva. När mängden data ökar kan inte mera traditionella metoder hitta de komplicerade mönstren i datan. Djupinlärning kan hitta dessa mönster förutsatt att nätverket är tillräckligt stort [19].

Många tekniker som används inom djupinlärning idag var redan påhittade i slutet av 1980-

talet, långt före boomen som började i början av 2000-talet. Tre faktorer som hade stor inverkan på detta är framsteg inom hårdvara, tillgång till data och algoritmiska framsteg. Hårdvaruframstegen kommer från datorspel som kräver en stor mängd parallella flytpunktsberäkningar, samma beräkningar som djupa neuronnät använder. Den hårdvaran har senare skräddarsytts för användning till beräkningar för artificiella neuronnät. Tillgången till data har ökat efter att internet blev populärt och kapaciteten för att spara en stor mängd data har ökat. Det möjliggör samlandet och distributionen av stora datamängder. De algoritmiska framstegen kommer från de två första faktorerna, mer intresse leder till mer investering och fler människor som forskar i området [2].

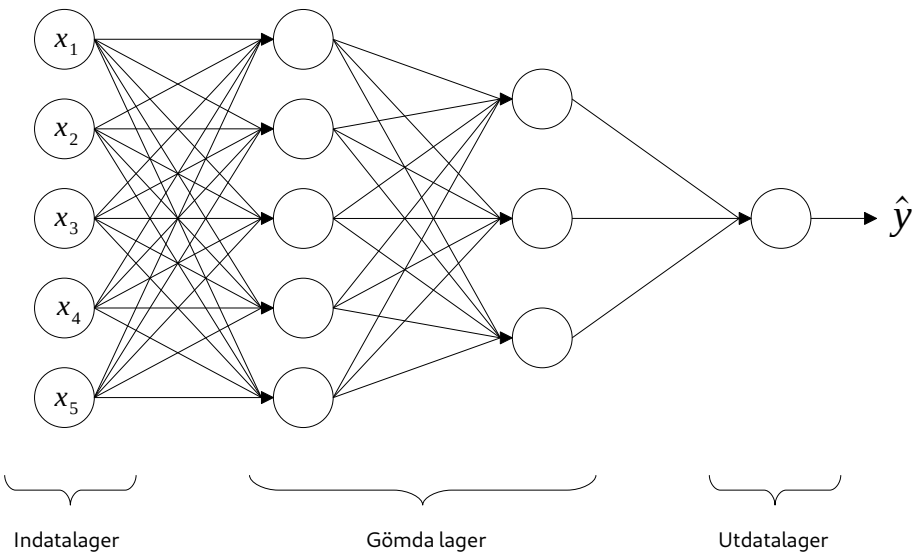
Teori över djupa neuronnät

I detta kapitel beskrivs hur neuronnät är uppbyggda och de matematiska operationer som ligger bakom metoden. Kapitlet fokuserar på standard framåtmatande neuronnät som använder övervakat inlärande. Djupa artificiella neuronnät är inspirerade av hjärnans nätverk av neuroner. Därför beskrivs först hur neuroner i hjärnan fungerar, följt av en beskrivning på en artificiell neuron samt hur nätverk av dessa skapas. Figur 2 visar ett neuronnät där varje cirkel är en enskild artificiell neuron. Neuronerna arrangeras i lager, där utdatan från föregående lager är indata för nästa lager [20].

Artificiella neuronnäts relation till hjärnan

En neuron i hjärnan är en cell med en cellkropp och en lång axon som är kopplad till många andra neuroner genom synapser. Neuronen mottar elektriska impulser via dessa synapser, om den får tillräckligt många inkommande signaler under några millisekunder skickar den iväg en signal via axonen till de neuroner den är kopplad till [4].

Artificiella neuronnät använder hjärnan som inspiration. Hjärnan fungerar som bevis på att ett stort parallellt nätverk av simpla neuroner kan utföra mycket komplexa beräkningar. Dagens neuronnät är specialiserade för att utföra en specifik uppgift, hjärnan ger dock hopp om att det finns algoritmer som är mer generella och att samma nätverk skulle klarar av att



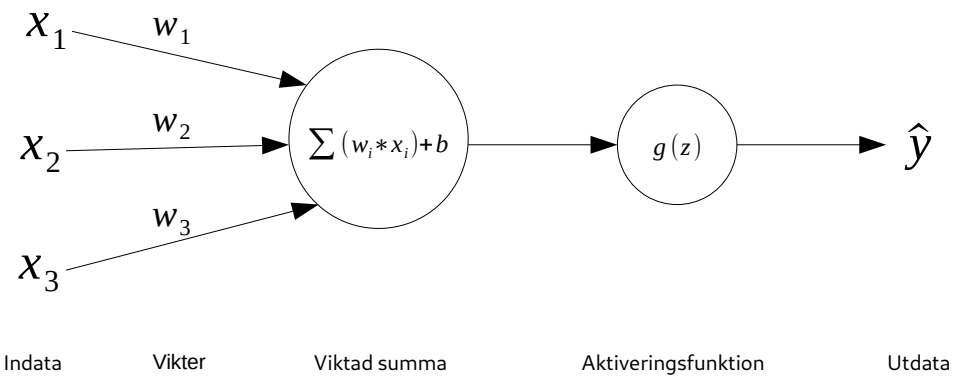
Figur 2: Nätverk av neuroner

lösa olika sorters uppgifter [7]. Sharma et al. [25] bevisade att hjärnans olika regioner kan användas för olika uppgifter än de är menade för. De kopplade unga illrars synnerv till deras hörselcentrum. Hörselcentret lärde sig att tolka signalerna från ögat men illrarna lärde sig inte att se lika bra som med det vanliga syncentrumet.

Hjärnan har tidigare fungerat mer som en planritning som forskare rakt försökt simulera i mjukvara. Neurovetenskap är inte längre den viktigaste guiden i området men fungerar ännu som inspiration. Hjärnan influerar arkitekturen men vi vet inte tillräckligt om hur den fungerar för att den skulle kunna påverka algoritmer. För att göra det måste det gå att mäta vad som händer i tusentals neuroner samtidigt [5].

En artificiell neuron

Den artificiella neuronen är en matematisk modell som beräknar utdata utgående från indata. Till skillnad från neuronerna i hjärnan måste inte in- och utsignalerna vara binära. I detta kapitel beskrivs en standard version av en artificiell neuron som kan ses på figur 3.



Figur 3: En artificiell neuron

Användningen av en artificiell neuron kan delas in i två faser, först en inlärningsfas och sedan en tillämpningsfas. Neuronen tränas med hjälp av en datamängd som består av träningsexempel där varje enskilt exempel består av en rad egenskaper $x_1 - x_n$ och en etikett y [20]. Målet under inlärningsfasen är att lära neuronen relationerna mellan in- och utdatan genom att approximera en funktion $y = f(x)$ [5]. Under tillämpningsfasen används neuronen för att utföra uppgiften den tränats för. Då ska neuronen ha förmågan att förutspå ett korrekt utvärde utgående från aldrig förr sedda indata.

Framåtsteget

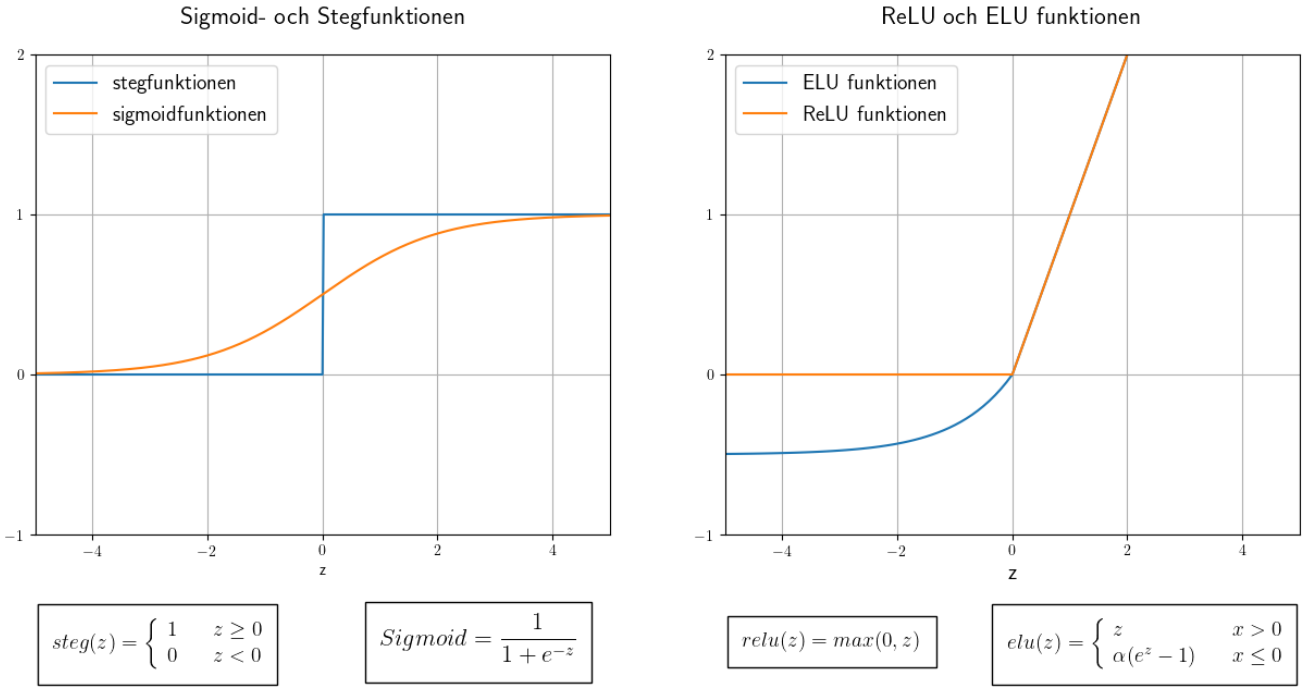
Beräkningen av neuronens utdata, $\hat{y}$, sker med ett framåtsteg, till skillnad från ett bakåtsteg som används vid inlärning och beskrivs senare. När $\hat{y}$ beräknas för ett exempel är den första beräkningen som sker en viktad summa av alla indata, summerad med en bias, b, enligt formel 1 [20].

$$z = \sum_i (w_i x_i) + b \tag{1}$$

Varje enskild egenskap i indatan har en tillhörande vikt $(w_1, w_2, ..., w_n)$ som beskriver hur

stor påverkan de enskilda egenskaperna i exemplet har för slutresultatet. Bias är ett tal som beskriver hur lätt hela neuronen antar ett större utvärde [20]. Vikterna och bias kallas nätverkets parametrar och det är de som förändras när nätverket lära sig sambandet mellan in- och utdata.

Den andra beräkningen är en aktiveringsfunktion som beräknar neuronens slutliga utvärde. Den exakta formuleringen av aktiveringsfunktionen är inte lika viktig som att den har vissa egenskaper som krävs av användningsändamålet. Vad dessa egenskaper är beskrivs genom att beskriva aktiveringsfunktioner som varit populära genom tiden. Grafer av aktiveringsfunktionerna ses på figur 4.



Figur 4: Aktiveringsfunktioner

Ett viktigt krav för aktiveringsfunktionen i enskilda neuroner och utdatalagret i neuronnät är att det beräknade värdet $\hat{y}$ antar liknande värden som exemplens etikett, y [19]. Perceptronen använder en stegfunktion som aktiveringsfunktion eftersom det är en binär klassificerare. Beroende på om z är positiv eller negativ blir resultatet 1 respektive 0. Före år 2010 var sigmoidfunktionen den mest använda aktiveringsfunktionen [21]. Den kan ses som en utslätad

stegfunktion och underlättar träningen av neuronnät eftersom det är lättare att räkna ut hur små förändringar i parametrarna leder till önskade förändringar i $\hat{y}$ [20]. Sigmoidfunktionen kan dock leda till långsam inlärning eftersom funktionens derivata är nära 0 för stora delar av möjliga värden på z. Orsakerna till varför sigmoidfunktionen fungerar bättra än stegfunktionen och varför inlärningen kan ske långsamt framgår när träning av en neuron beskrivs [19]. Den populäraste aktiveringsfunktionen sedan den introducerades år 2010 av Nair och Hinton [18] är ReLU (*rectified linear unit*), eftersom den gör att träningen sker snabbare på grund av att derivatan är 1 för alla positiva värden på z. Eftersom träningen sker snabbare kan större nätverk tränas, vilket i sin tur leder till högre prestanda. En nackdel med relu funktionen är att derivatan är 0 för negativa värden på z. I praktiken har det dock visats att det inte är ett stort problem [19]. Utöver dessa används många andra aktiveringsfunktioner. Till dem hör ELU (*Exponential Linear Units*) [3] som åtgärdar nackdelar med ReLU och uppnår en högre inlärningshastighet på grund av att den antar negativa värden då z är under 0 [21]. Det finns ingen regel som skulle bestämma vilken aktiveringsfunktion som fungerar bäst för ett ändamål. Valet sker ofta utgående från erfarenhet och genom försök och misstag [19]. Hur designprocessen fungerar beskrivs i slutet av kapitlet.

Om en aktiveringsfunktion inte används eller om den är linjär består beräkningen av endast linjära operationer och $\hat{y}$ beräknas endast som en linjär funktion av $x_1 - x_n$. Detta leder till att neuronen inte kan modellera lika komplexa förhållanden mellan in- och utdata [5]. Icke-linjära aktiveringsfunktioner är speciellt viktiga för djupa nätverk av artificiella neuroner. Om aktiveringsfunktionen är linjär eller om den inte används, ökar inte nätverkets förmåga att modellera en mer komplicerad funktion mellan in- och utdata när antaler lager ökar. Förmågan ökar inte eftersom en sammansatt funktion av linjära funktioner är själv en linjär funktion [19].

För att effektivera beräkningen av neuronens utdata och för att bättre utnyttja grafikkort och processorers förmåga att utföra parallella instruktioner sker beräkningen av neuronens utdata i vektorform. Det är inte nödvändigt för en artificiell neuron men för neuronnät är det viktigt eftersom det möjliggör att större nätverk tränas. En vektoriserad version för beräkningen av $\hat{y}$ för alla exempel samtidigt ges i formel 2 [19].

$$\hat{y} = g(w^T X + b) \tag{2}$$

Resultatvektor $\hat{y}$ är en radvektor som innehåller det beräknade resultatet $\hat{y}$ för alla exempel. Funktionen $g()$ är den aktiveringsfunktion som används. Vektor w innehåller alla vikter. Matrisen X innehåller alla egenskaper där varje kolumn i matrisen är egenskaperna i ett exempel. Radvektor b har samma längd som antalet exempel och varje element har värdet b [19].

Träning av en artificiell neuron

Inlärningen sker genom att modifiera neuronens parametrar så att den blir bättre på att modellera funktionen mellan egenskaperna och etiketterna i träningsmängden [20].

Strategin för inlärningen sker genom att iterera genom följande steg:

1. Beräkna neuronens utdata ($\hat{y}$) för alla exempel i träningsmängden.
2. Beräkna medelskillnaden mellan $\hat{y}$ och exemplens etiketter (y)
3. Uppdatera neuronens parametrar så att skillnaden mellan $\hat{y}$ och y minskar inför nästa iteration

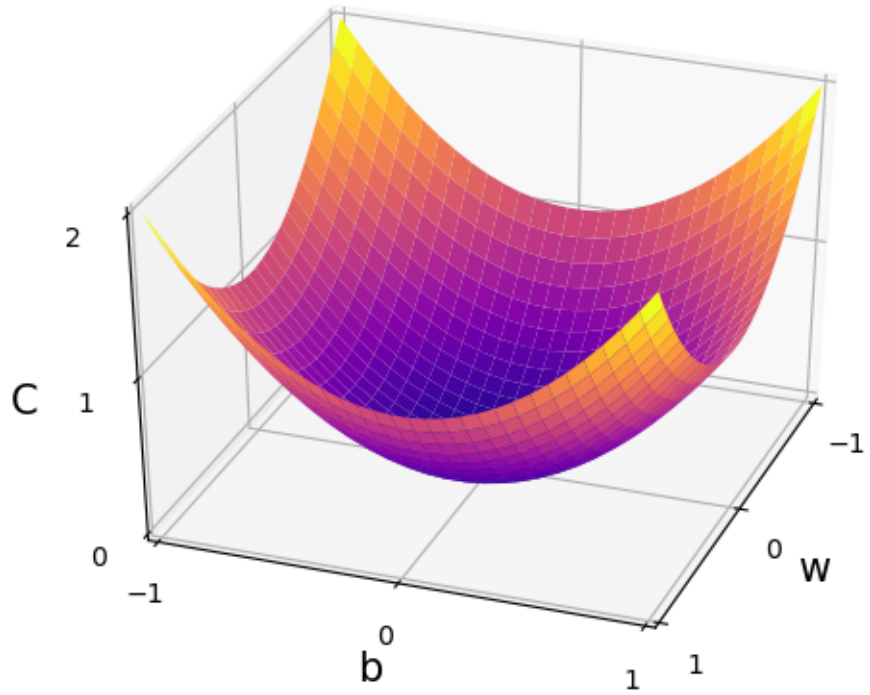
Genom att utföra dessa steg många gånger kommer skillnaden mellan $\hat{y}$ och y minska stegvis. När skillnaden har nått ett minimum, har neuronen lärt sig att modellera sambandet mellan egenskaperna och etiketten i träningsexemplen och kan användas för att förutspå y ur givna x [2].

Medelskillnaden mellan $\hat{y}$ och y för alla exempel beräknas med en kostnadsfunktion (*cost function*) C. Kostnadsfunktionen är medeltalet av skillnaden för alla enskilda exempel. Skillnaden för ett exempel beräknas med en förlustfunktion (*loss function*), L, som beskriver hur nära nätverkets nuvarande uppskattning av resultatet ligger det egentliga sanna resultatet. Formel 3 beskriver kostnadsfunktionen och visar en förlustfunktion.

$$C(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}), \quad L(\hat{y}, y) = (\hat{y} - y)^2 \tag{3}$$

Olika förlustfunktioner kan användas, som exempel används här kvadratfelfunktionen (*squared error*). Funktionen har egenskaper som lämpar sig för uppgiften. Negativa och positiva skillnader mellan $\hat{y}$ och y ger samma resultat och är större ju längre talen är från varandra. Små ändringar i neuronens parametrar resulterar i små ändringar i resultatet från kvadratfelfunktionen, vilket gör nätverket lättare att träna. [20].

För att minimera kostnadsfunktionen beräknas hur neuronens parametrar påverkar kostnaden och därefter modifieras de på så sätt att den totala kostnaden minskar [19]. Om kostnadsfunktionen endast skulle bero på en variabel skulle det vara möjligt att räkna ut var funktionens derivata är 0 och därefter hitta ett minimum. I neuronnät kan kostnadsfunktionen bero på miljarder variabler, därför sker minimeringen av kostnadsfunktionen genom nedstigning med gradientmetoden (*gradient decent*) [20]. Värdet på kostnadsfunktionen kan ses som höjden av en yta. Ytan visas på figur 5 med endast bias och en vikt för att den ska vara möjlig att rita.



Figur 5: Kostnadsfunktionens yta

Genom att beräkna gradienten av kostnadsfunktionen C och uppdatera parametrarna i den motsatta riktningen till den största lutningen, tas ett litet steg åt det hållet som minimerar C mest. Genom att upprepa processen tills gradienten är 0 hittas ett minimum [20].

För att uppdatera parametrarna beräknas partiella derivator av förlustfunktionen. Den partiella derivatan av förlustfunktionen med avseende på en vikt, beräknar hur förlustfunktionen ändrar när värdet på endast den vikten förändras. Genom att uppdatera parametrarna enligt formlerna 4 och 5, minskar värdet av förlustfunktionen [5]. Det kan ses som att ta ett litet steg mot bottnen av skålen i figur 5. Formlerna 4 och 5 anger hur parametrarna ska förändras för ett exempel i träningdatan. Genom att utföra beräkningen för alla träningsexempel och räkna medeltalet vet vi hur parametrarna ska förändras för att minimera kostnadsfunktionen [19].

$$w_i := w_i - \alpha \frac{\partial L(\hat{y}, y)}{\partial w_i} \tag{4}$$

$$b := b - \alpha \frac{\partial L(\hat{y}, y)}{\partial b} \tag{5}$$

Variabeln alpha heter inlärningshastighet (*learning rate*) och har ofta ett litet positivt värde. Inlärningshastigheten beskriver hur stort steget är mot minimum av C i denna iteration av gradientmetoden [20].

Nätverk av artificiella neuroner

En artificiell neuron kan modellera simpla funktioner mellan egenskaper och etiketter i träningsdatan. För att modellera mer komplicerade funktioner behövs en mer expressiv modell. Det kan uppnås genom att använda många artificiella neuroner organiserade som ett nätverk. Djupa nätverk, d.v.s nätverk med många lager, kan lära sig funktioner som nätverk med få lager inte kan. Ett tillräckligt stort neuronnät kan modellera vilken som helst funktion, $y = f(x)$, men det är inte sagt att inlärnigsalgoritmen klarar av att hitta rätt värden för nätverkets parametrar [5].

På figur 2 i början av kapitlet ses ett neuronnät som består av sammankopplade neuroner. De enskilda neuronerna i ett nätverk kallas för noder. Noderna arrangeras i lager, längst till vänster är indatalagret och längst till höger utdatalagret. Indatalagret utför inte beräkningar, det antar endast värden för enskilda egenskaper i träningsexemplen. Lagren mellan dem heter dolda lager (*hidden layer*) eftersom resultaten ur deras beräkningar inte är kända [20]. På figur 2 syns endast en neuron i utdatalagret men beroende av uppgiften som utförs kan det finnas många. Till exemplet vid klassificering finns det ofta en nod per klass. Varje lager i nätverket kan ses som en funktion som transformerar utdata från det föregående lagret till en ny representation: $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Genom att sätta många sådana transformationer i rad kan mycket komplexa funktioner åstadkommas [5].

Ett nätverk med endast ett dolt lager kan anpassas till vilken träningsmängd som helst men i värsta fall krävs det en nod per varje möjliga konfiguration av indatan. Djupare nätverk kan ofta använda mycket färre antal noder för att modellera samma funktion men är svårare att träna. Det ideala djupet och bredden av nätverket måste hittas med hjälp av en experimentell process som beskrivs närmare i slutet av kapitlet [5].

Beräkning av nätverkets framåtsteg

Beräkningar för varje nod i nätverket sker enligt samma princip som beräkningarna för en neuron. För att effektivera beräkningarna sker de för ett lager åt gången och för alla träningsexempel samtidigt enligt formel 6, där l anger vilket lager av nätverket beräkningen sker för. [2].

$$a^l = g(w^{(l)}a^{(l-1)} + b^{(l)}) \tag{6}$$

Resultatet av beräkningen är en tensor, a, som fungerar som indata för nästa lager av noder. Tensorer är databehållare av varierande dimensioner, ofta för numerisk data. En vektor är en 1D tensor, en matris en 2D tensor och så vidare där varje dimension ökar antalet axlar med ett [2]. Dimensionerna på tensorerna i beräkningen beror på indata och hur många noder det finns i det aktuella och föregående lagret. Utdatan för ett lager kallas för lagrets

aktivering (*activation*). Sista lagrets aktivering är utdata för hela nätverket, $\hat{y}$. Skillnaden mellan beräkningen för en nod och beräkningen för ett lager av noder är att tensoren för aktiveringen för ett lager har en högre dimension eftersom den innehåller aktiveringen för varje nod i lagret [19].

Bakåtfortplantning för nätverk

Principen för träningen av ett nätverk är den samma som för gradientmetoden för en neuron. Beräkningen av gradienterna sker med en algoritm som heter bakåtfortplantning (*backpropagation*) [24]. Namnet kommer från att informationen från kostnadsfunktionen fortplantas bakåt i nätverket lager för lager när gradienten beräknas. En detaljerad beskrivning av hur bakåtfortplantning i nätverket fungerar är utanför omfattningen av detta arbete. Till nästa följer en överblick över hur bakåtfortplantning fungerar. Bakåtfortplantning använder kedjeregeln för derivering av sammansatta funktioner för att räkna ut hur alla lager i nätverket påverkar kostnadsfunktionen. Om varje lager av nätverket ses som en funktion som opererar på utdatan från det föregående lagret, kan hela nätverket ses som funktioner kopplade i en kedja $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Med hjälp av kedjeregeln kan varje lagers derivator räknas i en bakåtgående ordning [5].

Design av djupa neuronnät

Vid design av nya neuronnät bör många faktorer tas i beaktande. Till dessa hör hur många lager nätverket har, hur många gånger iterationerna i gradientmetoden utförs och vilken aktiveringsfunktion som används. Dessa faktorer kallas hyperparametrar eftersom de påverkar nätverkets parametrar. Design av neuronnät är en iterativ process där designern väljer hyperparametrar och ser hur de påverkar resultatet. Nätverket lär sig att modellera datan som den tränas på och uppnår en högre prestanda på just den datan. Därför delas träningsdatan in i tre olika mängder, en för träning, en för utveckling och en för testning. Under desigprocessen väljs olika värden på hyperparametrar varefter nätverket tränas med hjälp av träningsdatan. Hur bra nätverket presterar evalueras sedan med hjälp av utvecklingsmängden. Denna iteration sker tills de bästa värden på hyperparametrarna hittas. Eftersom utvecklingsmängden påverkar valet av hyperparametrar och därför nätverkets parametrar krävs testdata för att mäta den

egentliga prestandan i slutet av processen. [19]

Regularisering av neuronnät

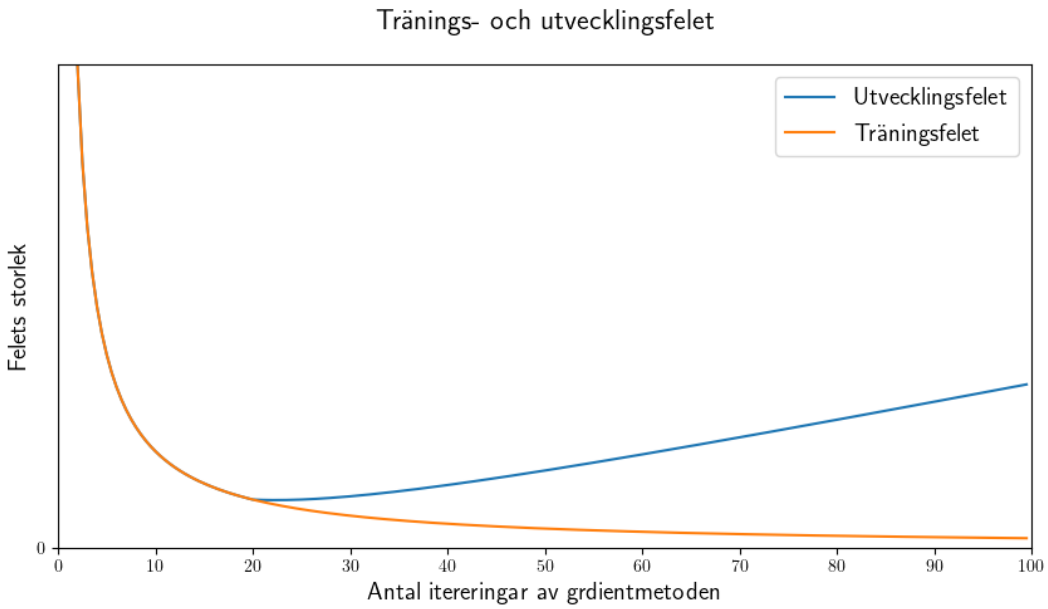
Regularisering är en av de viktigaste delarna av djupinlärning. Metoderna som går under namnet regularisering har som gemensamt mål att öka nätverkets prestanda under tillämpningsfasen. Tidigare har endast metoderna som beskrivs i kapitlet om L_2-regularisering kallats för regularisering men nuförtiden har begreppet en bredare mening [11]. I detta kapitel beskrivs först koncept som krävs för att förklara regularisering, därefter presenteras de populäraste regulariseringsmetoderna.

Över- och underanpassning

Eftersom djupa neuronnät kan hitta mycket komplexa samband mellan egenskaperna och etiketten i träningsdatan, hittar de samband som finns i träningsdatan men inte i datan under tillämpningsfasen. Skillnaden mellan sambanden i träningsdatan och de data som nätverket opererar på i tillämpningsfasen kan bero på mätningsfel eller annan brus. Hur bra neuronnät presterar på nya data kallas för hur bra nätverket generaliserar [5]. Under inlärningsfasen är målet att prestera så bra som möjligt på träningsdatan men det slutgiltiga målet är att generalisera till nya data med så hög prestanda som möjligt. För att mäta hur bra nätverket presterar introduceras ett prestanda- eller felmått (*error measure*). Exakt hur måttet är definierat beror på uppgiften nätverket tränas för. För klassificering kan måttet vara hur stor andel av datan som klassificeras fel. Felets storlek på träningsmängden kallas i detta arbete för träningsfelet och felets storlek på utvecklingsmängden för utvecklingsfelet. För att neuronnätet ska prestera bra måste både träningsfelet och skillnaden mellan träningsfelet och utvecklingsfelet vara små [5].

Figur 6 visar hur tränings- och utvecklingsfelet typiskt variera som en funktion av antalet iterationer av gradientmetoden. Vid början av inlärningen minskar både tränings- och utvecklingsfelet. I det skedet har neuronnätet inte modellerat alla mönster i träningsmängden. Om nätverket inte lyckas uppnå ett tillräckligt lågt träningsfel är det underanpassat (*underfitting*).

När fler iterationer av gradientmetoden utförs går träningsfelet ner men utvecklingsfelet börjar gå upp. Det är ett tecken på att modellen har överanpassat träningsmängden (*overfitting*), det vill säga lärt sig mönster som är specifika för träningsmängden och som inte går att generalisera till nya data. [2]. Förloppet på figur 6 kräver att nätverket har tillräcklig kapacitet för att överanpassa nätverket [5].



Figur 6: Tränings- och utvecklingsfelen

För att nätverket ska kunna uppnå ett lågt utvecklingsfel, d.v.s generalisera bra till nya data, måste det först ha ett lågt träningsfel. Åtgärder som kan minska träningsfelet är att träna ett större nätverk, träna nätverket en längre tid eller testa en annan nätverksarkitektur som CNN eller RNN [19]. Om nätverket har ett lågt träningsfel men är överanpassat kan metoder användas för att förbättra nätverkets förmåga att generalisera till nya data. Dessa metoder kallas för regulariseringsmetoder och beskrivs i resten av kapitlet.

Metoder för regularisering av neuronnät

Regularisering är ett kollektivnamn för alla förändringar som görs till inlärningsalgorimen vars mål är att få nätverket att generalisera bättre, det vill säga minska utvecklingsfelet [11]. Den bästa metoden för att minska överanpassning är att skaffa mer träningsdata [5]. Om modellen tränas på mer data finns det färre specifika mönster i träningsdatan som

nätverket kan modellera. Det finns dock en begränsad mängd träningsdata och det är inte alltid möjligt att skaffa mer. Det andra alternativet är att tvinga nätverket att fokusera på de mest generaliserbara mönstren i datan. Det är den tekniken regulariseringsmetoderna använder [2].

Nätverkets storlek och arkitektur

Antal lager, noder per lager och hur de är kopplade till varandra bestämmer nätverkets kapacitet. Kapacitet är ett mått på hur mycket nätverket kan lära sig ur datan. Ett större nätverk har fler parametrar och kan därför lära sig mer om träningsdatan. Ett nätverk med tillräckligt stor kapacitet kan lära sig ett perfekt samband mellan träningsexemplens egenskaper och etiketter. Ett sådant nätverk memorerar träningsexemplen istället för att lära sig mönster i datan och kan inte generalisera till nya exempel lika bra. Ett nätverk med begränsad kapacitet måste istället lära sig de mönster i datan som gäller för många exempel och har bättre förmåga att generalisera. Nätverket måste dock ha tillräckligt stor kapacitet för att kunna lära sig tillräckligt för att uppnå ett lågt träningsfel. Det finns ingen regel som bestämmer hur många lager och noder per lager som är krävs [2]. Erfarenhet och antaganden om hur komplicerat sambandet är mellan in- och utdatan kan ge en indikation på hur hög kapacitet nätverket måste ha [11]. Efter det första antagandet evalueras olika arkitekturer för att hitta en som fungerar bra. En möjlig metod är att börja med ett litet neuronnät och öka storleken tills en lämplig storlek hittas. [2]

Dataökning

Den bästa metoden för att minska överanpassning att öka antalet exempel i träningsmängden. Om det inte är möjligt eller för dyrt är en möjlig metod att använda dataökning (*data augmentation*). Metoden går ut på att skapa nya exempel från de redan existerande exemplen. De nya exemplen måste vara realistiska, en människa ska inte kunna avgöra vilka som är skapade [4]. Beroende på vad den ursprungliga träningsdatan är, är det olika svårt att skapa nya data. Om träningsdatan är bilder, är det till exempel möjligt att beskära, rotera, ändra exponeringen eller svänga bilden spegelvänt. Eftersom dessa är transformationer av en existerande bild kan det löna sig att göra det programmatiskt under inlärningen för att

minimera lagringsutrymmet [4]. Dessa nya bilder är inte lika bra som helt nya eftersom de inte ger lika mycket ny information [19]. När nya data skapas måste man vara försiktig med att de representerar den slutliga uppgiften. Spegelvända bilder på objekt representerar ännu bra samma objekt men om bilden svängs upp och ner försämras representationen. Siffror och text är exempel där bilden inte kan svängas 180 grader [5].

Ett annat sätt att skapa nya data är att sätta till slumpmässig brus till träningsdatan [22]. Det gör att nätverket är mer robust mot brus i datan under tillämpningsfasen. Människor kan känna igen tal trots att en inspelning har mycket bakgrundsbrus. Genom att sätta till brus till träningsdatan kan neuronnät tränas att göra samma sak. Brus kan också vara effektivt om det adderas till de dolda lagren. Denna metod gör att parametrarna hittar värden där små förändringar inte har en stor effekt på utsignalen [5].

Tidigt stopp

Tidigt stopp (*Early stopping*) är en version av regularisering som går ut på att stoppa körningen av gradientmetoden innan regionen av överanpassning nås. Teorin för varför detta fungerar visualiseras på figur 6 i kapitlet om överanpassning. På figuren ses att det finns en punkt där träningsfelet fortsäter gå ner men utvecklingsfelet börjar gå upp. Genom att välja antal iterationer av gradientmetoden som motsvarar denna punkt kan överanpassning undvikas [4]. Tidigt stopp är en av de mest använda metoderna inom djupinlärning eftersom den är simpel att implementera, effektiv och kräver inte mycket beräkningsresurser. De enda extra resurserna som krävs är att beräkna utvecklingsfelet under träningens gång. Genom att stoppa inlärningen sparar metoden dock de resurser som skulle krävas för att köra inlärningsalgoritmen till slut [5].

I praktiken sparas en kopia på värdet av nätverkets parametrar varje gång utvecklingsfelet minskar efter en iteration av gradientmetoden. Gradientmetoden stoppas när utvecklingsfelet inte blivit mindre på ett tidigare bestämt antal iterationer. Efter att träningsalgoritmen kört färdigt används de sparade parametrarna istället för de som sist uppdaterades av gradientmetoden [5].

Eftersom inlärningen stoppas innan nätverket lärt sig allt den kan ur träningsdatan, kan det leda till att träningsfelet aldrig når ett tillräckligt lågt värde. Kostnadsfunktionen minimeras aldrig eftersom gradientmetoden stoppas innan ett globalt minimum hittas. Andra mer sofistikerade regulatiseringsmetoder påverkar inte träningsfelet lika mycket [19]. Tidigt stopp är en bra metod att använda i samband med dem eftersom metoden inte kräver att någonting förändras i träningsalgoritmen som skulle kunna inverka på inlärningen eller andra regulariseringsmetoder. I det fallet sköter de andra metoderna största delen av regulariseringen och tidigt stopp ser till att inlärningen stoppas vid den ideala tidpunkten för att minimera utvecklingsfelet [5].

L_2-regularisering

L_2-regularisering, också kallad viktavklingning (*weight decay*), är en regulariseringsmetod som regulariserar neuronnätet genom att tvinga nätverkets vikter att ha små värden. Det uppnås genom att sätta till en ny term till kostnadsfunktionen. Den nya termen kan ses som en kostnad för stora vikter.

$$C = C_0 + \frac{\lambda}{2m} \sum_{l=1}^{L} ||w^{[l]}||_2^2, \qquad ||w^{[l]}||_2^2 = \sum_{w} w^2 \tag{7}$$

I formel 7 är C_0 den omodifierade kostnadsfunktionen och C den nya kostnadsfunktionen med den nya regulariseringstermen. Index l beskriver vilket lager vikterna hör till och L är det totala antalet lager. Det finns olika regulariseringsmetoder som fungerar enligt samma princip, där den enda skillnaden är vilken term som sätts till kostnadsfunktionen. I formel 7 och i detta kapitel beskrivs L_2-regularisering som är den populäraste. Termen som sätts till är då summan av kvadraterna på alla vikter i nätverket. Summan dimensioneras med $\lambda/2m$ där lambda är en hyperparameter som kallas regulariseringsparameter och m antalet exempel i träningsmängden [20]. Värdet på λ bestäms med samma teknik som andra hyperparametrar. Om λ är noll regulariseras modellen inte alls, och om λ är stor kommer alla vikter vara mycket nära noll [4].

Eftersom kostnadsfunktionen förändras måste också beräkningen av de olika derivatorna i

bakåtpropageringen förändras så att de tar den nya termen i beaktande [20]. Resultatet blir att det kommer till en ny term i beräkningen för de nya vikterna: $-\frac{\alpha\lambda}{m} * w_i$. Den omformulerade funktionen för uppdateringen av vikterna blir då:

$$w_i := (1 - \frac{\alpha\lambda}{m})w_i - \alpha\frac{\partial L(\hat{y}, y)}{\partial w_i} \tag{8}$$

Uppdateringen med regulariseringstermen är den samma som utan, förutom att vikten först minskar på grund av termen före vikten. Hur mycket den minskar beror på regulariseringsfaktor λ [19]. Stora värden på vikterna blir endast möjliga om de har en betydlig inverkan den originala kostnadsfunktionen eftersom den nya termen motverkar stora vikter. [20]. Det förklarar varför metoden leder till att nätverket inte överanpassar träningsmängden lika lätt. Om en vikt inte har en stor inverkan på skillnaden mellan $\hat{y}$ och y dras den vikten ner mot noll. Det leder till att nätverkets kapacitet minskar och endast de mest betydande vikterna inverkar på beräkningen [19].

Vanligtvis regulariseras vikterna men inte nodernas bias. Det är möjligt att tillsätta en term i kostnadsfunktionen som regulariserar bias med samma princip men empiriska test har visat att det inte har en stor inverkan [20].

Bortfallsregularisering

Bortfallsregularisering (*dropout*) [9] [27] går ut på att tillfälligt eliminera slumpmässiga noder i nätverket vid varje iteration av inlärningen. Vid varje inlärningssteg ges varje nod en sannolikhet, p, att deaktiveras för den iterationen av gradientmetoden. Inlärningssteget sker normalt varefter noderna aktiveras och nya noder deaktiveras för nästa inlärningssteg. Hyperparametern p kallas bortfallsgrad (*dropout rate*) och brukar väljas mellan 0.2 och 0.5 [2]. Deaktiveringen av noder sker endast under träningsskedet. När olika hyperparametrar evalueras med hjälp av utvärderingsmängden används alla noder för att få ett mer konstant resultat [19]. I praktiken går tekniken ut på att sätta de eliminerade nodernas aktiveringsvärde till 0. Efter det divideras de kvarvarande nodernas aktiveringsvärde med (1 - p) för att det totala aktiveringsvärdet från varje lager ska vara ungefär lika stora i träning och utvärderingsskedet

[19]. Detta görs för att hålla en konstant storlek på den totala indatasignalen till varje nod vid evaluering och slutlig användning [4]. Utdatalagret deaktiveras inte alls och indatalagret har mycket låg bortfallsgrad. [19].

Det är olika fenomen som samverkar för att förklara varför bortfallsregularisering fungerar. Vid varje iteration av gradientmetoden är nätverket mindre, vilket minskar nätverkets kapacitet. Bortfallsregularisering kan också ses som en form av bagging [1] (ur engelskans *bootstrap aggregating*). Det är en teknik som bygger på att träna många nätverk parallellt och under tillämpningsfasen låta dem alla rösta på vilket resultat som är korrekt. Orsaken till att det fungerar är att olika nätverk ofta inte gör samma fel och överanpassar träningsmängden på olika sätt [5]. Bortfallsregularisering kan ses som att olika nätverk tränas vid varje iteration av gradientmetoden. Det slutliga nätverket är som medeltalet mellan alla dessa nätverk och jämnar ut effekten av överanpassningen [20]. Bortfallsregularisering kräver mycket mindre träningstid och minne än bagging eftersom endast ett nätverk tränas och det finns inte olika parametrar som måste sparas för varje skilt nätverk [5]. När bortfallsregularisering används är dock orginalnätverket utan bortfalll större vilket leder till längre träningstid och krav av mer resurser [17].

Bortfallsregularisering kan inte rakt jämföras med bagging eftersom de olika nätverken endast tränas med en iteration av gradientmetoden och de delar på samma parametrar. Eftersom parametrarna inte alltid hör till samma nätverkstruktur måste de lära sig att fungera bra oberoende av vilka noder som är aktiverade tidigare i nätverket [5]. Noderna kan inte sätta en stor vikt på endast en möjlig insignal och sprider därför ut vikten jämnare mellan alla insignaler. Det är en liknande effekt som L_2-regularisering har på nätverket [19]. Warde-Farley m.fl. [29] har jämfört bortfallsregularisering till bagging av ett stort antal nätverk och visade att alla förbättringar som bortfallsregularisering ger till generalisering inte kan förklaras med endast bagging.

Sammanfattning och diskussion

Målet för neuronnät är att de ska prestera bra på den uppgift de är designade för, inte endast på träningsdatan, men på nya data under tillämpningsfasen. Djupa neuronnät har en hög kapacitet och modellerar mönster i träningsdatan som inte finns i mer generella data. Regulariseringsmetoder krävs för att nätverket ska bortse från dessa mönser som är specifika för träningsdatan och sätta mer vikt på mönster som generlariserar bättre till datan i tillämpningsfasen.

För att se om en regulariseringsmetod är nödvänding lönar det sig att först undersöka tränings- och utvecklingsfelen. Om regularisering används på ett neuronnät som inte kräver det, försämras troligtvis endast prestandan. Metoder som används för regularisering är mycket olika eftersom allting som påverkar nätverkets parametrar kan modifieras för att uppnå en regulariserande effekt [11]. Metoderna som presenterats i detta arbete modifierar bland annat träningsdatan, nätverkets uppbyggnad och val av förlustfunktion

När en regulariseringsmetod skall väljas är det bästa alternativet att ta fram mera träningsdata. Mer data minskar effekten av brus i datan och ger möjligheten för ett nätverk med hög kapacitet att utnyttja kapaciteten utan att överanpassa träningsdatan. Om mer data inte är tillgänglig är det i vissa fall möjligt att skapa mer data ur den existerande datan. Om det är möjligt, är dataökning nästan alltid använd eftersom det kräver lite resurser och möjliggör träningen av större neuronnät som inte överanpassar träningsdatan.

Om det inte går att ta fram mer träningsdata för att utnyttja nätverkets kapacitet, är det möjligt att göra motsatsen och begränsa nätverkets kapacitet på olika sätt. Till metoder som begränsar kapaciteten hör att minska nätverkets storlek, stoppa körningen av gradientmetoden i ett tidigt skede, L_2-regularisering och bortfallsregularisering. Minskning av nätverkets storlek har större inverkar på träningsfelet än de andra metoderna. De bästa resultaten nås med att träna ett så stort nätverk som möjligt och minska utvecklingsfelet med någon annan regulariseringsmetod [5]. Tidigt stopp används i nästan alla neuronnät i kombination med andra metoder eftersom tidigt stopp inte påverkar dem och stoppar gradientmetoden på det lämpligaste stället.

Som för många andra val under neuronnätets designprocess, finns det ingen regel som skulle bestämma vilken metod som är bäst till vilket problem. De olika metoderna har olika för- och nackdelar och påverkar inlärningen på olika sätt. Det är också möjligt att använda många metoder på samma gång. För att uppnå ett bra resultat är det viktigt att förstå hur metoderna fungerar, hur de påverkar resten av neuronnätet och vilka resurser de kräver. Genom denna kunskap och en experimentell process kan en eller fler regulariseringsmetoder väljas som möjliggör ett tillräckligt lågt träningsfel men inte överanpassar träningsdatan.

Moradi m.fl. [17] har jämfört olika regulariseringsmetoder för klassificering av bilder. De kom fram till att L_2-regularisering och dataökning är effektiva metoder för att minimera överanpassning, utan att öka resurskraven betydligt. Bortfallsregularisering ökar storleken på de neuronnät som går att träna utan att överanpassa träningsdatan, men kräver fler iterationer av gradientmetoden. Moradi m.fl. rekommenderar bortfallsregularisering om det finns mellan 5 tusen och 15 millioner träningsexempel. Metoden fungerar inte bra med färre exempel och med fler tar det för länge att hitta ett minimum med gradientmetoden.

Srivastava m.fl. [27] har visat att bortfallsregularisering är effektivare än andra beräkningsmässigt billiga metoder som till exempel L_2-regularisering. De visar också att en kombination av bortfallsregularisering och andra metoder ger ett ännu bättre resultat.

Framtiden för regulariseringsmetoder

Regularisering och utvecklingen av nya regulariseringsmetoder är en mycket viktig del av djupinlärning idag och kommer troligtvis att vara det i fortsättningen. En av de mest använda regulariseringsmetoderna, bortfallsregularisering, presenterades för under tio år sedan och har efter det gett inspiration till liknande tekniker. Som exempel för tekniker inspirerade av bortfallsregularisering kan nämnas DropConnect [28] som deaktiverar förbindelser mellan noder istället för hela noder. Dessa relativt nya och populära metoder i samband med fortsatt aktiv forskning tyder på att nya effektiva metoder kommer att utvecklas inom den nära framtiden. Allting i nätverket som påverkar neuronnätets parametrar kan användas som grund för en regulariseringsmetod. Det ger möjligheter till att nya metoder hittas som modifierar nätverket på olika sätt än de nuvarande metoderna.

Ingen har hittills utvecklat en helt övertygande teori över exakt hur regularisering orsakar neuronnät att generalisera bättre. Forskare hittar på nya metoder, jämför dem med de gamla och försöker förstå varför en del fungerar bättre än andra. Förståelsen över hur de fungerar bygger på heuristiska metoder och tumregler [20] [30]. Det erbjuder möjligheter för mer teoretisk forskning inom området.

Hardt m.fl [6] visar att neuronnät med hög kapacitet kan generalisera bra och kräver nödvändigtvis inte regularisering, förutsatt att de tränas snabbt. Också batch normalization [10] har en regulariserande effekt och gör i vissa fall regulariseringsmetoder onödiga, trots att batch normalization huvudsakligen används för att underlätta inlärningen. Om dessa trender fortsätter kommer betydelsen av skilda regulariseringsmetoder inte nödvändigtvis vara lika stor som i dagens läge.

Referenser

[1] L. Breiman, "Bagging predictors", *Machine Learning*, årg. 24, nr 2, s. 123–140, 1 aug. 1996, ISSN: 1573-0565. DOI: 10.1007/BF00058655. URL: https://doi.org/10.1007/BF00058655 (hämtad 2020-03-27).

[2] F. Chollet, *Deep Learning with Python*, 1 edition. Shelter Island, New York: Manning Publications, 22 dec. 2017, 384 s., ISBN: 978-1-61729-443-3.

[3] D.-A. Clevert, T. Unterthiner och S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)", 22 febr. 2016. arXiv: 1511.07289 [cs]. URL: http://arxiv.org/abs/1511.07289 (hämtad 2020-03-23).

[4] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1 edition. Beijing Boston Farnham: O'Reilly Media, 9 april 2017, 572 s., ISBN: 978-1-4919-6229-9.

[5] I. Goodfellow, Y. Bengio och A. Courville, *Deep Learning*. Cambridge, Massachusetts: The MIT Press, 18 nov. 2016, 775 s., ISBN: 978-0-262-03561-3. URL: http://www.deeplearningbook.org.

[6] M. Hardt, B. Recht och Y. Singer, "Train Faster, Generalize Better: Stability of Stochastic Gradient Descent", 7 febr. 2016. arXiv: 1509.01240 [cs, math, stat]. URL: http://arxiv.org/abs/1509.01240 (hämtad 2020-03-26).

[7] G. Hinton. (). Neural Networks for Machine Learning - Home, URL: https://www.coursera.org/learn/neural-networks/home/welcome (hämtad 2018-08-09).

[8] G. E. Hinton, S. Osindero och Yee-Whye Teh, "A Fast Learning Algorithm for Deep Belief Nets", *Neural Computation*, årg. 18, nr 7, s. 1527–1554, juli 2006, ISSN: 08997667. DOI: 10.1162/neco.2006.18.7.1527. URL: http://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=21004719&site=ehost-live (hämtad 2020-03-24).

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever och R. R. Salakhutdinov, "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors", 3 juli 2012. arXiv: 1207.0580 [cs]. URL: http://arxiv.org/abs/1207.0580 (hämtad 2020-03-27).

[10] S. Ioffe och C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", 2 mars 2015. arXiv: 1502.03167 [cs]. URL: http://arxiv.org/abs/1502.03167 (hämtad 2020-03-30).

[11] J. Kukačka, V. Golkov och D. Cremers, "Regularization for Deep Learning: A Taxonomy", 29 okt. 2017. arXiv: 1710.10686 [cs, stat]. URL: http://arxiv.org/abs/1710.10686 (hämtad 2020-03-26).

[12] Y. LeCun. (). MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges, URL: http://yann.lecun.com/exdb/mnist/ (hämtad 2018-08-20).

[13] (). Maskininlärning - Uppslagsverk - NE.Se, URL: https://www-ne-se.ezproxy.vasa.abo.fi/uppslagsverk/encyklopedi/l%C3%A5ng/maskininl%C3%A4rning (hämtad 2020-03-10).

[14] W. S. McCulloch och W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", *The bulletin of mathematical biophysics*, årg. 5, nr 4, s. 115–133, 1 dec. 1943, ISSN: 0007-4985, 1522-9602. DOI: 10.1007/BF02478259. URL: https://link.springer.com/article/10.1007/BF02478259 (hämtad 2018-08-05).

[15] M. Minsky och S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 22 sept. 2017, 317 s., ISBN: 978-0-262-53477-2. Google Books: PLQ5DwAAQBAJ.

[16] T. M. Mitchell, *Machine Learning*, 1st edition. New York, NY: McGraw-Hill, 1 okt. 1997, 414 s., ISBN: 978-0-07-115467-3.

[17] R. Moradi, R. Berangi och B. Minaei, "A survey of regularization strategies for deep models", *Artificial Intelligence Review*, 5 dec. 2019, ISSN: 1573-7462. DOI: 10.1007/s10462-019-09784-7. URL: https://doi.org/10.1007/s10462-019-09784-7 (hämtad 2020-03-30).

[18] V. Nair och G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines", s. 8,

[19] A. Ng. (). Neural Networks and Deep Learning - Home, URL: https://www.coursera.org/learn/neural-networks-deep-learning/home/welcome (hämtad 2018-07-30).

[20] M. A. Nielsen. (2015). Neural Networks and Deep Learning, URL: http : / / neuralnetworksanddeeplearning.com (hämtad 2018-07-30).

[21] D. Pedamonti, "Comparison of Non-Linear Activation Functions for Deep Neural Networks on MNIST Classification Task", 8 april 2018. arXiv: 1804.02763 [cs, stat]. URL: http://arxiv.org/abs/1804.02763 (hämtad 2020-03-23).

[22] B. Poole, J. Sohl-Dickstein och S. Ganguli, "Analyzing Noise in Autoencoders and Deep Networks", 6 juni 2014. arXiv: 1406.1831 [cs]. URL: http://arxiv.org/abs/1406.1831 (hämtad 2020-03-30).

[23] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.", *Psychological Review*, årg. 65, nr 6, s. 386–408, 1958, ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519. URL: http://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519 (hämtad 2018-08-05).

[24] D. E. Rumelhart, G. E. Hinton och R. J. Williams, "Learning Representations by Back-Propagating Errors", *Nature*, årg. 323, nr 6088, s. 533–536, okt. 1986, ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0 (hämtad 2018-08-23).

[25] J. Sharma och A. Angelucci, "Induction of Visual Orientation Modules in Auditory Cortex", *Nature*, årg. 404, nr 6780, s. 841, 20 april 2000, ISSN: 00280836. DOI: 10.1038/35009043. URL: http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=3069983&site=ehost-live (hämtad 2020-03-23).

[26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel och D. Hassabis, "Mastering the game of Go with deep neural networks and tree search", *Nature*, årg. 529, nr 7587, s. 484–489, 7587 jan. 2016, ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: https://www.nature.com/articles/nature16961 (hämtad 2020-03-22).

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever och R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", s. 30,

[28] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun och R. Fergus, "Regularization of Neural Networks using DropConnect", i *International Conference on Machine Learning*, 13 febr. 2013, s. 1058–1066. URL: http://proceedings.mlr.press/v28/wan13.html (hämtad 2020-03-30).

[29] D. Warde-Farley, I. J. Goodfellow, A. Courville och Y. Bengio, "An Empirical Analysis of Dropout in Piecewise Linear Networks", 2 jan. 2014. arXiv: 1312.6197 [cs, stat]. URL: http://arxiv.org/abs/1312.6197 (hämtad 2020-03-28).

[30] C. Zhang, S. Bengio, M. Hardt, B. Recht och O. Vinyals, "Understanding Deep Learning Requires Rethinking Generalization", 26 febr. 2017. arXiv: 1611.03530 [cs]. URL: http://arxiv.org/abs/1611.03530 (hämtad 2020-03-26).