Optimering av hyperparametrar till artificiella neurala nätverk med genetiska algoritmer.

Simon Stensson

Juni 6, 2016

Abstract

This master thesis explores the feasibility of using genetic algorithms in order to automate the process of optimizing hyperparameters for artificial neural networks (ANN). Today there is no standard way to optimize hyperparameters for ANN; often they are set manually by trial and error. In order to explore the feasibility of using genetic algorithms to optimize hyperparameters for ANN, two algorithms are implemented in Python. The first is a genetic algorithm and the second is an algorithm that trains a neural network and enables predictions. The two algorithms interact in a feedback loop where the genetic algorithm adjusts hyperparameters for the neural network, and the neural network performs predictions on data which is used as feedback to the genetic algorithm. In order to evaluate the method, the implemented models are evaluated on three classification problems. The results are compared to predictions made from neural networks where the hyperparameters are manually set. The method using genetic algorithms to optimize hyperparameters performes slightly better on all three problems, but without a significant improvement in prediction accuracy. The implemented models offer an automated way to optimize hyperparameters for ANN and test results indicates that prediction accuracy is maintained.

Keywords

Artificial neural networks, genetic algorithms, optimiziation, machine learning, classification

Innehåll

1. Inledning ... 1
1.1. Bakgrund och syfte ... 1
1.2. Avgränsningar ... 2
1.3. Rapportens disposition ... 4
2. Teoretisk bakgrund ... 5
2.1. Artificiella neurala nätverk ... 5
2.1.1. Linjär klassificering med perceptronen ... 6
2.1.2. Nätverkstopologi ... 7
2.1.3. Icke-linjära aktiveringsfunktioner ... 8
2.1.4. Neuralt nätverk på matrisform ... 12
2.1.5. Anpassning av neurala nätverk ... 15
2.2. Genetiska algoritmer ... 23
2.2.1. Fitness-funktion ... 24
2.2.2. Representation av parametrar ... 24
2.2.3. Initial population ... 25
2.2.4. Selektion ... 25
2.2.5. Utförande av rekombination ... 26
2.2.6. Utförande av mutation ... 27
3. Metod ... 28
3.1. Implementation i Python ... 28
3.2. Introduktion till dataset ... 28
3.3. Algoritm för neurala nätverk ... 29
3.3.1. Strukturen i neurala nätverk ... 29
3.3.2. Träning av nätverken ... 30
3.3.3. Kostnadsfunktion ... 30
3.3.4. Initiala parametrar och normalisering av data ... 33
3.3.5. Avgöra konvergens för ett nätverk ... 35
3.3.6. Sammanställning av hyperparametrar ... 36
3.4. Implementation av den genetiska algoritmen ... 37
3.4.1. Val av fitness-funktion ... 37
3.4.2. Representation av parametrar ... 38
3.4.3. Initial population ... 38
3.4.4. Selektion ... 39

3.4.5. Rekombination 40
3.4.6. Mutation 41
3.4.7. Samspel mellan de genetiska operatorerna 41
3.4.8. Stoppvillkor 41
3.4.9. Sammanställning av standardparametrar 42
4. Test av utvecklade modeller 42
4.1. Proben1 (Prechelt, 1994) 42
4.2. Beskrivning av klassificeringsproblemen 43
4.2.1. ”Dabetes1” 43
4.2.2. ”Cancer1” 44
4.2.3. ”Glass1” 45
4.3. Utförande 45
4.4. Resultat 46
4.4.1. ”Diabetes1” 47
4.4.2. ”Cancer1” 48
4.4.3. ”Glass1” 49
5. Slutsatser 50
6. Diskussion 51
7. Rekommendation om vidare forskning 52
8. Litteraturförteckning 53

1. Inledning

1.1. Bakgrund och syfte

Forskning kring artificiella neurala nätverk har nått stora framgångar på senare tid. För tillfället ges de bästa lösningarna till många problem inom bildigenkänning, taligenkänning samt datorlingvistik av just artificiella neurala nätverk. (A. Nielsen, 2014) De artificiella neurala nätverken är en statistisk modell som är väldigt flexibel, vilket gör att den är både kraftfull och mångsidig. Flexibiliteten bidrar dock även till att det kan vara svårt att utnyttja modellen till dess fulla potential för enskilda problem. Det uppstår ett svårt optimeringsproblem för att anpassa en lämplig nätverksmodell för varje enskilt problem.

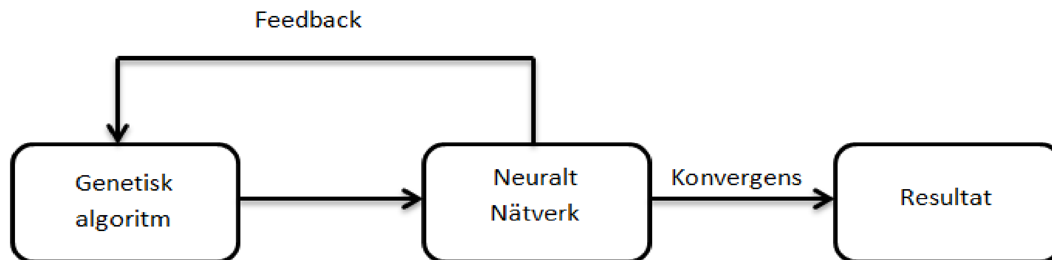
Att anpassa ett artificiellt neuralt nätverk, till ett givet problem, kan delas upp i två steg. Det ena avser att bestämma hyperparametrar till modellen[1], vilket definierar en nätverksmodell. Då hyperparametrarna är bestämda är antalet modellparametrar givna och det andra steget blir att anpassa dessa. Anpassning av modellparametrarna sker vanligtvis genom att minimera en kostnadsfunktion med gradientbaserad optimering. För att optimera hyperparametrar finns däremot ingen vedertagen metod. Ofta bestäms dessa manuellt med hjälp av tumregler och att testa sig fram. Detta är något som är både svårt och i många fall blir en mycket tidskrävande uppgift.

Med tanke på utvecklingen av datorkraft, som bara blir billigare och mer lättåtkomligt för var dag som går, är det attraktivt att automatisera denna typ av processer. På så sätt kan arbetsbörda för människan överföras till arbete av datorer.

Målet med detta arbete är att undersöka om det med hjälp av genetiska algoritmer, vilket är en optimeringsteknik inspirerad från biologins evolutionsteori, är möjligt att automatisera processen att anpassa artificiella neurala nätverk för enskilda problem.

[1] Hyperparametrar kan ses som ett steg i modellvalet av en statistisk modell. Det är inte förän att en modell, inklusive hyperparametrar, är bestämd, som dess modellparametrar är fullt definierade. Modellparametrarnas numeriska värden kan sedan bestämmas genom att anpassa modellen till det givna problemet. För en mer utförlig beskrivning av skillnaden mellan hyperparametrar och modellparametrar hänvisas läsaren till (Davidson-Pilon, 2015).

För detta ändamål kommer två algoritmer att implementeras. En algoritm som modellerar neurala nätverk och möjliggör prediktion, samt en genetisk algoritm som är anpassad att optimera parametrar till dessa nätverk. De två algoritmerna samspelar genom en så kallad ”feedback loop”, se Figur 1 nedan, där nätverksalgoritmen utför prediktioner vilket sedan används som feedback till den genetiska algoritmen som i sin tur återkopplar och utför justeringar i det neurala nätverket.



Figur 1. Skiss över hur den genetiska algoritmen samspelar med det neurala nätverket. Den genetiska algoritmen bestämmer hyperparametrar till det neurala nätverket, det neurala nätverket optimerar modellparametrar och utför prediktion. Resultatet från prediktionen används som feedback till den genetiska algoritmen för att justera parametrarna till de neurala nätverken. Denna loop fortsätter tills dess att konvergens har uppnåtts och en anpassad nätverksmodell erhålls som resultat.

Den implementerade metoden kommer att bedömas utifrån fyra aspekter; robusthet, automatiseringsgrad, generalitet, samt precision. För att bedöma precisionen kommer den utvecklade metoden att appliceras på tre verkliga klassificeringsproblem. Resultaten kommer sedan att jämföras med prediktioner utförda med neurala nätverk där hyperparametrar har bestämts manuellt och modellparametrar har bestämts genom gradientbaserad optimering.

1.2. Avgränsningar

Det finns självklart möjligheter att automatisera optimeringsprocessen för neurala nätverk på många olika sätt. Exempelvis kan metoder så som ”grid search” eller ”random search” utnyttjas för att optimera hyperparametrar till neurala nätverk. (James & Bengio, 2012), (Loshchilov & Hutter, 2016) Med begränsningar både i tid- och textomfång kan inte alla olika möjligheter undersökas. Målet med detta arbete är inte att göra en komparativ studie av andra metoder att optimera hyperparameterar, istället kommer endast genetiska algoritmer att utnyttjas för detta ändamål.

Den främsta anledningen till att just genetiska algoritmer studeras för detta ändamål är dess flexibilitet. Det är exempelvis möjligt att justera den genetiska algoritmen så att den kan optimera topologin inom nätverket, det vill säga bestämma vilka neuroner som ska länkas med vilka och hur. Det går även att utnyttja genetiska algoritmer för att optimera modellparametrar, så som vikter och bias, inom nätverket och att transformera eller reducera in-datan. En annan fördel med genetiska algoritmer är att de är enkla att parallellisera, vilket medför att lösningen kan skalas upp för större problem om tillräcklig beräkningskraft erbjuds.

Artificiella neurala nätverk kan användas till klassificeringsproblem, regressionsproblem samt klustring. (Teuvo & Panu, 2002), (Trevor Hastie, 2009) I detta arbete kommer endast optimering av neurala nätverk för klassificeringsproblem att undersökas. Det krävs dock endast små modifikationer i implementationen av ett neuralt nätverk anpassat för

klassificering jämfört med ett nätverk anpassat för regression. Det finns därför skäl att tro; att om en metod för att optimera neurala nätverk för klassificering fungerar bra, så kommer även denna metod lämpa sig för att optimera neurala nätverk anpassade för regression.

Det finns många olika typer av neurala nätverk. I detta arbete kommer endast så kallade ”feed forwad” nätverk att användas. Detta är den vanligaste typen av artificiella neurala nätverk och här tillåts endast att information skickas framåt inom det neurala nätverket. Andra vanliga neurala nätverk är så kallade ”recurrent” nätverk samt nätverk som anpassas genom oövervakad lärning. Med ”recurrent” nätverk menas nätverk där information även tillåts skickas bakåt inom nätverket och neuroner kan tillåtas att ha loopar. Exempel på nätverk som anpassas genom oövervakad lärning, för träningsdata som ej har definierat klassvärde, är ”self-organizion maps”. (Ronald & Zipser, 1989), (Teuvo & Panu, 2002)

På senare år har det blivit väldigt populärt med så kallade djupa neurala nätverk. De djupa neurala nätverken försöker att modellera en hög nivå av abstraktion i data genom att processa datan i multipla lager, som är sammansatta i komplexa strukturer. På så sätt kan djupa nätverk bygga upp en hierarki av avancerade koncept. Exempelvis har så kallade ”Convolutional neural networks” nått stora framgångar senaste tiden inom framför allt bildigenkänning. (A. Nielsen, 2014)

Med begränsade beräkningsresurser har djupa neurala nätverk inte studerats i detta arbete. Till detta arbete har endast en mindre beräkningsdator med 8 processorer varit tillgängligt, det har inte heller varit möjligt att utföra GPU-beräkningar. Istället har endast fullt kopplade nätverk använts med maximalt två dolda lager. Detta eftersom att syftet med arbetet har varit att studera interaktionen mellan genetiska algoritmer och artificiella neurala nätverk.

Även om många djupa neurala nätverk empiriskt har visat sig erhålla bättre resultat än klassiska fullt kopplade neurala nätverk, går det att bevisa att ett neuralt nätverk innehållandes endast ett dolt lager med icke-linjära aktiveringsfunktioner kan approximera en godtycklig funktion. (Cybenko, 1989), (Kurt, Maxwell, & Halbert, 1989) Vilket innebär att det är en otroligt kraftfull modell med få teoretiska begräsningar.

Eftersom att inga andra typer av artificiella neurala nätverk, än ”feed forward” används i detta arbete, kommer för enkelhetens skull termerna ”artificiella neurala nätverk” eller ”neurala nätverk” att användas för att syfta till just artificiella neurala nätverk av typen ”feed forward”.

För att kunna utvärdera den implementerade modellen kommer den att testas på tre verkliga klassificeringsproblem och resultaten kommer att jämföras med resultat erhållna från neurala nätverk med manuellt bestämda hyperparametrar. Givetvis hade det varit önskvärt med ett bredare test för att validera generalitet och robusthet, detta ligger dock inte inom ramen för detta arbete. De tre problemen som används har däremot valts så att klassificeringsproblemen täcker olika domäner och att antalet klassvärden varierar.

1.3. Rapportens disposition

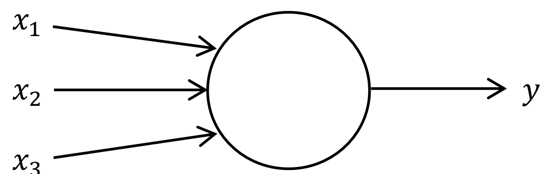
I kapitel 2 ges den teoretiska bakgrund som behövs till detta arbete, dels om neurala nätverk och dels om genetiska algoritmer. I kapitel 3 är fokus att beskriva de val som har gjorts för att implementera algoritmen för neurala nätverk samt den genetiska algoritmen som används för att optimera hyperparametrar till neurala nätverk. I kapitel 4 testas de implementerade algoritmerna på 3 olika klassificeringsproblem och resultaten redovisas i jämförelse med resultat erhålla från neurala nätverk där hyperparametrar bestämts manuellt. Detta följs upp av kapitel 5 där slutsatserna redovisas och i kapitel 6 ges en diskussion om dessa. Rapporten avslutas med kapitel 7 där rekommendationer om vidare forskning inom detta område ges.

2. Teoretisk bakgrund

Detta avsnitt avser att ge den teoretiska bakgrund till neurala nätverk och genetiska algoritmer som behövs till detta arbete. Kapitel 2.1. ger en redogörelse för neurala nätverk och inleds med en beskrivning om hur neurala nätverk är uppbyggda samt hur de kan användas till klassificeringsproblem, därefter sammanställs beräkningarna som sker i ett nätverk på matrisform, och kapitlet avslutas med att beskriva hur neurala nätverk kan anpassas för specifika klassificeringsproblem. Kapitel 2.2. redogör för genetiska algoritmer. Först ges en beskrivning om hur genetiska algoritmer arbetar för att optimera problem därefter beskrivs de genetiska operatorerna som används. Operatorerna beskrivs dels utifrån dess teoretiska inverkan och dels ges exempel på hur de kan implementeras.

2.1. Artificiella neurala nätverk

Ett artificiellt neuralt nätverk är en statistisk modell som kan användas både för klassificering och till regression. (Trevor Hastie, 2009) Det neurala nätverket är ett system där flera så kallade neuroner är sammanlänkade. Varje neuron accepterar en eller flera in-variabler, $x_1, x_2, \ldots, x_p$, utför en funktionsberäkning och returnerar ett funktionsvärde y, som vanligtvis ligger i intervallen: $y \in [-1, 1]$ eller $y \in [0, 1]$. Neuroner i ett neuralt nätverk brukar betecknas med en cirkel enligt Figur 2 nedan.



Figur 2. Skiss över en neuron med tre in-variabler.

Det finns många olika typer av neuroner som kan användas till neurala nätverk[2]. Det som skiljer är dess aktiveringsfunktion, det vill säga vilken funktionsberäkning som utförs. I detta avsnitt beskrivs först en typ av neuron som brukar benämnas för perceptron, därefter ges en beskrivning om hur beräkningar i ett neuralt nätverk utförs och avsnittet avslutas med en genomgång av andra neuroner som kommer att användas i detta arbete.

[2] Med en viss typ av neuron syftas i detta arbete till vilken aktiveringsfunktion en given neuron i ett visst lager har. Exempelvis är perceptronen en typ av neuron och neuronen med sigmoidfunktion en annan. Inte att förväxla med neuronerna tillhörande ett visst typ av lager, ex input, dolt eller output.

2.1.1. Linjär klassificering med perceptronen

Perceptronen fungerar enligt följande; varje in-variabel, $x_1, x_2, \dots, x_p$, multipliceras med varsin realvärd vikt, $\omega_1, \omega_2, \dots, \omega_p$, som anger dess inflytande. Om den viktade summan $\sum_i \omega_i x_i$ är större än ett förbestämt tröskelvärde returneras värdet 1, annars returneras värdet 0. Detta summeras i ekvation (1) nedan, här är u det förbestämda tröskelvärdet:

$$y = \begin{cases} 0 \; om \; \sum_i \omega_i x_i \leq u \\ 1 \; om \; \sum_i \omega_i x_i > u \end{cases} \tag{1}$$

Med $b = -u$, kan ekvation (1) skrivas:

$$y = \begin{cases} 0 \; om \; \sum_i \omega_i x_i + b \leq 0 \\ 1 \; om \; \sum_i \omega_i x_i + b > 0 \end{cases} \tag{2}$$

Där b tolkas som ett bias för modellen och definierar när neuronen ska aktiveras och därmed returnera värdet 1. Med ekvation (2) ovan bestäms perceptronmodellen med $p \; st$ in-variabler $x_1, x_2, \dots, x_p$ av $p + 1$ antal parametrar; vikterna $\omega_1, \omega_2, \dots, \omega_p$ och biasparametern b.

För att förstå hur en perceptron kan användas till ett klassificeringsproblem samt vilken inverkan vikterna och biasparametern har, ges ett kort exempel för att illustrera detta nedan:

Exempel 1:

Detta exempel avser att utforma ett automatiskt system för att sortera ut skräpmail från inkorgen till en E-post. För att hålla exemplet enkelt antas att modellen tar hänsyn till 3 st in-variabler, $x_1, x_2 \; och \; x_3$, som alla antar värdet 0 eller värdet 1. Låt x_1 motsvara om avsändaren till mailet inte finns med i mottagarens kontaktlista, x_2 motsvarar om det finns en länk till en hemsida i brevet och x_3 är relaterad till om mailet är ett massutskick eller inte. Detta sammanställs i ekvation (3) nedan.

$$\begin{aligned} x_1 &= \begin{cases} 1 & om \; avsändaren \; inte \; finns \; i \; kontaktlistan \\ 0 & om \; avsändaren \; finns \; i \; kontaktlistan \end{cases} \\ x_2 &= \begin{cases} 1 & om \; brevet \; innehåller \; en \; länk \\ 0 & om \; brevet \; inte \; innehåller \; en \; länk \end{cases} \\ x_3 &= \begin{cases} 1 & om \; det \; är \; ett \; massutskick \\ 0 & om \; det \; inte \; är \; ett \; massutskick \end{cases} \end{aligned} \tag{3}$$

En tänkbar modell för att avgöra om ett mail ska klassificeras som ett skräpmail med hänsyn till dessa variabler är: om brevets avsändare finns med i kontaktlistan är det inte ett skräpmail.

Kommer brevet från en okänd avsändare krävs det också att minst en av variablerna x_2 och x_3 är lika med 1 för att det ska klassificeras som ett skräpmail.

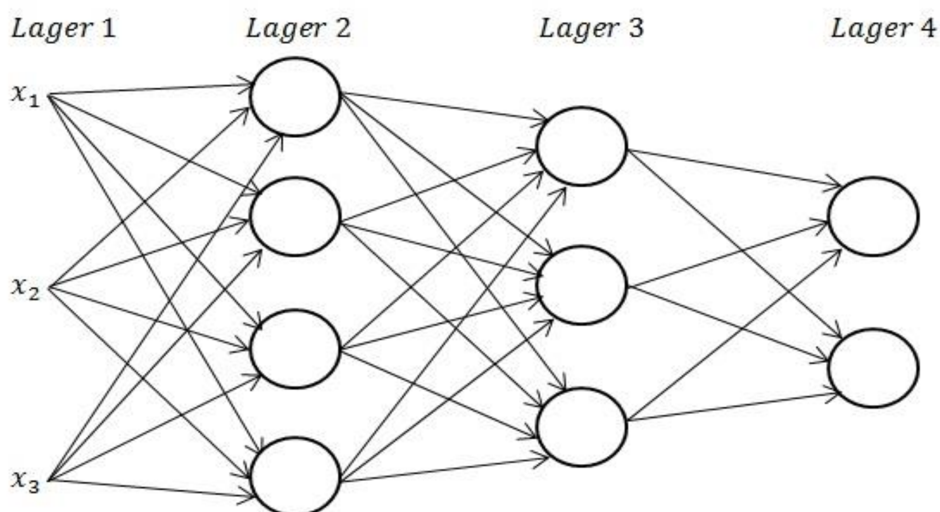
Då variabeln x_1 är den mest betydelsefulla variabeln ges denna variabel den största vikten och en möjlig parameteruppsättning för en perceptron som beskriver ovanstående modell är: $\omega_1 = 2, \quad \omega_2 = \omega_3 = 1, b = -2$ där ett mail klassificeras som ett skräpmail om perceptronen returnerar värdet 1. Denna perceptron sammanfattas i ekvation (4):

$$y = \begin{cases} 0 \; om \;\; 2x_1 + x_2 + x_3 - 2 \leq 0 \\ 1 \; om \;\; 2x_1 + x_2 + x_3 - 2 > 0 \end{cases} \tag{4}$$

Perceptronen bildar hyperplanet $\sum_i \omega_i x_i + b = 0$ och separerar de två klasserna, skräpmail eller icke-skräpmail, beroende på vilken sida av hyperplanet som in-variablerna ligger på. Att skapa en modell med endast en perceptron är vanligtvis inte tillräckligt. Det begränsar modellen till att endast kunna separera två klasser åt gången, samtidigt krävs det även att de två klasserna är linjärt separerbara för att hyperplan ska kunna separera dem.

Detta exempel illustrerar dock hur enstaka neuroner används för att vikta in-variabler och utföra ett beslut. För en mer kraftfull och komplex modell kan man istället bilda ett system där flera neuroner är sammankopplade i ett närverk, ett så kallat neuralt nätverk. Ett neuralt nätverk brukar oftast redovisas med ett schema enligt det i Figur 3 nedan.

2.1.2. Nätverkstopologi



Figur 3. En skiss över ett neuralt nätverk med 4 lager. Nätverket använder 3 st in-variabler och returnerar 2 st ut-variabler.

Detta nätverk består av 4 lager. Det första lagret består av alla in-variabler, $x_1, x_2 \; och \; x_3$, det sista lagret innehåller de värden som returneras från nätverket och som sedan används till klassificeringen. De mellersta lagren, $Lager$ 2 och $Lager$ 3, kallas dolda lager och innehåller fyra respektive tre neuroner. Termen dolda lager refererar till att de värden som returneras från dessa lager inte är direkt observerbara för användaren av ett neuralt nätverk, som skickar

in-data till det första lagret och använder de värden som returneras från det sista lagret. När detta nätverk används till klassificering sker följande:

Från det första lagret skickas alla in-variabler till varje enskild neuron i det andra lagret. I det andra lagret utförs beslut i varje neuron med hänsyn till in-variablerna $x_1,\ x_2\ och\ x_3$. De fyra besluten som fattas i $Lager$ 2 skickas vidare och används som in-variabler till varje neuron i $Lager$ 3 där tre nya beslut fattas som i sin tur skickas vidare till de två neuronerna i $Lager$ 4. I $Lager$ 4 utförs de sista besluten vilka används för att ge ett klassvärde.

Det går att förtydliga schemat i Figur 3. Här är varje neuron utsatt med en cirkel, detta betyder dock inte att alla neuroner måste vara av samma neurontyp. Alla neuroner viktar och summerar in-variablerna och adderar ett bias, men funktionsberäkningen som sker kan skilja. Det är vanligt att variera vilka typer av neuroner som används inom de dolda lagren för specifika problem. I det sista lagret väljs neuronerna beroende på om det är ett regressionsproblem eller ett klassificeringsproblem.

Att länka samman neuroner i ett neuralt nätverk på detta sätt, så att beslut fattas i flera neuroner och i flera nivåer, ger både en komplex och kraftfull modell för klassificering. Det går att bevisa att ett neuralt nätverk beståendes av endast ett dolt lager, innehållandes neuroner med icke-linjära aktiveringsfunktioner, kan approximera en godtycklig kontinuerlig funktion. (Cybenko, 1989), (Kurt, Maxwell, & Halbert, 1989) Det är även möjligt att utföra klassificering för mer än två grupper genom att tillåta fler än två neuroner i det sista lagret.

2.1.3. Icke-linjära aktiveringsfunktioner

Det neurala nätverk som är skissat i Figur 3 är ett relativt litet neuralt nätverk, innehållandes 2 dolda lager med 4 respektive 3 neuroner i varje lager. Det är vanligt att betydligt fler neuroner används inom varje dolt lager, även fler dolda lager kan användas. Men även för ett relativt litet neuralt nätverk är det många vikter och biastermer som måste bestämmas. Ofta används gradientbaserad optimering för att bestämma dessa modellparametrar. Detta har fått som konsekvens att perceptronen används i allt mindre utsträckning i moderna neurala nätverk.

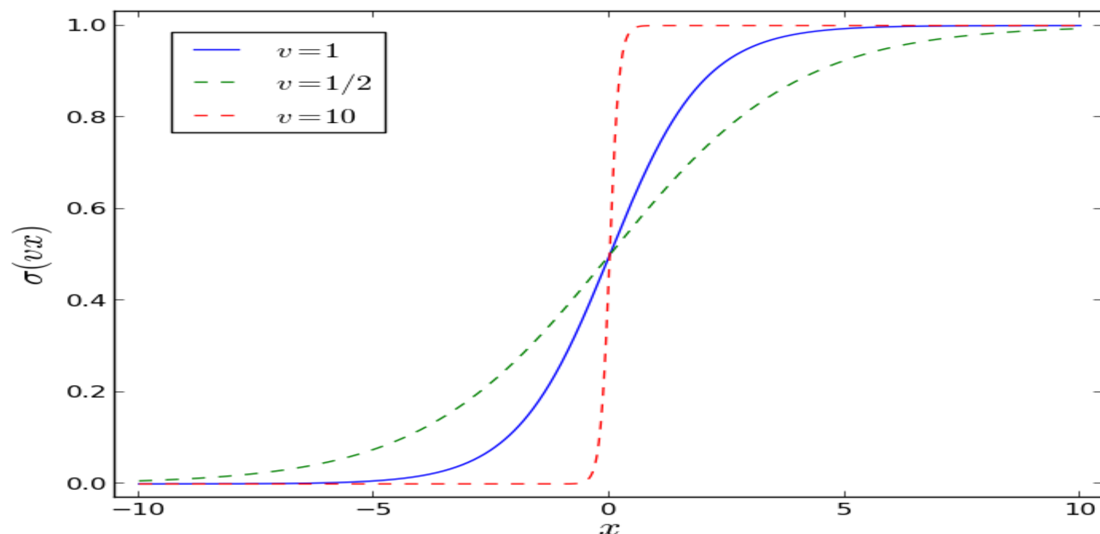
Om perceptronen används i ett neuralt nätverk innebär det att klassificeringen som ges från nätverket är en diskontinuerlig funktion av in-variablerna $x_1, x_2, \dots, x_p$ och det går inte att beräkna en gradient till denna funktion. Detta har medfört att stegfunktionen i perceptronen har ersatts med kontinuerliga aktiveringsfunktioner. Vanligtvis är det sigmoid-neuronen som har ersatt. (Trevor Hastie, 2009)

Sigmoid-neuronen fungerar på liknande sätt som perceptronen; med $p\, st$ in-variabler $x_1, x_2, \ldots, x_p$ består modellen av lika många vikter, $\omega_1, \omega_2, \ldots, \omega_p$ och en biasterm b. Men istället för att använda stegfunktionen enligt ekvation (2), som returnerar värdet 0 eller 1. Används en kontinuerlig funktion som returnerar ett reellt tal i intervallet $[0, 1]$. Sigmoid-funktionen är definierad enligt ekvation (5) nedan och för en sigmoid-neuron returneras värdet, y, enligt ekvation (6).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

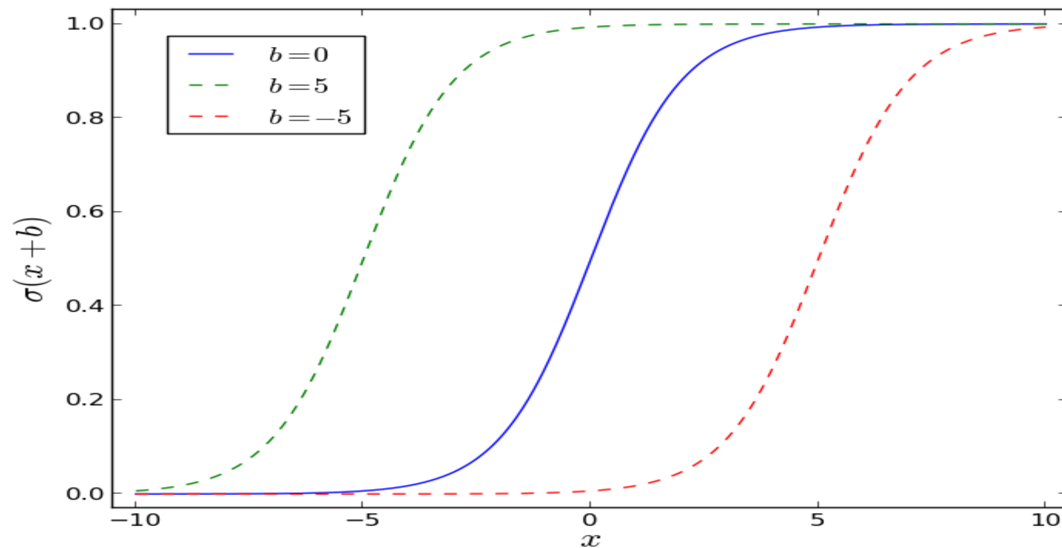
$$y = \frac{1}{1 + e^{-\sum_i \omega_i x_i - b}} \tag{6}$$

I Figur 4 nedan är sigmoid-funktionen skissad med en skalparameter v, $\sigma(vx)$. Den blå heldragna linjen motsvarar $v = 1$ och innebär att ingen skalning sker, den streckade gröna linjen motsvarar $v = \frac{1}{2}$ och den streckade röda linjen motsvarar $v = 10$. Utifrån figuren ses att ju större värde på skalparametern v, desto brantare blir kurvan och den liknar mer och mer stegfunktionen som används i perceptronen. För ett lågt värde på v blir sigmoid-funktionen en nästintill linjär funktion. Skalparametern v kan relateras till normen av vikterna $\|\boldsymbol{\omega}\|$ där $\boldsymbol{\omega}$ motsvarar en vektor med alla vikter, $\boldsymbol{\omega} = [\omega_1, \omega_2, \ldots, \omega_p]$. Ett stort värde på $\|\boldsymbol{\omega}\|$ motsvarar ett stort värde på v och ett litet värde på $\|\boldsymbol{\omega}\|$ motsvarar i sin tur ett litet värde på v.



Figur 4. Här visas sigmoid-funktionen med en skalparameter $\boldsymbol{v}$, $\boldsymbol{\sigma(vx)}$. Den blåa linjen motsvarar att $\boldsymbol{v = 1}$, gröna motsvarar $\boldsymbol{v = \frac{1}{2}}$ och den röda motsvarar $\boldsymbol{v = 10}$. Notera att funktionen blir mycket brantare för ett större värde på skalparametern $\boldsymbol{v}$. Detta kan jämföras med om vikterna är stora i ett nätverk.

I Figur 5 nedan är sigmoid-funktionen skissad med en lägesparameter b. Denna visar effekten av biastermen, b, i ekvation (6). Med ett negativt värde på b förskjuts kurvan åt höger och för ett positivt värde på b förskjuts kurvan åt vänster. Förskjutningen av kurvan åt vänster innebär att ett mindre värde för den viktade summan av in-variablerna behövs för aktivering, och vice versa om kurvan förskjuts åt höger.

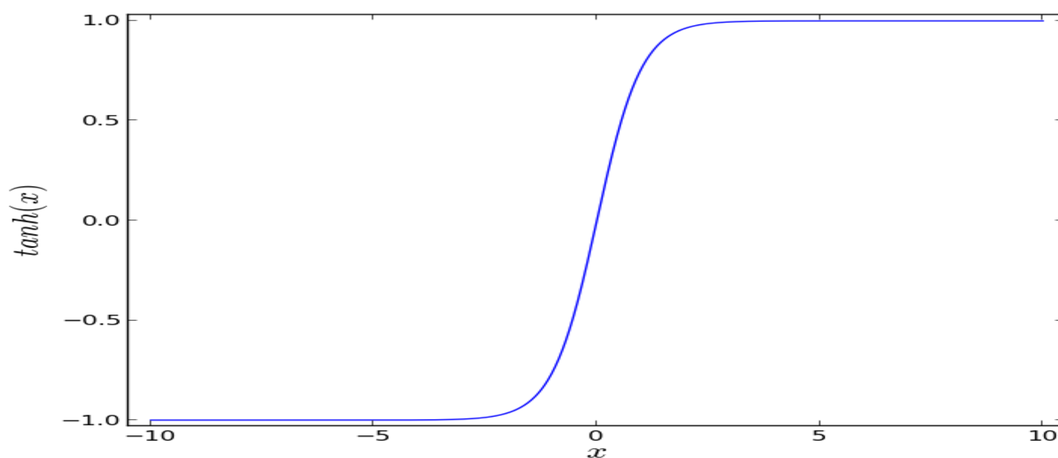


Figur 5. Här visas sigmoid-funktionen med en lägesparameter b. Den blåa linjen motsvarar att $b = 0$, gröna motsvarar $b = 5$ och den röda motsvarar $b = -5$. Detta visar effekten av biasparametern b.

Som ett alternativ till att använda sigmoid-funktionen i ekvation (5) kan denna aktiveringsfunktion ersättas med tangens hyperbolicus och en tanh-neuron erhålls. Tangens hyperbolicus är definierad enligt ekvation nedan.

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

Denna neuronmodell fungerar på samma sätt som sigmoid-neuronen och effekten av vikterna och biasparametern är densamma. Skillnaden är att denna neuron kan returnera värden i intervallet $[-1,1]$. En skiss av tangens hyperbolicus visas i Figur 6 nedan.



Figur 6. Här visas en skiss över tangens hyperbolicus, notera att den kan anta värden i intervallet $[-1,1]$

Tidigare nämndes att neuronerna som används i det sista lagret väljs beroende på om det neurala nätverket avser att användas för klassificering eller regression. Då klassificering utförs är det vanligt att en softmax-funktion användas i det sista lagret. Låt z_k definiera den viktade summan av alla in-variabler inklusive biasparametern till neuron k i det sista lagret, då beräknas softmax-funktionen för denna neuron enligt följande:

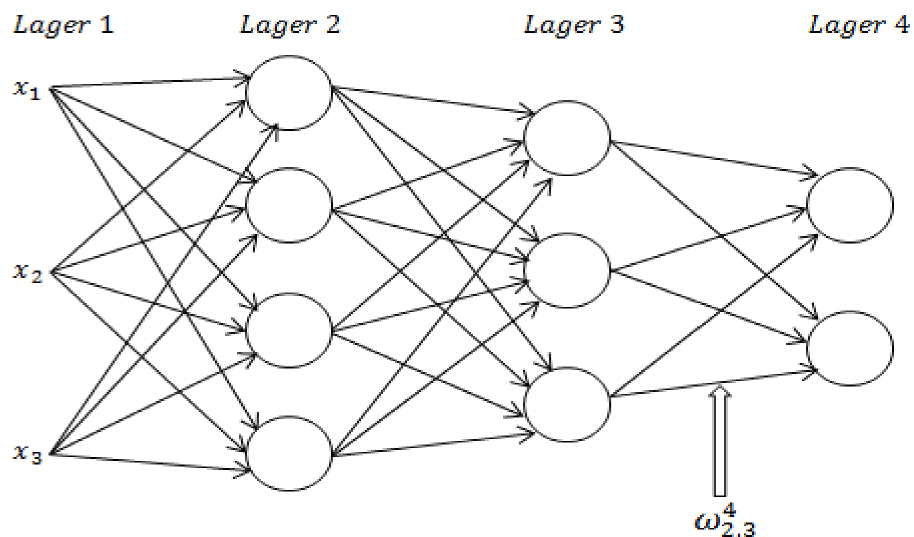
$$\sigma(z_k) = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

Här går summan i nämnaren över alla neuroner i det sista lagret. Denna funktion utför då en normalisering så att varje neuron returnerar ett värde i intervallet $[0,1]$ och summan av alla returnerade värden i det sista lagret summeras till 1. Det skattade klassvärdet bestäms sedan av den neuron som returnerar det högsta värdet.

Om det neurala nätverket istället används till regression kan exempelvis en sigmoid-funktion användas i det sista lagret.

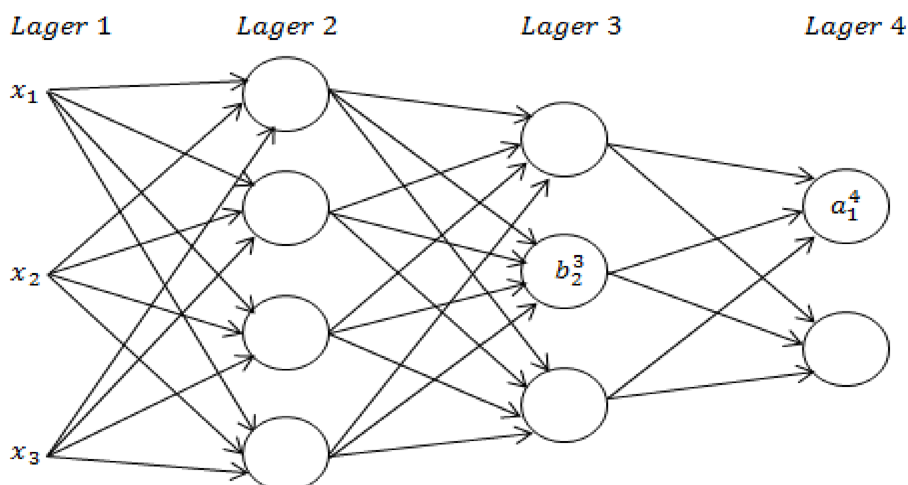
2.1.4. Neuralt nätverk på matrisform

Ett neuralt nätverk beskrivs med fördel på matrisform, det ger framförallt en överblick över vad som sker i nätverket samtidigt som det underlättar för en implementering av ett neuralt nätverk. För att kunna sammanfatta ett neuralt nätverk på matrisform krävs dock en del nya definitioner och notationer. Låt börja med Figur 7 nedan, här refererar $lager$ 1 till in-variablerna, $lager$ 2 och $lager$ 3 refererar till de två dolda lagrena och $lager$ 4 refererar till det lager som returnerar ut-variablerna. Termen $\omega^l_{j,k}$ används som notation till vikten som kopplas mellan ut-datan från neuron nr k i lager $(l-1)$ till neuron nr j i lager l. Här är $\omega^4_{2,3}$ utritad och är då den vikt som kopplar det returnerade värdet från den $3{:}e$ neuronen i det $3{:}e$ lagret till den $2{:}a$ neuronen i det $4{:}e$ lagret.



Figur 7. En skiss av ett neuralt nätverk där vikten $\boldsymbol{\omega^4_{2,3}}$ är utsatt.

En liknande notation som för vikterna används även för att notera bias och ut-datan, även kallat aktivering, från en neuron. Termen b^l_j används för att notera bias i neuron nr j i lager nr l och a^l_j används för det värde som returneras från neuron nr j i lager nr l. I Figur 8 nedan är b^3_2 och a^4_1 utsatt.



Figur 8. En skiss av ett neuralt nätverk där bias b^3_2 och a^4_1 är utsatt.

Med notationerna enligt ovan kan värdet som returneras från neuron nr j i lager nummer l, a_j^l, skrivas som en funktion av aktiveringarna från det tidigare lagret, $(l-1)$, enligt ekvation (7) nedan:

$$a_j^l = \sigma\left(\sum_k \omega_{j,k}^l \cdot a_k^{l-1} + b_j^l\right) \tag{7}$$

Här är funktionen $\sigma(x)$ den aktiveringsfunktion som används för neuron nr j i lager l, denna funktion kan som tidigare nämnts vara olika för olika neuroner. Summan i ekvationen sträcker sig över samtliga neuroner i lager $(l-1)$. Ekvation (7) gäller för att beräkna det värde som returneras från neuronerna i lager 2 och framåt. Det första lagret innehåller och returnerar enligt tidigare definition endast in-variablerna $x_1, x_2, \dots, x_p$, vilket betyder att $a_k^1 = x_k$.

För att formulera beräkningarna som sker i nätverket på matrisform definieras följande vektorer och matriser enligt ekvation (8)-(10):

$$\boldsymbol{a}^l = [a_1^l, a_2^l, \dots, a_m^l]^T \tag{8}$$

$$\boldsymbol{\omega}^l = \begin{bmatrix} \omega_{1,1}^l & \omega_{1,2}^l & \cdots & \omega_{1,m}^l \\ \omega_{2,1}^l & \omega_{2,2}^l & \cdots & \omega_{2,m}^l \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{j,1}^l & \omega_{j,2}^l & \cdots & \omega_{j,m}^l \end{bmatrix} \tag{9}$$

$$\boldsymbol{b}^l = [b_1^l, b_2^l, \dots, b_m^l]^T \tag{10}$$

Här är $\boldsymbol{a}^l$ den vektor som innehåller alla aktiveringar från lager l i nätverket, ovan antas att lager l består av $m\ st$ neuroner och därmed innehåller vektorn $\boldsymbol{a}^l$ $m\ st$ element. På samma sätt är $\boldsymbol{b}^l$ den vektor som innehåller alla biastermer från lager l och antalet element i vektorn är detsamma som antalet neuroner i lager l. Matrisen $\boldsymbol{\omega}^l$ innehåller alla vikter som är länkade mellan lager $(l-1)$ och lager l, här motsvarar kolumn nr i vikterna som kopplar samman neuron nr i i lager $(l-1)$ till samtliga neuroner i lager l.

För att kunna skriva ekvation (7) ovan på matrisform används symbolen $\boldsymbol{\sigma}(\boldsymbol{x})$ för att definiera en vektorfunktion, för vektorn $\boldsymbol{x} = \left[x_1, x_2, \dots, x_p\right]^T$, där funktionsberäkningen $\sigma(x_i)$ sker elementvis för varje element i vektorn $\boldsymbol{x}$, se ekvation (11) nedan:

$$\boldsymbol{\sigma}(\boldsymbol{x}) = \begin{bmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_p) \end{bmatrix} \tag{11}$$

Ekvation (7) ovan kan då skrivas på följande kompakta form:

$$\boldsymbol{a}^l = \boldsymbol{\sigma}(\boldsymbol{\omega}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l) \tag{12}$$

Innan en sammanfattning av beräkningarna i det neurala nätverket på matrisform sker definieras ytterligare en hjälpvariabel, $\boldsymbol{z}^l$, enligt ekvation (13) nedan:

$$\boldsymbol{z}^l \equiv \boldsymbol{\omega}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l \tag{13}$$

$\boldsymbol{z}^l$ är en vektor med lika många element som det finns neuroner i lager l. Varje element i $\boldsymbol{z}^l$ är lika med den viktade summan av in-variablerna, inklusive biastermen, för neuronerna i lager l: $z_j^l = \left(\sum_k \omega_{j,k}^l \cdot a_k^{l-1} + b_j^l\right)$. Vektorn $\boldsymbol{z}^l$ medför att beräkningarna som sker inom nätverket kan sammanfattas på än mer kompakt form, se ekvation (14) nedan, och kommer att vara till stor användning senare, då parametrarna inom nätverket ska bestämmas.

$$\boldsymbol{a}^l = \boldsymbol{\sigma}(\boldsymbol{z}^l) \tag{14}$$

Sammanställning av beräkningar

Beräkningarna som sker i ett neuralt nätverk kan nu sammanfattas enligt följande:

I det första lagret används inga vikter eller biastermer, den första aktiveringsvektorn, $\boldsymbol{a}^1$, består av in-datan $\boldsymbol{x}$:

$$\boldsymbol{a}^1 = \boldsymbol{x} = \left[x_1,\ x_2, \dots, x_p\right]^T$$

För att beräkna den vektor $\boldsymbol{a}^L$ som returneras från det neurala nätverket sker följande beräkningar iterativt från lager 2 fram till det sista lagret L:

$$\boldsymbol{z}^l = \boldsymbol{\omega}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l, \quad \text{för } l = 2,3, \dots, L$$

$$\boldsymbol{a}^l = \boldsymbol{\sigma}(\boldsymbol{z}^l), \quad \text{för } l = 2,3, \dots, L$$

2.1.5. Anpassning av neurala nätverk

Ovan diskuterades teorin bakom de neurala nätverken och hur dem kan användas för klassificering. Det utelämnades dock hur ett neuralt nätverk designas för att på bästa sätt anpassas till ett specifikt klassificeringsproblem. Detta kan delas upp i två steg:

1. Bestämma strukturen i nätverket
2. Bestämma vikter och biastermerna

Att bestämma strukturen i nätverket syftar till att välja antal dolda lager, antal neuroner inom varje lager och vilken typ av neuroner som ska användas[3]. Då strukturen är bestämd är också antalet vikter och biasparametrar givna och det som kvarstår är att bestämma dess numeriska värden. Processen att bestämma värden för vikterna och bias brukar ofta kallas för träning av nätverket. Träningsprocessen kräver att det finns tillgång till träningsdata, det vill säga in-data där klassvärdet redan är känt, och vikter och bias justeras så att nätverket klassificerar träningsdatan med så lågt fel som möjligt.

2.1.5.1. Träning av nätverk

Träning av ett nätverk innebär att bestämma de numeriska värdena för vikter och bias så att modellen anpassas till träningsdatan.

Låt $\boldsymbol{\theta}$ definiera den mängd som innehåller alla vikter och bias. Antag även att det finns en kostnadsfunktion, $C = C(\boldsymbol{\theta})$, som är ett mått på hur mycket modellens skattningar skiljer sig från de sanna klassvärdena i träningsdatan. Målet är då att finna det $\boldsymbol{\theta}$ som minimerar kostnadsfunktionen C.

Det finns många sätt att välja kostnadsfunktionen, C, och i härledningarna av hur kostnadsfunktionen minimeras antags en godtycklig differentierbar kostnadsfunktion som beror av $\boldsymbol{\theta}$. Det kan dock öka förståelsen med ett konkret exempel att referera till, och det är vanligt att summan av felen i kvadrat väljs som kostnadsfunktion, se ekvation (15) nedan:

$$C = \sum_{i=1}^{N} \|y(\boldsymbol{x}_i) - \hat{y}(\boldsymbol{x}_i)\|^2 = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(y_k(\boldsymbol{x}_i) - \hat{y}_k(\boldsymbol{x}_i)\right)^2 \qquad (15)$$

Här är $y(\boldsymbol{x_i})$ det sanna klassvärdet för träningsvektor i och $\hat{y}(\boldsymbol{x_i})$ är det från modellen skattade klassvärdet för träningsvektorn i; $\hat{y}(\boldsymbol{x_i})$ är då en funktion som beror av alla parametrar i $\boldsymbol{\theta}$. I summorna motsvarar N antalet träningsvektorer och K motsvarar antalet möjliga klassvärden.

Gradient Decent

Det finns givetvis många sätt att minimera kostnadsfunktionen, C, med avseende på $\boldsymbol{\theta}$. I detta arbete används gradientbaserad optimering för att bestämma vikter och bias i nätverken, vilket kräver att kostnadsfunktionen, C, är differentierbar. Den gradientbaserade metod som används kallas gradient decent och kan beskrivas i följande steg:

[3] Om det används ett neuralt nätverk där samtliga neuroner inte är fullt sammankopplade, ingår det även att bestämma topologin i detta steg.

Låt C vara en differentierbar funktion i $\mathbb{R}^n$ och $\nabla C(\boldsymbol{\theta})$ beteckna dess gradientvektor i punkten $\boldsymbol{\theta}$:

$$\nabla C(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial C}{\partial \theta_1}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial C}{\partial \theta_n}(\boldsymbol{\theta}) \end{pmatrix}$$

Gradientvektorn anger då vilken riktning från punkten $\boldsymbol{\theta}$ som funktionen C har sin maximala ökning. Som en konsekvens till detta pekar den negativa gradientvektorn, $-\nabla C(\boldsymbol{\theta})$, i den riktning där C har sin maximala minskning. (Böiers, 2010)

Detta utnyttjas för att bestämma vikter och bias i en iterativ process där $\boldsymbol{\theta}$ ges initiala värden och därefter uppdateras genom att förflytta varje parameter inom $\boldsymbol{\theta}$ i den negativa gradientens riktning. Justeringen av $\boldsymbol{\theta}$ i den negativa gradientens riktning fortsätter till dess att lösningen har konvergerat, eller att ett förbestämt villkor är uppfyllt. Detta beskrivs i punktform nedan:

1. Ange initiala vikter och bias, $\boldsymbol{\theta_0}$
2. Uppdatera $\boldsymbol{\theta}$ enligt: $\boldsymbol{\theta_1} = \boldsymbol{\theta_0} - \lambda \cdot \nabla C(\boldsymbol{\theta_0})$
3. Kontrollera om $\boldsymbol{\theta}$ är en tillräcklig lösning, om inte fortsätt till steg 4.
4. Uppdatera $\boldsymbol{\theta}$ tills att lösningen är tillräcklig enligt: $\boldsymbol{\theta_{k+1}} = \boldsymbol{\theta_k} - \lambda \cdot \nabla C(\boldsymbol{\theta_k})$

Här är λ en parameter som kallas steglängd och definierar hur stort steg som tas i den negativa gradientens riktning i varje iteration.

Det är viktigt att veta att Gradient decent endast är en metod för att bestämma lokala optimum, ett globalt optimum kan inte garanteras. Om en funktion med flera optimum undersöks är det valet av de initiala parametrarna som avgör mot vilket optimum algoritmen konvergerar. (Böiers, 2010)

Härledning av partiella derivator

För att bestämma vikter och bias i ett neuralt nätverk med den gradientbaserade optimeringen krävs att gradientvektorn till kostnadsfunktionen kan beräknas. Detta innebär att ett uttryck för de partiella derivatorna till kostnadsfunktionen, C, med avseende på varje vikt, ω, och bias, b måste bestämmas. Detta görs utifrån de matrisformuleringar som introducerades ovan och med antagandet att det nätverk som ska optimeras totalt har $L\ st$ lager:

$$\boldsymbol{a}^1 = \boldsymbol{x} = \left[x_1, x_2, \dots, x_p\right]^T \quad (16)$$

$$\boldsymbol{z}^l = \boldsymbol{\omega}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l, \ l = 2, 3, \dots, L \quad (17)$$

$$\boldsymbol{a}^l = \boldsymbol{\sigma}(\boldsymbol{z}^l), \ l = 2, 3, \dots, L \quad (18)$$

Det antas även att den kostnadsfunktion som används kan skrivas som en funktion av nätverkets ut-data, det vill säga $C = C(\boldsymbol{a}^L)$. De partiella derivator som måste bestämmas är:

$$\frac{\partial C(\boldsymbol{a}^L)}{\partial \omega_{j,k}^l} \tag{19}$$

$$\frac{\partial C(\boldsymbol{a}^L)}{\partial b_j^l} \tag{20}$$

Härledningarna för de partiella derivatorna med avseende på vikterna ω och bias b är snarlika och för att undvika upprepningar utnyttjas kedjeregeln för partiella derivator och följande omskrivning av uttryck (19) och (20) erhålls:

$$\frac{\partial C(\boldsymbol{a}^L)}{\partial \omega_{j,k}^l} = \frac{\partial C(\boldsymbol{a}^L)}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial \omega_{j,k}^l} \tag{21}$$

$$\frac{\partial C(\boldsymbol{a}^L)}{\partial b_j^l} = \frac{\partial C(\boldsymbol{a}^L)}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} \tag{22}$$

Från ekvation (21) och (22) ses att båda ekvationerna innehåller termen $\partial C(\boldsymbol{a}^L)/\partial z_j^l$. För att underlätta härledningarna definieras en ny variabel, δ_j^l, för denna term:

$$\delta_j^l \equiv \frac{\partial C(\boldsymbol{a}^L)}{\partial z_j^l} \tag{23}$$

Det gäller även att element nr j i vektorn $\boldsymbol{z}^l$ enligt definition är lika med:
$z_j^l = \left(\sum_k \omega_{j,k}^l \cdot a_k^{l-1} + b_j^l\right)$. Detta ger att den andra partiella derivatan i högerledet i ekvation (21) och (22) blir:

$$\frac{\partial z_j^l}{\partial \omega_{j,k}^l} = a_k^{l-1} \tag{24}$$

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \tag{25}$$

Ekvation (21) och (22) kan med hjälp av ekvation (23)-(25) formuleras enligt följande:

$$\frac{\partial C(\boldsymbol{a}^L)}{\partial \omega_{j,k}^l} = \delta_j^l \cdot a_k^{l-1} \tag{26}$$

$$\frac{\partial C(\boldsymbol{a}^L)}{\partial b_j^l} = \delta_j^l \tag{27}$$

Det som kvarstår är att bestämma ett uttryck för termen δ_j^l. Först härleds ett uttryck för δ_j^l i det sista lagret, det vill säga då $l = L$, sedan härleds hur δ_j^l kan uttryckas som en funktion av δ_j^{l+1}.

Då kedjeregeln tillämpas på ekvation (23) kan δ_j^L uttrycks som en partiell derivata med avseende på nätverkets ut-data, a_k^L, enligt:

$$\delta_j^L = \sum_k \frac{\partial C(\boldsymbol{a}^L)}{\partial a_k^L} \cdot \frac{\partial a_k^L}{\partial z_j^L} \tag{28}$$

Summan i ekvation (28) går över alla neuroner k i det sista lagret. Här gäller att $\partial a_k^L / \partial z_j^L = 0$ om $k \neq j$ eftersom att värdet som returneras från neuron nr k i lager L endast beror av in-datan till denna, vilken är z_k^L, och ekvation (28) kan skrivas enligt:

$$\delta_j^L = \frac{\partial C(\boldsymbol{a}^L)}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} \tag{29}$$

Från ekvation (18) gäller att $a_j^L = \sigma(z_j^L)$ och genom att låta $\sigma'(x)$ notera derivatan av funktionen $\sigma(x)$ i punkten x kan den partiella derivatan skrivas enligt: $\partial a_j^L / \partial z_j^L = \sigma'(z_j^L)$ och ekvation (29) blir:

$$\delta_j^L = \frac{\partial C(\boldsymbol{a}^L)}{\partial a_j^L} \cdot \sigma'(z_j^L) \tag{30}$$

Nästa steg är att visa hur δ_j^l beräknas för de övriga lagren och kan uttryckas som en funktion av δ_j^{l+1}. Detta innebär att formulera termen $\delta_j^l = \partial C / \partial z_j^l$ som en funktion av $\delta_j^{l+1} = \partial C / \partial z_j^{l+1}$, till detta utnyttjas återigen kedjeregeln enligt:

$$\delta_j^l = \frac{\partial C(\boldsymbol{a}^L)}{\partial z_j^l} \tag{31}$$

$$\Longleftrightarrow$$

$$\delta_j^l = \sum_k \frac{\partial C(\boldsymbol{a}^L)}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{32}$$

$$\Longleftrightarrow$$

$$\delta_j^l = \sum_k \delta_k^{l+1} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{33}$$

I omformuleringen av ekvation (31) till ekvation (32) utnyttjas att z_k^{l+1} är en funktion av z_j^l, och här går summan över alla neuroner, k, i lager $(l+1)$, vilka $\sigma(z_j^l)$ skickas till. I steget mellan ekvation (32) och (33) ersätts den partiella derivatan $\partial C(\boldsymbol{a}^L)/\partial z_k^{l+1}$ med termen δ_k^{l+1}, enligt definitionen i ekvation (23).

Det sista steget för att bestämma ett uttryck för δ_j^l blir att beräkna den partiella derivatan $z_k^{l+1}/\partial z_j^l$. Termen z_k^{l+1} kan skrivas som en funktion av z_j^l enligt:

$$z_k^{l+1} = \sum_j \omega_{k,j}^{l+1} a_j^l + b_k^{l+1} = \sum_j \omega_{k,j}^{l+1} \sigma(z_j^l) + b_k^{l+1} \tag{34}$$

Med ekvation (34) kan den partiella derivatan $\partial z_k^{l+1} / \partial z_j^l$ beräknas till:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \omega_{k,j}^{l+1} \sigma'(z_j^l) \tag{35}$$

Ekvation (33) och (35) ger att δ_j^l kan uttryckas på följande form:

$$\delta_j^l = \omega_{k,j}^{l+1} \cdot \delta_k^{l+1} \cdot \sigma'(z_j^l) \tag{36}$$

Med hjälp av ekvation (26), (27), (30) och (36) kan de sökta partiella derivatorna i uttryck (19) och (20) bestämmas i en iterativ process. Här används ekvation (30) till att bestämma δ_j^L för alla neuroner i det sista lagret, ekvation (36) kan sedan användas till att bestämma δ_j^l för alla neuroner i de resterande lagren. Ekvation (26) och (27) ger de sökta partiella derivatorna när δ_j^l är bestämd för alla neuroner i nätverket. Följande beskrivs i punktform nedan:

1. Bestäm δ_j^L för alla neuroner j i det sista lagret L enligt:

$$\delta_j^L = \frac{\partial C(\boldsymbol{a}^L)}{\partial a_j^L} \cdot \sigma'(z_j^L)$$

2. Sätt $l = L - 1$
3. Bestäm δ_j^l för alla neuroner j i det l:e lagret enligt:

$$\delta_j^l = \omega_{k,j}^{l+1} \cdot \delta_k^{l+1} \cdot \sigma'(z_j^l)$$

4. Om $l > 2$ återgå till punkt 2 för att bestämma δ_j^{l-1}. Då $l = 2$ har δ_j^l bestämts för lager nr 2 fram till det sista lagret, det vill säga för alla lager som innehåller vikter och bias.
5. När δ_j^l har bestämts för alla neuroner i lager 2 och framåt bestäms de partiella derivatorna för vikter och bias enligt:

$$\frac{\partial C(\boldsymbol{a}^L)}{\partial \omega_{j,k}^l} = \delta_j^l \cdot a_k^{l-1}$$

$$\frac{\partial C(\boldsymbol{a}^L)}{\partial b_j^l} = \delta_j^l$$

Det har nu härletts ett system för att beräkna de partiella derivatorna för kostnadsfunktionen C med avseende på alla vikter $\omega_{j,k}^l$ och bias b_j^l i ett nätverk. Eftersom att det önskades att hålla härledningarna så generella som möjligt innehåller de framtagna ekvationerna följande uttryck: $\partial C(\boldsymbol{a}^L)/ \partial a_j^L$ och $\sigma'(z_j^l)$. Dessa måste alltså beräknas då kostnadsfunktionen är vald och alla aktiveringsfunktioner är bestämda. Med antagandet att kostnadsfunktionen väljs

enligt ekvation (15) och låter det skattade klassvärdet vara den vektor som returneras från nätverket, det vill säga $\hat{y}(\boldsymbol{x}_i) = \boldsymbol{a}^L$, erhålles följande kostnadsfunktion:

$$C = \sum_{i=1}^{N} \|y(\boldsymbol{x}_i) - \hat{y}(\boldsymbol{a}^L)\|^2 = \sum_{i=1}^{N}\sum_{j=1}^{J}\left(y_j(\boldsymbol{x}_i) - a_j^L\right)^2$$

Och $\partial C(\boldsymbol{a}^L)/\,\partial a_j^L$ blir i så fall följande:

$$\frac{\partial C(\boldsymbol{a}^L)}{\partial a_j^L} = -2\sum_{i=1}^{N} y_j(\boldsymbol{x}_i) - a_j^L$$

Här är summan över alla träningsvektorer $\boldsymbol{x}_i$.

Och då sigmoid-neuroner används så att aktiveringsfunktionen är en sigmoid-funktion:

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

Är dess derivata följande:

$$\sigma'(z_j^l) = \frac{e^{-z_j^l}}{\left(1 + e^{-z_j^l}\right)^2} = \sigma(z_j^l) \cdot \left(1 - \sigma(z_j^l)\right)$$

Partiella derivator på matrisform

Innan en sammanställning av den gradientbaserade algoritmen för att uppdatera vikter och bias kan redovisas, formuleras först hur de partiella derivatorna kan uttryckas på matrisform så att algoritmen kan skrivas på kompakt form. Till att börja med införs notationen $A \odot B$, för att notera den elementvisa matrismultiplikationen för matris A och B och följande matriser och vektorer definieras:

$$\boldsymbol{\delta}^l = \begin{bmatrix} \delta_1^l \\ \delta_2^l \\ \vdots \\ \delta_m^l \end{bmatrix}$$

$$\nabla_a C = \begin{bmatrix} \partial C(\boldsymbol{a}^L)/\,\partial a_1^L \\ \partial C(\boldsymbol{a}^L)/\,\partial a_2^L \\ \vdots \\ \partial C(\boldsymbol{a}^L)/\,\partial a_m^L \end{bmatrix}$$

$$\boldsymbol{\sigma}'(\boldsymbol{z}^L) = \begin{bmatrix} \sigma'(z_1^L) \\ \sigma'(z_2^L) \\ \vdots \\ \sigma'(z_m^L) \end{bmatrix}$$

Termerna δ_j^l och δ_j^L som behövs för att beräkna de partiella derivatorna i den gradientbaserade optimeringen kan då beräknas enligt:

$$\boldsymbol{\delta}^L = \nabla_a C \odot \boldsymbol{\sigma}'(\boldsymbol{z}^L)$$

$$\boldsymbol{\delta}^l = ((\boldsymbol{\omega}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \boldsymbol{\sigma}'(\boldsymbol{z}^L)$$

Sammanställning för uppdatering av vikter och bias

Det har nu härletts ett uttryck för de partiella derivatorna som behövs till den gradientbaserade optimeringen för att bestämma vikter och bias i ett neuralt nätverk. Algoritmen för att uppdatera vikter och bias kan nu sammanfattas. Detta sker i en iterativ process och här beskrivs hur parametrarna uppdateras i en iteration, k.

1. För varje träningsvektor, $\boldsymbol{x} = [x_1, x_2, \dots, x_p]$, sätt:
   $\boldsymbol{a}^{x,1} = \boldsymbol{x}$

2. För varje lager $l = 2,3, \dots, L$ i nätverket: beräkna $\boldsymbol{z}^{x,l}$ och $\boldsymbol{a}^{x,l}$ enligt:
   $\boldsymbol{z}^{x,l} = \boldsymbol{\omega}^l \boldsymbol{a}^{x,l-1} + \boldsymbol{b}^l$
   $\boldsymbol{a}^{x,l} = \boldsymbol{\sigma}(\boldsymbol{z}^{x,l})$

3. Beräkna $\boldsymbol{\delta}^{x,L}$ för alla träningsvektorer i det sista lagret enligt:
   $\boldsymbol{\delta}^{x,L} = \nabla_a C \odot \boldsymbol{\sigma}'(\boldsymbol{z}^{x,L})$

4. Iterera bakåt i nätverket för att bestämma $\delta^{x,l}$ för alla träningsvektorer i de tidigare lagren enligt:
   $\boldsymbol{\delta}^l = ((\boldsymbol{\omega}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \boldsymbol{\sigma}'(\boldsymbol{z}^l)$

5. För varje lager $l = L, L-1, \dots, L-2$ uppdatera vikter enligt:

$$\boldsymbol{\omega}_k^l = \boldsymbol{\omega}_{k-1}^l - \eta \sum_{x=1}^{N} \boldsymbol{\delta}^{x,l} (\boldsymbol{a}^{x,l-1})^T$$

$$\boldsymbol{b}_k^l = \boldsymbol{b}_{k-1}^l - \eta \sum_{x=1}^{N} \boldsymbol{\delta}^{x,l}$$

2.1.5.2. Bestämma hyperparametrar till neurala nätverk.

Det diskuterades tidigare hur gradientbaserad optimering kan användas för att optimera vikterna och biasparametrarna i ett neuralt nätverk. Däremot diskuterades inget om hur följande bestäms:

- Vilken steglängd som ska användas i den gradientbaserade optimeringen
- Avgöra hur många iterationer som den gradientbaserade optimeringen ska fortgå för att erhålla en bra anpassning till datan men fortfarande undvika överanpassning till träningsdatan.
- Antal dolda lager med neuroner
- Antal neuroner i varje dolt lager
- Vilka aktiveringsfunktioner som ska används i neuronerna

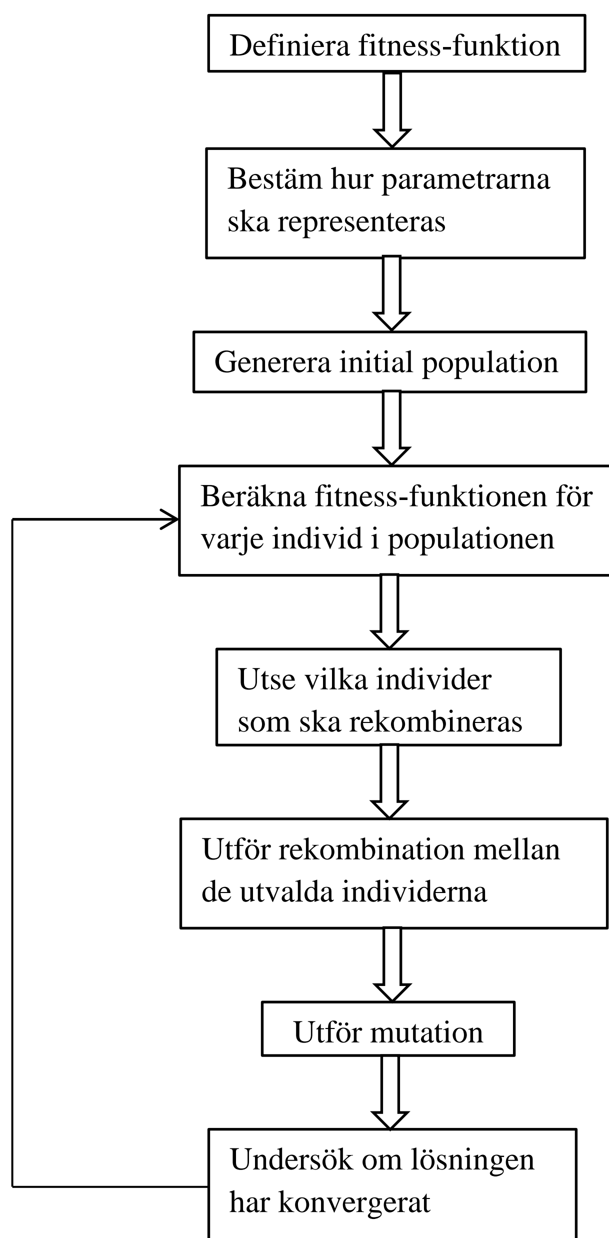
Ovanstående punkter brukar benämnas för hyperparametrar till ett neuralt nätverk. Termen hyperparametrar används för att skilja på de parametrar som används för att definiera en nätverksmodell och de modellparametrar som uppkommer från denna modell. I detta fall refererar modellparametrar till nätverkets vikter och biasparametrar. Hyperparametrarna är problemberoende och måste bestämmas specifikt till varje klassificeringsproblem.

Att bestämma hyperparametrar till ett neuralt nätverk är ett betydligt svårare optimeringsproblem än att justera vikter och biasparametrar i ett nätverk, där gradientbaserad optimering kan användas. Stor komplexitet uppkommer i och med att det är en kombination av både kontinuerliga- och diskreta parametrar som ska optimeras. Parametrarna är även beroende av varandra, exempelvis krävs en annan steglängd i den gradientbaserade optimeringen om det används ett dolt lager med neuroner jämfört med om det används två dolda lager. Detta medför att det inte går att se varje parameter som ett enskilt optimeringsproblem, där man optimerar en parameter i taget medan de andra parametrarna hålls fixa.

Målet med detta arbete är att undersöka om det med hjälp av genetiska algoritmer är möjligt att automatisera denna process, att bestämma hyperparametrar.

2.2. Genetiska algoritmer

Genetiska algoritmer är en optimeringsteknik baserad på principerna från genetik och naturligt urval. Det är en iterativ process där varje möjlig lösning till ett problem ses som en individ och ett antal kandiderande lösningar utvärderas i varje iteration. Varje iteration brukar benämnas för en generation och de lösningar som studeras under en iteration är vår population. Parametrarna som utgör en möjlig lösning representeras i form av en vektor, likt gener i en kromosom och för varje generation skapas en ny population genom rekombination och mutation där de bättre anpassade individerna ges större sannolikhet att föra vidare sin arvsmassa. På detta sätt anpassas populationen i varje generation för att kunna lösa ett optimeringsproblem. En genetisk algoritm för optimering kan beskrivas enligt schemat i Figur 9 nedan. Här genomförs de tre första stegen endast en gång, medan steg 4-8 sker i en iterativ process till dess att lösningen har konvergerat.



Figur 9. Här redovisas ett flöde över hur genetiska algoritmer arbetar för optimering. De tre första punkterna utförs innan den genetiska algoritmen kan initieras, de övriga fem utförs i varje generation.

2.2.1. Fitness-funktion

För att kunna använda genetiska algoritmer till ett optimeringsproblem är det viktigt att det finns ett väldefinierat mått på hur bra en möjlig lösning är. En funktion som ger ett mått på hur bra en specifik löning är definieras på förhand. Denna funktion brukar benämnas för fitness-funktion och relaterar till hur väl anpassad en individ är inom en population.

Fitness-funktionen används i varje generation av den genetiska algoritmen och de bästa lösningarna ges större sannolikhet att överföra sin arvsmassa när nya lösningar skapas genom rekombination.

I detta arbete ska hyperparametrar till neuralt nätverk bestämmas och en fitness-funktion som ger ett mått på hur bra de framtagna neurala nätverken är måste definieras. Det som önskas är ett nätverk som klassificerar den tillgängliga träningsdatan med hög precision och samtidigt undviker överanpassning till denna.

2.2.2. Representation av parametrar

Då en lämplig fitness-funktion har valts är nästa steg att bestämma hur de parametrar som önskas optimeras ska representeras.

I genetiska algoritmer brukar en möjlig lösning till problemet benämnas för en individ eller en kromosom. Varje individ är i sin tur uppbyggd av gener där varje gen motsvarar varsin parameter som ska optimeras. Det är vanligt att varje parameter kodas till en binär sträng av ettor och nollor. I den binära modellen kan varje etta och nolla jämföras med DNA-molekyler som tillsammans bildar en gen. De direkta relationerna till genetiken är en av anledningarna varför den binära representationen används flitigt i implementationer av genetiska algoritmer.

Det finns dock svagheter med en sådan representation. I det fall parametrarna som ska optimeras är realvärda och att det är viktigt att dem kan bestämmas med hög precision krävs långa binära strängar för att erhålla tillräcklig upplösning. Långa binära strängar i kombination med att många parametrar ska optimeras innebär att det krävs mycket av datorns minne för att lagra alla lösningar. (Wu, Tzeng, Goo, & Fang, 2007)

För att undvika problem med att kunna representera parametrar med full upplösning kan en kontinuerlig genetisk algoritm användas. Den kontinuerliga genetiska algoritmen använder ingen kodning av parametrarna utan varje gen är dess parametervärde. Eftersom att den kontinuerliga genetiska algoritmen inte behöver avkoda varje parameter för att beräkna fitness-funktionen i varje generation är det även en snabbare algoritm jämfört med när den binära representationen används. (Randy & Sue Ellen, 2004)

2.2.3. Initial population

Innan den iterativa processen där nya lösningar skapas genom rekombination av tidigare lösningar kan starta; måste en initial population skapas. För att konstruera en initial population krävs dels att storleken på populationen definieras och dels att tillvägagångssättet för att skapa de initiala individerna bestäms.

Storleken på populationen är en av flera parametrar i den genetiska algoritmen som har stor betydelse för dess prestanda och är en parameter som varierar för olika problem. En för liten storlek på populationen kan begränsa algoritmen så att den löper en stor risk att konvergerar mot en suboptimal lösning, en för stor population kan å andra sidan medföra att algoritmen blir ineffektiv och det tar lång tid att finna lösningar. (Diaz-Gomez & Hougen, 2007) Ett vanligt sätt att hantera detta problem är genom så kallad ”bootstrapping”. Det innebär att den initiala populationen väljs till en väldigt stor population, men att den snabbt reduceras till en mer hanterbar populationsstorlek. Detta gör att den initiala populationen samplar en stor del av lösningsrymden och eftersom att den reduceras snabbt blir det inte någon större inverkan på tidseffektiviteten.

Då individerna till den initiala populationen ska genereras är det önskvärt med hög diversitet, det vill säga stor mångfald, inom populationen. Diversitet eftersträvas eftersom att det innebär att det finns fler byggstenar och fler kombinationer att konstruera nya lösningar, genom rekombination. Samtidigt motverkar diversiteten risken att den genetiska algoritmen konvergerar mot en suboptimal lösning.

2.2.4. Selektion

Då en initial population har genererats kan den iterativa process där populationen anpassas för att lösa det givna problemet starta. Den iterativa processen skapar en ny population för varje generation genom naturligt urval, rekombination och mutation.

Det naturliga urvalet sker genom att fitness-funktionen beräknas för varje individ i populationen. Detta används sedan för att avgöra vilka individer som är tillräckligt anpassade för att överleva och ha möjlighet att föra vidare sin arvsmassa.

Här eftersträvas att de individer som är bättre anpassade till problemet ska ges större sannolikhet att överföra sin arvsmassa. Det önskas dock en balans i hur sannolikheterna fördelas. Om de bästa individerna ges för hög sannolikhet finns det risk att dessa snabbt tar över populationen och den genetiska algoritmen konvergerar mot en suboptimal lösning. I det fall de bästa individerna ges för låg sannolikhet resulterar detta i långsam evolution. (Melanie, 1999) Som en konsekvens av detta har många algoritmer för selektion, av individer till rekombination, utvecklats.

En vanlig algoritm brukar kallas för rouletthjulsurval där sannolikheterna för att en viss individ ska väljas är direkt proportionerlig mot dess värde av fitness-funktionen och individerna väljs genom dragning med återläggning. Denna algoritm kan dock orsaka problem om ett fåtal av individerna i populationen har betydligt högre värde på fitness-funktionen än de övriga individerna. Då kan denna metod leda till att den genetiska algoritmen konvergerar för snabbt mot en suboptimal lösning. Det är vanligt att detta problem uppstår i början av den

genetiska algoritmen, då variansen oftast är väldigt hög mellan individerna inom populationen. (Melanie, 1999)

För att undvika problemet som uppstår när variansen är för hög har en liknande metod utvecklats. Denna metod utför också selektionen genom dragning med återläggning, men sannolikheterna är istället proportionerliga till individernas rank inom populationen. Genom att utnyttja individernas rank hålls skillnaderna i sannolikheterna hela tiden konstanta och är oberoende av hur stora de absoluta skillnaderna är mellan olika individer.

En något snabbare algoritm har också utvecklats som brukar kallas turneringsurval. I denna algoritm behöver inte sannolikheterna för alla enskilda individer beräknas. För att välja en individ bestäms först parametern turneringsstorlek, som anger hur många individer som tävlar inom varje turnering. Därefter dras lika många individer som turneringsstorleken helt slumpmässigt från populationen och den bästa individen inom turneringen väljs och får föra sin arvsmassa vidare. När en individ har valts läggs alla individer som deltog i turneringen tillbaks i den gamla populationen så att de kan delta i flera turneringar.

2.2.5. Utförande av rekombination

Skapandet av nya individer sker genom rekombination. På samma sätt som med det naturliga urvalet finns det en mängd med algoritmer som försöker efterlikna den naturliga evolutionen för att utföra rekombination.

En algoritm för rekombination bör framförallt uppfylla följande två villkor (Radcliffe, 1991):

1. Om två individer som delvis består av exakt samma genetiska material utsätts för rekombination, då ska även dess avkommor innehålla detta genetiska material.
   Detta villkor kan jämföras med om två föräldrar båda är blåögda så måste även dess barn, skapade genom rekombination, ha blåa ögon.
2. För två olika individer ska det vara möjligt att genom rekombination skapa en individ som består av genetiskt material från båda föräldrarna.
   Detta villkor kan jämföras med om en förälder har brunt hår och den andra föräldern har blåa ögon. Då måste det vara möjligt att rekombinera dessa för att bilda ett barn som har brunt hår och blåa ögon.

De flesta av de utvecklade algoritmerna utför rekombination mellan två individer för att producera två avkommor. Den vanligaste metoden kallas enpunkts-överkorsning, här väljs en helt slumpmässig överkorsningspunkt som delar de två individerna i två delar och den första delen av varje individ kombineras med den andra delen av den andra individen för att skapa två avkommor. I Figur 10 nedan illustreras hur enpunkts-överkorsning utförs mellan två individer bestående av 9 element.

Det finns dock svagheter i att endast använda en överkorsningspunkt. Bland annat är antalet kombinationer som avkommor kan skapas från de två individerna begränsade samtidigt behandlas vissa element fördelaktigt eftersom att det första- och sista värdet i en individ alltid ges till två olika avkommor. (Melanie, 1999)

För att motverka dessa problem har metoder som använder flera slumpmässiga överkorningspunkter utvecklats där det genetiska materialet som ligger mellan två överkorningspunkter byter plats hos individerna. Se Figur 11 nedan för en illustration över hur rekombination med två överkorsningspunkter utförs.

Även en uniform rekombination är vanlig att använda ur vilken två avkommor skapas genom att varje element i de två ursprungliga individerna byter plats slumpmässigt och oberoende med en förbestämd sannolikhet.

Individ 1	1001 | 10010
Individ 2	0101 | 00101
Avkomma 1	1001 00101
Avkomma 2	0101 10010

Figur 10. Illustration över hur enpunkts-överkorsning utförs på två individer bestående av 9 element.

Individ 1	100 | 1100 | 10
Individ 2	010 | 1001 | 01
Avkomma 1	100 1001 10
Avkomma 2	010 1100 01

Figur 11. Illustration över hur tvåpunkts-överkorsning utförs på två individer bestående av 9 element

2.2.6. Utförande av mutation

Den sista genetiska operatorn som utförs för att anpassa populationen är mutation. Mutationen sker på en individ i taget och utför en slumpvandring i närheten av en individ i dess lösningsrymd. Detta möjliggör att diversiteten kan behållas inom den aktuella populationen och är samtidigt ett sätt att införa nytt genetisk material i populationen. (Chen, 2006), (Randy & Sue Ellen, 2004)

Enligt Melanie Mitchell är det en vanlig syn att mutationen endast används för att undvika att den genetiska algoritmen konvergerar för snabbt mot en suboptimal lösning och ses som en bakgrundsoperator. Hon poängterar att mutationen kan ha större roll än så och att den relativa betydelsen av att använda mutation kontra överkorsning varierar över tiden som den genetiska algoritmen används. Det absolut viktigaste är därför att hitta en bra balans mellan överkorsning och mutation. (Melanie, 1999)

Utförandet av mutation är ofta ganska enkelt, här beskrivs det vanligaste sättet att utföra mutation. I det fall en binär representation används och varje individ består av ettor och nollor utförs mutationen genom att gå igenom varje element i en individ och med en förbestämd sannolikhet byter elementet värde. En etta blir en nolla och en nolla blir en etta. Varje element byter värde oberoende av varandra. Om istället en realvärd representation används är det inte självklart hur ett element kan byta värde. Vanligtvis ersätts det element som ska muteras med ett generat slumptal.

3. Metod

Tillvägagångssättet för att optimera neurala nätverk har i detta arbete delats upp i två optimeringsproblem. Det ena problemet avser att bestämma numeriska värden för vikter och biasparametrar i ett neuralt nätverk då alla hyperparametrar är bestämda. Det andra problemet har som mål att bestämma hyperparametrarna som definierar nätverksmodellen.

För att hantera dessa optimeringsproblem har två algoritmer implementerats. Den ena algoritmen skapar ett neuralt nätverk för givna hyperparametrar, tränar detta och möjliggör prediktion, dels på träningsdata och dels på osedd data. Den andra algoritmen är en genetisk algoritm som syftar till att optimera hyperparametrarna och kommunicerar med nätverksalgoritmen för att få feedback om hur väl en given hyperparameteruppsättning kan användas till prediktion.

3.1. Implementation i Python

Det har varit viktigt att ha full kontroll över de två implementationerna för att dem ska kunna samverka i en feedback loop på bästa sätt. Det har även varit viktigt att ha flexibilitet och erhålla möjlighet att justera implementationerna till fullo för att kunna testa möjliga hypoteser. Det har därför valts att inte använda några färdiga paket i exempelvis MATLAB eller Python för neurala nätverk eller genetiska algoritmer. De båda implementationerna, som beskrevs ovan, har istället skrivits från grunden i Python för detta arbete.

3.2. Introduktion till dataset

I utvecklandet av de implementerade algoritmerna har två klassificeringsproblem använts. I det ena problemet skulle irisblommor delas in i tre grupper beroende på fyra olika mått på dess blad. Det andra problemet bestod av att skilja mellan sex olika tidserier som samtliga bestod av 60 in-variabler. De två problemen beskrivs i detalj nedan.

Iris-data:
Detta problem avser att klassificera tre olika arter av irisblommor (Iris Setosa, Iris Versicolour och Iris Virginica). De beskrivande variablerna som används till detta är längden samt bredden på foderbladen respektive kronbladen. En av arterna är linjärt separerbar från de övriga två arterna, de senare två är inte linjärt separerbara.

Datasetet innehåller 50 exempel av varje art, totalt 150 blommor.

Tidsserie-data:
Detta problem avser att klassificera sex olika tidsserier som samtliga består av 60 mätpunkter. Datasetet innehåller 100 simulerade tidsserier av varje typ, totalt 600 tidsserier. De simulerade tidsserierna listas nedan. Här gäller att $1 \leq t \leq 60$ samt att r ett uniformt fördelat slumptal i intervallet $[-6, 6]$.

1. Likafördelat brus: $y(t) = 30 + r$

2. Cyklisk tidsserie: $y(t) = 30 + r + a \cdot \sin(\frac{2\pi t}{T})$
   Här antar a och T värden mellan 10 och 15 för varje tidsserie

3. Ökande trend: $y(t) = 30 + r + g \cdot t$
   Här antar g värden mellan 0.2 och 0.5 för varje tidsserie

4. Minskande trend: $y(t) = 30 + r - g \cdot t$
   Här antar g värden mellan 0.2 och 0.5 för varje tidsserie

5. Ökande trend med hopp: $y(t) = 30 + r + k \cdot x$
   Här antar x värden mellan 7.5 och 20 för olika tidsserier. $k = 0$ innan tiden t_3 och 1 efter denna tid. t_3 antar värden mellan 20 och 40 för varje tidsserie.

6. Minskad trend med hopp: $y(t) = 30 + r + k \cdot x$
   Här antar x värden mellan 7.5 och 20 för olika tidsserier. $k = 0$ innan tiden t_3 och 1 efter denna tid. t_3 antar värden mellan 20 och 40 för varje tidsserie.

3.3. Algoritm för neurala nätverk

Stort fokus har lagts på att utforma den algoritm som skapar och tränar de neurala nätverken. Det finns många möjligheter och beslut som måste fattas då ett neuralt nätverk ska implementeras. För att genetiska algoritmer ska kunna användas till att bestämma hyperparametrar måste det vara tydligt vilka friheter som finns inom nätverket, exempelvis om det ska finnas möjlighet att använda mer än ett dolt lager och i så fall hur många. Det finns även många beslut som måste fattas för att bestämma numeriska värden på vikter och biasparametrar inom nätverket, dels för att undvika överanpassning till träningsdatan och dels för att effektivisera träningen. Nedan redovisas de beslut som har fattats till implementationen av de neurala nätverken samt vilka hyperparametrar som uppkommer från denna modell.

3.3.1. Strukturen i neurala nätverk

Att bestämma strukturen i ett neuralt nätverk syftar till att bestämma antal dolda lager, antal neuroner i varje lager samt vilka typer av neuroner som ska användas. Allt detta ses i denna nätverksmodell som hyperparametrar vilka önskas optimeras av den genetiska algoritmen.

I detta arbete har det tillåtits strukturer med ett respektive två dolda lager med godtyckligt många neuroner inom varje lager. De typer av neuroner som tillåts användas i de dolda lagren är sigmoid-neuronen och tanh-neuronen, vilka beskrevs i teoridelen. Båda dessa neuroner har fördelen att de använder kontinuerliga aktiveringsfunktioner vilket medför att gradientbaserad optimering kan användas för att träna nätverken.

För att möjliggöra att alla beräkningar i det neurala nätverket ska kunna utföras på matrisform, vilket effektiviserar träningsprocessen, tillåts endast en typ av neuroner i samma dolda lager. Används två dolda lager är det däremot möjligt att det är en typ av neuroner i det första lagret och en annan typ i det andra lagret. I det sista lagret som returnerar skattningen av klassvärdet används uteslutande softmax-funktionen som aktiveringsfunktion.

3.3.2. Träning av nätverken

Då strukturen i ett neuralt nätverk är bestämd kvarstår att bestämma de numeriska värdena på vikter och biasparametrar för att nätverket ska anpassas till ett givet klassificeringsproblem. I detta arbete har den gradientbaserade optimeringsteknik som beskrevs i teoridelen använts för detta ändamål.

Givetvis är denna metod inte det enda alternativet för att träna ett nätverk. Exempelvis skulle en genetisk algoritm kunna utvecklas för att även bestämma dessa parametervärden. Detta har testats men då ett neuralt nätverk består av väldigt många vikter och biasparametrar som måste bestämmas med hög precision blir detta en ineffektiv process i jämförelse med att använda gradientbaserade metoder.

Det finns exempel där memetiska algoritmer har använts för att optimera modellparametrar till neurala nätverk med goda resultat. Memetiska algoritmer liknar genetiska algoritmer, grovt kan skillnaden beskrivas genom att genetiska algoritmer efterliknar koncepten från den biologiska evolutionen medan memetiska algoritmer efterliknar kulturell evolution. I naturen kan vanligtvis inte gener modifieras under en individs livstid, det kan däremot memer. (Petalas & Vrahatis, 2004). Detta används inte i detta arbete på grund av begräsningar i beräkningskraft.

Det kan även tänkas att träningsprocessen kan bli snabbare genom att använda optimering där högre ordning av de partiella derivatorna används. Genom att använda optimeringstekniker som använder Hessian-matrisen, istället för gradienten, skulle ett minimum till kostnadsfunktionen kunna erhållas i ett färre antal iterationer. Det går att härleda uttryck för att bestämma Hessian-matrisen men problemet som uppstår är att Hessian-matrisen blir väldigt stor eftersom att antalet vikter och biasparametrar är många i ett neuralt nätverk. Problemet med en stor Hessian-matrisen är att de optimeringstekniker som använder denna också behöver beräkna inversen till denna matris, vilket är en tidskrävande uppgift då matrisen är stor.

I teoridelen härleddes en algoritm för hur gradientbaserad optimering kan användas för att bestämma vikter och biasparametrar i ett neuralt nätverk. Denna algoritm utgick från att den kostnadsfunktion som skulle minimeras var definierad, att det fanns initiala värden på de parametrar som skulle optimeras samt att det gick att avgöra när en lösning har konvergerat. Hur detta bestäms har stor betydelse för prestandan i ett neuralt nätverk. Nedan redovisas hur detta har valts i detta arbete.

3.3.3. Kostnadsfunktion

I teoridelen gavs som exempel att summan av alla felskattningar i kvadrat kan användas som kostnadsfunkton, se ekvation (37) nedan. Här antogs att $\hat{y}(\boldsymbol{x}_i) = \boldsymbol{a}^L$, det vill säga att skattingen för träningsvektorn i är den vektor som returneras från det sista lagret i det neurala nätverket.

$$C = \sum_{i=1}^{N} \|y(\boldsymbol{x}_i) - \hat{y}(\boldsymbol{x}_i)\|^2 = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(y_k(\boldsymbol{x}_i) - \hat{y}_k(\boldsymbol{x}_i)\right)^2 \tag{37}$$

Att använda ovanstående kostnadsfunktion faller ofta naturligt och används i många algoritmer för att träna neurala nätverk. Det finns dock skäl att använda andra mått som kostnadsfunktion. Åtminstone i det fall då det sista lagret i det neurala nätverket använder en icke-linjär aktiveringsfunktion. Anledningen till detta förklaras då man studerar hur den gradientbaserade optimeringen utförs och hur de partiella derivatorna till denna beräknas. I teoridelen beskrevs att den gradientbaserade optimeringen justerade parametrarna, $\boldsymbol{\theta}$, för varje iteration enligt:

$$\boldsymbol{\theta}_{\boldsymbol{k+1}} = \boldsymbol{\theta}_{\boldsymbol{k}} - \lambda \cdot \nabla C(\boldsymbol{\theta}_{\boldsymbol{k}})$$

Detta betyder att för en enstaka vikt $\omega_{j,k}^{l}$ är värdet som denna justeras med i en iteration proportionerligt mot den partiella derivatan $\frac{\partial C}{\partial \omega_{j,k}^{l}}$. Motsvarande gäller även för biastermerna.

I teoridelen härleddes att de partiella derivatorna av kostnadsfunktionen med avseende på vikter och bias i det sista lagret är proportionerliga mot termen δ_j^L som beräknades enligt:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L)$$

Här är $\sigma'(z_j^L)$ derivatan av aktiveringsfunktionen som används i det sista lagret i punkten z_j^L.

Termen δ_j^L användes även för att bestämma de partiella derivatorna med avseende på vikter och bias i tidigare lager. Detta betyder att om δ_j^L antar ett litet värde så justeras många av vikterna och biasparametrarna i nätverket med små steg.

Då kostnadsfunktionen väljs enligt ekvation (37), beräknas δ_j^L för varje träningsvektor i genom:

$$\delta_j^L = -2 \cdot \left(y_j(\boldsymbol{x}_i) - \hat{y}_j(\boldsymbol{x}_i) \right) \cdot \sigma'(z_j^L) \qquad (38)$$

Från ekvation (38) ovan är δ_j^L proportionerlig dels mot hur stort felet i skattningen är samt mot storleken av derivatan till aktiveringsfunktionen i det sista lagret i punkten z_j^L.

Att δ_j^L är proportionerlig mot hur stort felet är i skattningen innebär att parametrarna som ska optimeras förändras mer under en iteration om skattningen är dålig jämfört med om den är bra. Problemet som uppstår är att om aktiveringsfunktionen i det sista lagret är icke-linjär så är det inte säkert att termen $\sigma'(z_j^L)$ följer samma princip.

I implementationen av de neurala nätverken används den icke-linjära softmax-funktionen som är definierad enligt:

$$\sigma\left(z_j^L\right) = \frac{e^{z_j^L}}{e^{\sum z_k^L}}$$

Här går summan i nämnaren över alla neuroner i det sista lagret. Den partiella derivatan av ovanstående funktion med avseende på z_j^L kan skrivas enligt:

$$\sigma'\left(z_j^L\right) = \sigma\left(z_j^L\right) \cdot \left(1 - \sigma\left(z_j^L\right)\right) \tag{39}$$

Detta betyder att om det skattade värdet skiljer sig mycket från det sanna värdet finns det stor risk att derivatan är väldigt liten och till och med kan anta värdet 0. Detta innebär att det finns risk att träningen av många vikter och biasparametrar kan stagnera som en konsekvens av att derivatan, $\sigma'\left(z_j^L\right)$, blir liten. Används däremot en linjär aktiveringsfunktion i det sista lagret är derivatan konstant och risken att träningsprocessen stagnerar på grund av detta undviks.

För att undvika att behöva använda en linjär aktiveringsfunktion i det sista lagret har andra kostnadsfunktioner undersökts. Valet blev att använda en variant av följande kostnadsfunktion:

$$C = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left[y_k(\boldsymbol{x}_i)\ln \hat{y}_k(\boldsymbol{x}_i) + \left(1 - y_k(\boldsymbol{x}_i)\right)\ln\left(1 - \hat{y}_j(\boldsymbol{x}_i)\right)\right]$$

Här är skattningen $\hat{y}(\boldsymbol{x}_i) = \boldsymbol{a}^L$, det vill säga att skattningen för träningsvektorn i är den vektor som returneras från det sista lagret i det neurala nätverket.

Genom att ersätta $\hat{y}(\boldsymbol{x}_i)$ med $\boldsymbol{a}^L$ kan ovanstående kostnadsfunktion för en träningsvektor i skrivas enligt:

$$C(\boldsymbol{a}^L) = -\frac{1}{N}\sum_{k=1}^{K}\left[y_k(\boldsymbol{x}_i)\ln \boldsymbol{a}^L + \left(1 - y_k(\boldsymbol{x}_i)\right)\ln(1 - \boldsymbol{a}^L)\right]$$

Termen δ_j^L kan då beräknas enligt:

$$\delta_j^L = -\frac{1}{N}\left[\frac{y_j(\boldsymbol{x}_i)}{a_j^L} - \frac{\left(1 - y_j(\boldsymbol{x}_i)\right)}{1 - a_j^L}\right] \cdot \sigma'(z_j^L) \tag{40}$$

Genom att utnyttja att $a_j^L = \sigma(z_j^L)$ och skriva bråktalen under gemensam nämnare kan ekvation (40) ovan formuleras enligt:

$$\delta_j^L = -\frac{1}{N}\left[\frac{y_j(\boldsymbol{x}_i) \cdot \left(1 - \sigma\left(z_j^L\right)\right) - \sigma\left(z_j^L\right) \cdot \left(1 - y_j(\boldsymbol{x}_i)\right)}{\sigma\left(z_j^L\right) \cdot \left(1 - \sigma\left(z_j^L\right)\right)}\right] \cdot \sigma'(z_j^L)$$

Från ekvation (39) ovan är uttrycket som står i nämnaren lika med $\sigma'(z_j^L)$ och uttrycket kan förenklas till:

$$\delta_j^L = -\frac{1}{N}\left[y_j(\boldsymbol{x}_i) \cdot \left(1 - \sigma\left(z_j^L\right)\right) - \sigma\left(z_j^L\right) \cdot \left(1 - y_j(\boldsymbol{x}_i)\right)\right]$$

Genom att ersätta $\sigma(z_j^L)$ med a_j^L och förenklar ytterligare kan uttrycket slutligen skrivas enligt:

$$\delta_j^L = -\frac{1}{N}\left(y_j(\boldsymbol{x}_i) - a_j^L\right)$$

Med denna kostnadsfunktion kommer storleken som varje vikt och biasparameter i det sista lagret justeras med att vara proportionerlig mot storleken av felet av det skattade klassvärdet. Detta innebär att vikterna justeras snabbare desto sämre de klassificerar och att träningsprocessen inte riskerar att stagnera på grund av de olinjära egenskaperna i aktiveringsfunktionen i det sista lagret.

Än så länge har valet av kostnadsfunktion endast motiverats utifrån att träningsprocessen ska vara så tidseffektiv som möjligt. Eftersom att det används ett stort antal vikter och biasparametrar i ett neuralt nätverk är överanpassning ett vanligt problem. Överanpassning innebär att det neurala nätverket börjar modellera brus i träningsdatan istället för att beskriva det underliggande problemet. Detta resulterar i att nätverket kan utföra prediktioner med högre precision på träningsdatan men mister generalitet, och precisionen på osedd data minskar.

En metod som empiriskt har visat sig kunna motverka problemet med att de neurala nätverken överanpassas till träningsdatan är att inkludera en straffterm för stora vikter i kostnadsfunktionen. (Trevor Hastie, 2009), (A. Nielsen, 2014) Detta medför att mindre värden på vikterna kommer att prioriteras när det neurala nätverket tränas. Hur mycket som de stora vikterna ska straffas styrs av en hyperparameter λ.

Strafftermen, $J(\theta)$, som används i detta arbete redovisas i ekvation (41) nedan.

$$J(\theta) = \frac{\lambda}{2N}\sum_{\omega}\omega^2 \tag{41}$$

Här är N antalet träningsvektorer som används och summan går över alla vikter i det neurala nätverket. Med denna straffterm är den slutgiltiga kostnadsfunktion som används följande:

$$C = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left[y_k(\boldsymbol{x}_i)\ln\hat{y}_k(\boldsymbol{x}_i) + \left(1 - y_k(\boldsymbol{x}_i)\right)\ln\left(1 - \hat{y}_j(\boldsymbol{x}_i)\right)\right] + \frac{\lambda}{2N}\sum_{\omega}\omega^2$$

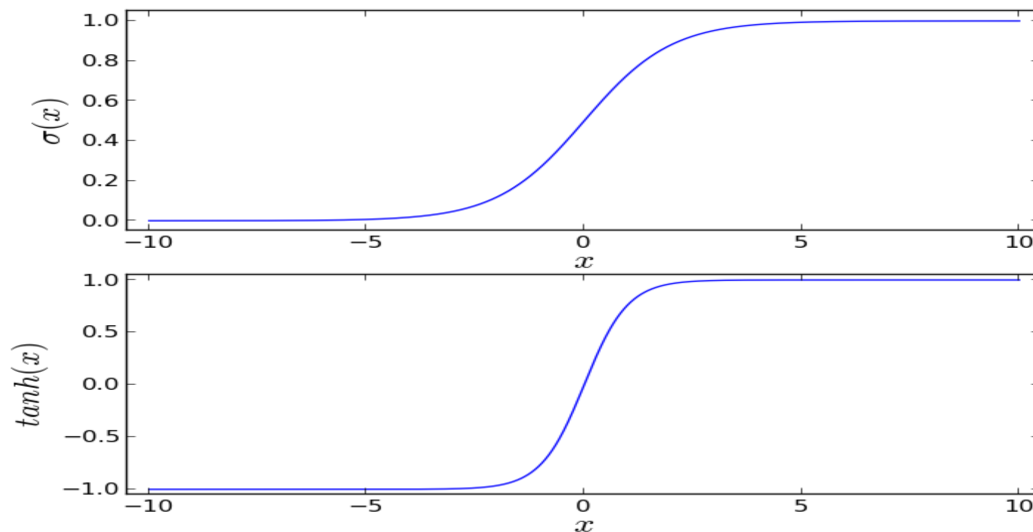
3.3.4. Initiala parametrar och normalisering av data

För att den gradientbaserade optimeringen ska kunna användas till att bestämma numeriska värden för vikter och biasparametrar i ett neuralt nätverk måste dessa parametrar ges initiala värden. Hur dessa bestäms har stor betydelse för hur väl optimeringen fungerar. Eftersom att inverkan av de olika parametrarna i ett neuralt nätverk snabbt blir väldigt komplex är det svårt att på förhand bestämma exakta initiala värden. Det är vanligt att de initiala värdena slumpas från en Gaussisk fördelning. Med detta sagt finns det dock två rekommendationer då initiala parametrar väljs. Den första rekommendationen är att låta alla parametrar anta små värden nära noll. Den andra rekommendationen är att undvika att sätta initiala parametrar till exakt noll.

För att inse varför detta har betydelse studeras hur den gradientbaserade optimeringen beskrevs i teoridelen. Det framgick att då en vikt $\omega_{k,j}^{l}$ eller biasterm b_j^l optimeras är värdet som de justeras med proportionerligt mot termen δ_j^l, vilken beräknas enligt:

$$\delta_j^l = \omega_{k,j}^{l+1} \cdot \delta_k^{l+1} \cdot \sigma'(z_j^l) \tag{42}$$

Här är $\sigma'(z_j^l)$ derivatan av aktiveringsfunktionen i lager l i punkten z_j^l. Det är just denna derivata som kan orsaka problem för den gradientbaserade optimeringen. I Figur 12 nedan redovisas utseendet på de två aktiveringsfunktioner som används i detta arbete.



Figur 12. Den övre bilden visar en skiss över sigmoid-funktionen och den undre en skiss av tangens hyperbolicus. Notera hur kurvorna planar ut för stora absoluta värden på x

I figuren ovan ses att båda kurvorna planar ut för stora positiva respektive negativa värden på z_j^l vilket betyder att derivatan, $\sigma'(z_j^l)$, går mot 0 här. Eftersom att z_j^l är den viktade summan av in-variablerna till neuron j i lager l inklusive biastermen är risken stor att z_j^l antar ett stort absolutvärde om de initiala parametrarna antar stora värden.

Då det inte är känt på förhand vilket värde som önskas returneras från neuron j i lager l blir risken stor att denna neuron exempelvis returnerar värdet 1 då den egentligen borde returnera värdet 0. Om detta sker blir det svår för den gradientbaserade optimeringen att justera de involverade parametrarna på grund av att $\sigma'(z_j^l)$ är nära 0 och dessa parametrar justeras väldigt lite i varje itteration. Det är därför önskvärt att välja initiala parametrar så att termen z_j^l antar ett relativt lågt absolutvärde i de dolda lagren.

Studeras ekvation (42) ovan inses att då parametrar i lager l ska optimeras justeras dessa även i proportion till hur stora parametrarna i lager $(l+1)$ är. Då parametrarna i lager $(l+1)$ är lika med 0 innebär detta att parametrarna i lager l inte justeras med den gradientbaserade optimeringen. Detta medför att man ska undvika att sätta initiala parametrar till just värdet 0.

Som sagt är det önskvärt att låta den gradientbaserade optimeringen initieras så att alla termer z_j^l i de dolda lagren antar värden runt 0. Storleken på z_j^l i det första dolda lagret beror givetvis även på nätverkets in-data, $\boldsymbol{x}$. För att undvika att träningsprocessen sker långsamt på grund av att den data som ska klassificeras antar stora absoluta värden sker normalisering av denna. I detta arbete har all in-data normaliserats så att den antar värden i intervallet $[0,1]$.

I vår implementation har det valts att inte använda slumptal som initiala parametrar utan spridit ut dessa deterministiskt med en blandning av positiva och negativa tal nära noll. Storleken på de initiala parametrarna bestäms i relation till hur många in-variabler som är kopplade till den specifika neuronen för att undvika att den viktade summan av in-variablerna antar ett stort absolutvärde. Anledningen till att de initiala parametrarna väljs deterministiskt är för att underlätta för den genetiska algoritmen att bestämma hyperparametrar. Detta eftersom att värdet på vikter och biasparametrar efter att de har optimerats är beroende av dess initiala värden. Väljs dessa slumpmässigt är det svårt att skilja på om precisionen i det neurala nätverket beror på de slumpmässigt valda initiala parametrarna eller hyperparametrarna till nätverket. Konsekvensen av detta val är dock att modellparametrarna riskerar att konvergera mot ett lokalt minimum.

3.3.5. Avgöra konvergens för ett nätverk

När den gradientbaserade optimeringen används i ett neuralt nätverk måste det finnas ett tydligt villkor för när den ska avbrytas. Det finns många sätt att definiera ett stoppvillkor. Det kan vara att optimeringen fortlöper i ett förbestämt antal iterationer, att lösningen uppfyller ett förbestämt villkor alternativt att lösningen inte förändras under ett antal iterationer, det vill säga att lösningen har konvergerat. I detta arbete används konvergens av lösningen som stoppvillkor.

För att avgöra konvergens studeras hur mycket felet i skattningarna förändras under de femtio senaste iterationerna. Om medeltalet av de femtio senaste förändringarna är mindre än en förbestämd konvergensparameter antas att optimeringsprocessen har konvergerat och den avbryts. Denna konvergensparameter ses som en hyperparameter till vår nätverksmodell och bestäms av den genetiska algoritmen. Valet att avgöra konvergens under just 50 iterationer kommer från tester på de två dataseten som beskrevs i kapitel 3.2, där konvergensen bestämdes på betydligt fler iterationer. Det var då extremt sällsynt att modellparametrarna justerades signifikant efter att konvergens var uppnådd efter 50 iterationer.

3.3.6. Sammanställning av hyperparametrar

Den implementerade nätverksmodellen består av 8 st hyperparametrar. Samtliga parametrar sammanställs nedan.

1. Antal dolda lager i det neurala nätverket
2. Antal neuroner i det första dolda lagret
3. Antal neuroner i det andra dolda lagret (endas om nätverket består av 2 dolda lager)
4. Vilken aktiveringsfunktion som ska användas i det första dolda lagret
5. Vilken aktiveringsfunktion som ska användas i det andra dolda lagret (endas om nätverket består av 2 dolda lager)
6. Steglängden som ska användas i gradientbaserade optimeringen av vikter och bias
7. Storleken på λ som avgör hur mycket stora vikter straffas i kostnadsfunktionen
8. Konvergensparametern

3.4. Implementation av den genetiska algoritmen

Då en genetisk algoritm implementeras finns många metoder som är problemspecifika och beror på det underliggande optimeringsproblemet. I detta arbete har många metoder testats för att utveckla en genetisk algoritm som är anpassad för att optimera hyperparametrar till ett neuralt nätverk. För att undersöka effekten av olika metoder har de två dataset för klassificering som introducerades i kapitel 3.2 använts.

Detta avsnitt syftar åt att beskriva de metoder som används i den genetiska algoritmen och motivera varför de används. Den implementerade algoritmen följer schemat i Figur 9 från teoridelen och de olika momenten beskrivs i detalj nedan.

3.4.1. Val av fitness-funktion

Fitness-funktionen är den funktion som den genetiska algoritmen avser att optimera och används för att jämföra kandiderande lösningar inom populationen. Den utvecklade genetiska algoritmen har som mål att bestämma hyperparametrar till ett neuralt nätverk. Med en given uppsättning av hyperparametrar och ett givet dataset är vår nätverksmodell fullt definierad och vår fitness-funktion kommer att vara ett mått på hur bra denna nätverksmodell är.

Att ge ett mått på hur bra ett framtaget neuralt nätverk passar ett klassificeringsproblem är en icke-trivial uppgift. Givetvis är målet att det neurala nätverket ska klassificera med så hög träffsäkerhet som möjligt. Problemet som uppkommer är att de neurala nätverken lätt blir överanpassade till dess träningsdata. En högre träffsäkerhet i klassificeringen av träningsdatan innebär då inte nödvändigtvis att träffsäkerheten på osedd data ökar.

För att kunna motverka problemet med överanpassning har alla dataset som använts delats upp i tre delar; träningsdata, valideringsdata samt testdata. Träningsdatan är det dataset som används av den gradientbaserade optimeringen för att bestämma vikter och biasparametrar i det neurala nätverket. Prediktioner på träningsdatan samt valideringsdatan tillåts när fitness-funktionen beräknas. Testdatan utelämnas helt under modellframtagandet och används för att modellera osedd data och ger en uppskattning om hur väl den framtagna modellen fungerar på okänd data.

För att bestämma fitness-funktion har de två dataseten som beskrevs ovan använts. Precisionen på klassificering av dels testdatan och dels träning- och valideringsdatan har studerats när olika fitness-funktioner optimeras. Detta används för att avgöra vilken fitness-funktion som erhåller en hög precision på samtliga dataset.

Användes ett mått som endast tar hänsyn till hur bra klassificeringarna blir på valideringsdatan hade den genetiska algoritmen en förmåga att överanpassa de neurala nätverken till detta dataset. Istället användes en fitness-funktion som tog hänsyn till hur nätverken presterade både på träningsdatan och på valideringsdatan. Det visade sig vara fördelaktigt att addera en term som mäter skillnaden i precision på träningsdatan och valideringsdatan. Detta får den genetiska algoritmen att gynna en prediktionsmodell som erhåller liknande prestanda på de båda dataseten och minskar risken för överanpassning till ett av dataseten.

Den fitness-funktion som används till den genetiska algoritmen redovisas i ekvation (43) nedan.

$$fitness = -(f_1 + f_2 + f_3) \tag{43}$$

Här är f_1 och f_2 det procentuella antalet klassificeringar som är fel på träningsdatan respektive valideringsdatan. Termen f_3 är ett mått på hur mycket skattningarna på träningsdatan och valideringsdatan skiljer sig.

För att bestämma f_3 beräknas MSE enligt ekvation (44) nedan för både träningsdatan och valideringsdatan och f_3 sätts till absolutvärdet av skillnaden mellan MSE för träningsdatan och valideringsdatan, se ekvation (45) nedan.

$$MSE = \frac{1}{N}\sum_{i=1}^{N} \|y(\boldsymbol{x}_i) - \boldsymbol{a}^L(\boldsymbol{x}_i)\|^2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(y_k(\boldsymbol{x}_i) - a_k^L(\boldsymbol{x}_i)\right)^2 \tag{44}$$

$$f_3 = abs\big(MSE(träningsdata) - MSE(valideringsdata)\big) \tag{45}$$

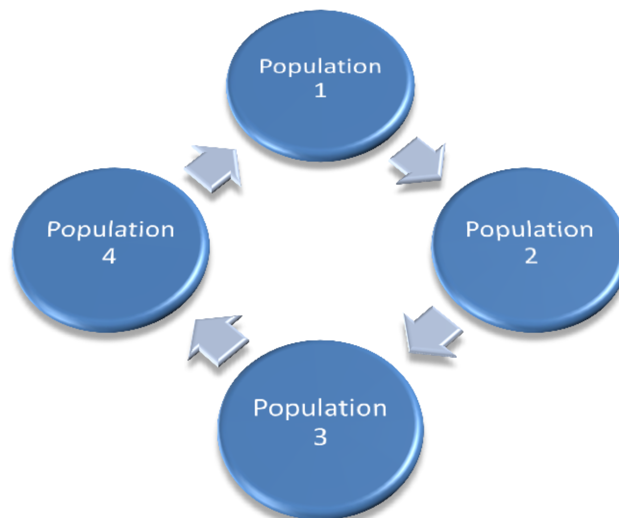
Här är N antalet träningsvektorer för det givna datasetet, K är antalet möjliga klassvärden, $y(\boldsymbol{x}_i)$ är det sanna klassvärdet för träningsvekor $\boldsymbol{x}_i$ och $\boldsymbol{a}^L(\boldsymbol{x}_i)$ är den vektor som returneras från det neurala nätverket med träningsvekor $\boldsymbol{x}_i$ som in-data.

3.4.2. Representation av parametrar

Den implementerade genetiska algoritmen syftar till att bestämma 8 hyperparametrar till ett neuralt nätverk. Fem av parametrarna kan endast anta heltalsvärden och de övriga tre är realvärda parametrar. De tre realvärda parametrarna används av den gradientbaserade optimeringen för att bestämma vikter och biasparametrar i det neurala nätverket och måste bestämmas med stor noggrannhet. För att undvika problemet att kunna representera de realvärda variablerna med full upplösning har en binär representation undvikits till förmån för en realvärd representation. Detta innebär att ett genom till en individ består av 8 gener där varje gen motsvarar varsin hyperparameter som representeras av dess numeriska värde.

3.4.3. Initial population

I teoridelen beskrevs genetiska algoritmer utifrån att en population med kandiderande lösningar utvecklades för att anpassas och optimera det underliggande problemet. I detta arbete har istället en genetisk algoritm som använder 4 subpopulationer som utvecklas parallellt implementerats. Evolutionen inom varje subpopulation sker på samma sätt som om en population används, genom selektion, rekombination och mutation. Samspel mellan subpopulationerna sker genom att två individer från varje subpopulation migrerar till en annan subpopulation var tionde generation. Migrationen följer ett cirkulärt schema där individer från den första subpopulationen migrerar till den andra subpopulationen, individer från den andra migrerar till den tredje och så vidare tills cirkeln sluts med att individer från den fjärde subpopulationen migrerar till den första subpopulationen. Migrationen mellan de 4 olika subpopulationerna förtydligas i schemat som redovisas i Figur 13 nedan.



Figur 13. Schema över hur migrationen mellan de 4 subpopulationerna sker. Migrationen sker genom att individer överförs till populationen närmst till höger i det cirkulära schemat och individer tas emot från populationen närmst till vänster.

Metoden med subpopulationer har använts då den tillåter den genetiska algoritmen att söka efter flera optimum parallellt och samtidigt bibehåller diversitet inom den totala populationen vilket motverkar att den genetiska algoritmen konvergerar för snabbt och därmed riskar att fastna i en suboptimal lösning. (Randy & Sue Ellen, 2004)

För att erhålla de initiala subpopulationerna generas först en population med 2000 lösningar helt slumpmässigt där de 200 bästa individerna väljs ut och fördelas jämnt inom subpopulationerna, det vill säga 50 individer inom varje subpopulation.

3.4.4. Selektion

Evolutionen av varje subpopulation initieras med att bestämma vilka individer som ska ha möjlighet att föra vidare sin arvsmassa, nya individer utvecklas genom rekombination och mutation.

Den implementerade algoritmen låter den bästa individen i varje subpopulation vara en elitindivid för att undvika att dessa förstörs av genetiska operatorer. En elitindivid överlever alltid oförändrad till nästa generation och har samtidigt möjlighet att föra vidare sin arvsmassa till nya individer genom det naturliga urvalet.

Det naturliga urvalet avser att gynna individer som är bättre anpassade till problemet. I den implementerade algoritmen väljs de individer som ska föra sin arvsmassa vidare genom rankbaserad selektion. Detta sker genom att alla individer inom populationen sorteras i förhållande till dess värde på fitness-funktionen. Individer väljs sedan genom dragning med återläggning. En individ på plats n i den sorterade listan dras med sannolikheten P_n som beräknas enligt följande:

$$P_n = \frac{N_{pop} - n + 1}{\sum_{i=1}^{N_{pop}} i}$$

Här är N_{pop} antalet individer i populationen.

3.4.5. Rekombination

De metoder för rekombination som beskrevs i teoridelen är standardmetoder då en binär representation används till genomen i en individ. I det fall en realvärd representation används är det sällan tillräckligt att använda en eller flera överkorsningspunkter och bilda nya individer genom att låta det genetiska materialet byta plats mellan dessa punkter. Eftersom att en gen i den realvärda representationen endast upptar en plats i kromosomen skulle detta innebära att inget nytt genetiskt material bildas genom rekombination, endast olika kombinationer av det initialt slumpvalda genetiska materialet. Den genetiska algoritmen skulle få förlita sig helt på mutation för att införa nya kontinuerliga tal.

Istället har en metod där avkommor bildas som en linjär kombination av två individer använts. (Radcliffe, 1991)

Den använda metoden skapar två avkommor från två individer enligt följande:

Låt gen nummer k för den första individen betecknas med p_k^1 och motsvarnade gen för den andra individen betecknas med p_k^2. I vårt fall ska 8 parametrar optimeras och varje individ består därmed av 8 gener och de två individera kan skrivas enligt:

$$Individ1 = [p_1^1, p_2^1, \dots, p_8^1]$$

$$Individ2 = [p_1^2, p_2^2, \dots, p_8^2]$$

För att bilda två avkommor generas ett slumptal β_k i intervallet $[0,1]$ för varje gen k. Varje gen k till den första avkomman skapas enligt ekvation (46) och motsvarande gen för avkomma två skapas enligt ekvation (47).

$$p_k = \beta_k \cdot p_k^1 + (1 - \beta_k) \cdot p_k^2 \quad (46)$$

$$p_k = (1 - \beta_k) \cdot p_k^1 + \beta_k \cdot p_k^2 \quad (47)$$

Om de två individerna som skapar avkommor har samma värde för gen k, det vill säga $p_k^1 = p_k^2$, kommer båda avkommorna att anta detta värde för gen k. Om istället $p_k^1 \neq p_k^2$ kommer de två avkommorna att anta varsitt värde i intervallet $[\min(p_k^1, p_k^2), \max(p_k^1, p_k^2)]$ för denna gen. Detta betyder att de gener som motsvarar parametrar vilka endast antar heltalsvärden kan utvecklas till realvärda tal. För att undvika detta problem avrundas dessa gener till närmsta heltal efter rekombinationsprocessen.

3.4.6. Mutation

Mutationen är det sista steget för att utveckla subpopulationerna i varje generation. Mutationen utförs på varje individ, exklusive elitindividen, i en population.

Mutation av en individ utförs genom att iterera över alla dess element, det aktuella elementet muteras med en procents sannolikhet, annars bevaras det.

I den implementerade genetiska algoritmen utförs uniform mutation, vilket innebär att om ett element i en individ ska muteras så ersätts detta element med ett uniformt fördelat slumptal.

3.4.7. Samspel mellan de genetiska operatorerna

De implementerade genetiska operatorerna har beskrivits ovan. För att förstå hur dessa samverkar för att utveckla de initiala populationerna ges här en sammanfattning hur evolutionen utförs i varje generation.

För varje generation uppdateras subpopulationerna och ersätts med varsin ny population. En ny population skapas genom att först addera elitindividen från den tidigare populationen, därefter väljs två individer åt gången från den tidigare populationen. De två individerna utsätts för rekombination och skapar två nya avkommor med 90 % sannolikhet. I det fall rekombination utförs adderas avkommorna till den nya populationen annars adderas de två valda individerna. Detta fortsätter tills 50 individer har valts. Då en ny subpopulation har skapats utförs mutation på alla individer, exklusive elitindividen. Utöver detta sker även migration mellan subpopulationerna var tionde generation.

Här har sannolikheten för rekombination satts till 90 % och sannolikheten för mutation till 1 %. Dessa värden har valts som utgångspunkt då de har get bra resultat i många tidigare applikationer av genetiska algoritmer. (V.Kapoor, 2010) Detta är parametrar som kan behövas justeras för olika klassificeringsproblem för att den genetiska algoritmen ska prestera optimalt. På de två dataseten som har använts i utvecklandet av den genetiska algoritmen har andra kombinationer för dessa parametervärden testats utan att det har förbättrat prestandan.

3.4.8. Stoppvillkor

Den genetiska algoritmen fortsätter att utveckla varje subpopulation till dess att ett förbestämt stoppvillkor är uppfyllt. I detta arbete har följande stoppvillkor använts:

Om det högsta uppmätta värdet på fitness-funktionen, beräknat på alla subpopulationer, inte har förändrats under de senaste 100 generationerna så avbryts den genetiska algoritmen. Dessutom tillåts maximalt 1000 generationer.

Även då detta stoppvillkor är uppfyllt finns det givetvis en möjlighet att en bättre individ kan utvecklas, men sannolikheten för detta bedöms som väldigt liten. Villkoret gällande maximalt 1000 generationer har bestämts för att garantera att den genetiska algoritmen avbryts inom rimlig tid. Då den genetiska algoritmen använts på de två tidigare beskrivna dataseten erhölls konvergens betydligt snabbare än 1000 generationer. Villkoret för konvergens om ingen förändring skett under de senaste 100 generationerna har testats fram på de två dataseten som beskrivs i kapitel 3.2. Här testades att låta den genetiska algoritmen att fortsätta längre än

detta, det skedde dock ingen förändring efter att den bästa individen ej förändrats under 100 generationer.

3.4.9. Sammanställning av standardparametrar

Den utvecklade genetiska algoritmen innehåller parametrar som avgör följande; sannolikheten för rekombination, sannolikheten för mutation, populationsstorlek, migrationsstorlek samt migrationshastighet. Detta är parametrar som med fördel kan justeras för olika problem, och kan ha inverkan både på hur snabbt den genetiska algoritmen konvergerar samt hur bra lösningar den konvergerar mot. I detta arbete har dessa parametrar valts till standardvärden som empiriskt har gett bra lösningar på många andra problem och som även har validerats på de två datasetsen som introducerades i kapitel 3.2. Dessa redovisas i Tabell 1 nedan.

Tabell 1. Här redovisas de standardparametrar som används i den genetiska algoritmen. Här är parametern populationsstorlek den totala populationsstorleken för samtliga subpopulationer som används i varje iteration, den initiala populationsstorleken används endast till att sampla den första populationen.

Parameter	Värde
Antal subpopulationer	$4\ st$
Migrationsstorlek	$2\ st$
Migrationshastighet	$1/10$
Initial populationsstorlek	$2000\ st$
Populationsstorlek	$200\ st$
Mutationssannolikhet	$1\ \%$
Rekombinationssannolikhet	$90\ \%$

4. Test av utvecklade modeller

I detta kapitel kommer de utvecklade algoritmerna att appliceras på tre verkliga, det vill säga ej syntetiska, klassificeringsproblem. Samtliga problem är hämtade från *Proben1* som är en samling av problem lämpade för neurala nätverk. Denna problemsamling innehåller även mått på hur bra andra neurala nätverk har presterat på de olika problemen. Dessa resultat kommer att användas i en jämförelse med hur bra de utvecklade algoritmerna utför klassificering på de tre problemen.

4.1. Proben1 (Prechelt, 1994)

Proben1 är en problemsamling skapad i mån att underlätta för utvecklare av neurala nätverk att kunna jämföra sina resultat. Samtliga dataset är indelade i tre delar; träningsdata, valideringsdata samt testdata. Detta möjliggör att användare av dataseten kan erhålla direkt jämförbara resultat, eftersom att det är tydligt vilka problemexempel som används till vad.

Det finns även resultat över hur neurala nätverk där hyperparametrar har bestämts manuellt till varje problem har presterat. För att bestämma dessa hyperparametrar har användaren tillåtit nätverksstrukturer med ett respektive två dolda lager med sigmoid-neuroner. De hyperparametrar som gav bäst precision på valideringsdatan har valts. När användaren har

specificerat hyperparametrar för ett problem tränas 60 neurala nätverk där de initiala modellparametrarna slumpas fram. Därefter mäts precisionen för samtliga nätverk på träningsdatan, valideringsdatan samt testdatan, precisionen redovisas med medelvärde samt standardavvikelse för de 60 anpassade nätverken.

Då precisionen för ett neuralt nätverk ska mätas är det framförallt hur nätverket presterar på testdatan som är intressant. Detta modellerar hur nätverket presterar på osedd data som inte använts för att bestämma dess modellparametrar. I *Proben1* används två olika mått för att mäta precisionen för neurala nätverk. Det procentuella antalet klassificeringsfel mäts på testdatan, och ett mått de kallar ”squared error percentage”, se ekvation (48) nedan, används på samtliga dataset.

$$SEP = 100 \cdot \frac{1}{N \cdot K} \sum_{i=1}^{N} \sum_{k=1}^{K} \left(y_k(\boldsymbol{x}_i) - a_k(\boldsymbol{x}_i) \right)^2 \qquad (48)$$

Här är N antalet exempel som ska klassificeras, K är antalet möjliga klassvärden, $\boldsymbol{x}_i$ är den vektor som ska klassificeras, $y_k(\boldsymbol{x}_i)$ är dess sanna klassvärde och $a_k(\boldsymbol{x}_i)$ är värdet som returneras från neuron nr k i det sista lagret.

För att motverka överanpassning till träningsdatan används en metod som kallas ”early stopping”, detta innebär att träningsprocessen avbryts då felet på valideringsdatan inte längre minskar. I praktiken innebär detta att träningsprocessen fortsätter tills dess att felet på träningsdatan har konvergerat, sedan studeras för vilken iteration som felet på valideringsdatan var som minst, och de modellparametrar som gav det minsta valideringsfelet används. När precisionen på träningsdata redovisas i *Proben1* mäts detta för de modellparametrar som gav lägst fel på träningsdatan, alltså inte nödvändigtvis de modellparametrar som gav det lägsta valideringsfelet och som användes för att mäta felet på testdatan.

4.2. Beskrivning av klassificeringsproblemen

Nedan följer en kort beskrivning av varje klassificeringsproblem som använts. Här redovisas även dimensionen på problemen samt fördelningen mellan träningsdata, valideringsdata samt testdata.

Anledningen till att varje dataset är indelat i 3 tydliga grupper är att erhålla likvärdiga förutsättningar i jämförelse mellan olika utvecklare av neurala nätverk. Eftersom att dataseten innehåller relativt få klassificeringsexempel skulle det kunna vara mer lämpligt att använda samtliga exempel till validering och testdata. Träningsdata skulle istället kunna genereras genom att deformera den tillgängliga datan med lämpligt brus. Men eftersom att dessa tester görs för att erhålla jämförbara resultat, behålls den ursprungliga uppdelningen av träningsdata, valideringsdata samt testdata.

4.2.1. ”Dabetes1”

Diagnos av diabetes. Baserat på personlig information (t ex ålder och antal graviditeter) samt resultat från medicinska undersökningar (t ex blodtryck, BMI, etc.) är målet att avgöra om enskilda Pima indianer har diabetes eller inte.

Datasetet består av 8 in-variabler, 2 klasser och 768 exempel, samtliga in-variabler antar kontinuerliga värden. Uppdelningen mellan träningsdata, valideringsdata samt testdata är följande:

Träningsdata: 384 exempel

Valideringsdata: 192 exempel

Testdata: 192 exempel

4.2.2. ”Cancer1”

Diagnos av bröstcancer. Problemet består av att klassificera en tumör som alltingen godartad eller elakartad baserat på cellbeskrivningar erhållna från cellprover som studerats i mikroskop (finnålsbiopsi).

Datasetet består av 9-invariabler, 2 klasser och 699 exempel, samtliga in-variabler antar kontinuerliga värden. Uppdelningen mellan träningsdata, valideringsdata samt testdata är följande:

Träningsdata: 350 exempel

Valideringsdata: 175 exempel

Testdata: 174 exempel

4.2.3. ”Glass1”

Klassificering av glastyper. Resultat från kemisk analys av glassplitter används för att bestämma glassplittrets ursprung. Den kemiska analysen innehåller information angående procentuellt innehåll från 8 olika grundämnen samt glasets brytningsindex. Denna studie är motiverad utifrån kriminologiska undersökningar, där glassplitter från en brottsplats kan användas som bevismaterial.

Datasetet består av 9-invariabler, 6 klasser och 214 exempel, samtliga in-variabler antar kontinuerliga värden. Uppdelningen mellan träningsdata, valideringsdata samt testdata är följande:

Träningsdata: 107 exempel

Valideringsdata: 54 exempel

Testdata: 53 exempel

4.3. Utförande

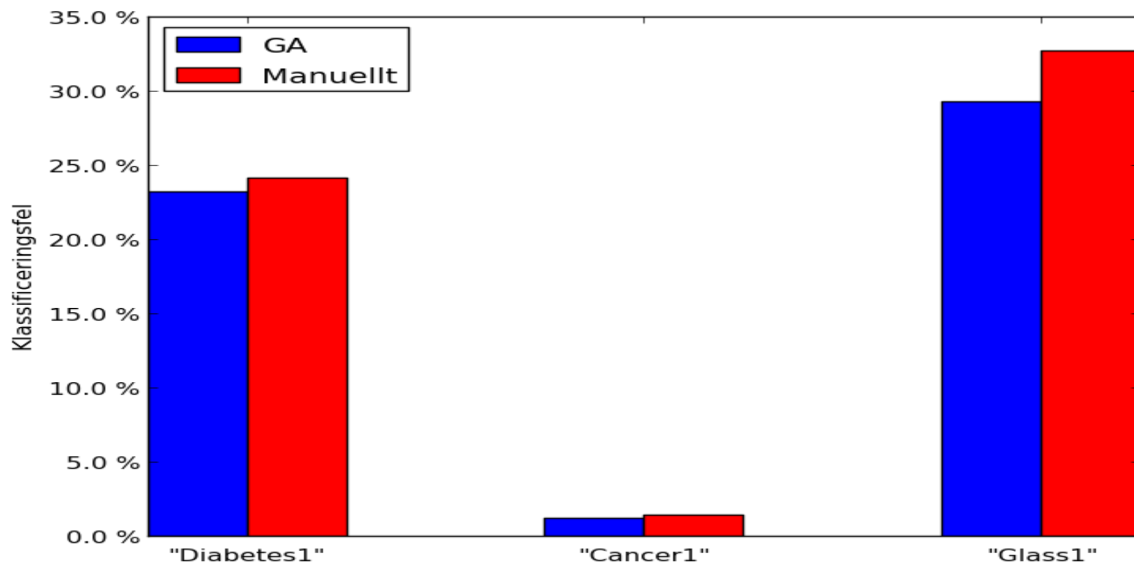
Eftersom att den genetiska algoritmen är en stokastisk optimeringsmetod finns inga garantier att den konvergerar mot samma lösning varje gång den används på ett problem. För att erhålla mer tillförlitliga resultat appliceras den genetiska algoritmen 10 gånger på varje problem. Att den genetiska algoritmen endast applicerades 10 gånger per problem, istället för 60 som används i *proben 1,* beror på begränsade tidsresurser. Det kan ta upp mot 60 minuter för den genetiska algoritmen att konvergera.

Efter en genomgång av den genetiska algoritmen används det neurala nätverk som erhöll högst värde på fitness-funktionen för att mäta precisionen på träningsdatan, valideringsdatan samt testdatan. Resultatet presenteras med medelvärde samt standardavvikelse av de uppmätta precisionerna efter att den genetiska algoritmen har applicerats 10 gånger på ett problem.

De implementerade algoritmerna som används för att optimera parametrar till neurala nätverk använder inte metoden ”early stopping” för att motverka överanpassning till träningsdatan. Istället anpassas modellparametrarna för att ge ett lågt klassificeringsfel på både träningsdatan och valideringsdatan. Därför beräknas precisionen för de neurala nätverken på träningsdatan med samma modellparametrar som används för valideringsdatan och testdatan. Detta medför att den precision som erhålls med vår modell på träningsdatan inte kommer att vara direkt jämförbar med den precision som redovisas i *Proben1*. Precisionen på valideringsdatan och testdatan är däremot direkt jämförbar.

4.4. Resultat

I detta avsnitt redovisas dels de resultat som erhölls då den genetiska algoritmen användes för att bestämma hyperparametrar till neurala nätverk och dels de resultat som är hämtade från *Proben1*. I Figur 14 nedan redovisas medelvärdet av det procentuella antalet klassificeringsfel på testdatan för samtliga problem. De blå staplarna visar resultatet då den genetiska algoritmen användes för att bestämma hyperparametrar och de röda staplarna motsvarar när dessa bestämdes manuellt, hämtade från *Proben1*.



Figur 14. Här redovisas medelvärdet av det procentuella antalet klassificeringsfel på testdatan för samtliga problem. De blåa staplarna motsvarar den modellen där hyperparametrar bestäms med den genetiska algoritmen och de röda staplarna motsvarar när dessa bestäms manuellt, hämtade från *Proben1*.

I den resterande delen av detta avsnitt redovisas resultaten för samtliga dataset inom varje problem. Resultaten redovisas i tabellformat, en beskrivning av varje tabellrad ges nedan:

Träningsdata (SEP): Här redovisas medelvärde samt standardavvikelse då ”squared error percentage”, se ekvation (48), har beräknats på träningsdatan.

Valideringsdata (SEP): Här redovisas medelvärde samt standardavvikelse då ”squared error percentage”, se ekvation (48), har beräknats på valideringsdatan.

Testdata (SEP): Här redovisas medelvärde samt standardavvikelse då ”squared error percentage”, se ekvation (48), har beräknats på testdatan.

Testdata (Klassificering): Här redovisas medelvärde samt standardavvikelse av det procentuella antalet felklassificeringar på testdatan.

4.4.1. ”Diabetes1”

Nedan redovisas de resultat som erhölls då neurala nätverk har använts på klassificeringsproblemet ”Diabetes1”. I Tabell 2 redovisas resultat då den genetiska algoritmen har använts för att optimera hyperparametrar till neurala nätverk. I Tabell 3 redovisas de resultat som är hämtade från *Proben1*, där hyperparametrar har bestämts manuellt.

Tabell 2. Här redovisas resultaten från problemet ”Diabetes1” då den genetiska algoritmen användes för att bestämma hyperparametrar till neurala nätverk.

	Medelvärde	Standardavvikelse
Träningsdata (SEP)	15.03	(0.13)
Valideringsdata (SEP)	15.41	(0.10)
Testdata (SEP)	15.78	(0.08)
Testdata (Klassificering)	23.23	(0.85)

Tabell 3. Här redovisas resultaten från problemet ”Diabetes1” som är hämtade från *Proben1*, där hyperparametrar har bestämts manuellt.

	Medelvärde	Standardavvikelse
Träningsdata (SEP)	14.36	(1.14)
Valideringsdata (SEP)	15.93	(1.04)
Testdata (SEP)	16.99	(0.91)
Testdata (Klassificering)	24.10	(1.91)

4.4.2. ”Cancer1”

Nedan redovisas de resultat som erhölls då neurala nätverk har använts på klassificeringsproblemet ”Cancer1”. I Tabell 4 redovisas resultat då den genetiska algoritmen har använts för att optimera hyperparametrar till neurala nätverk. I Tabell 5 redovisas de resultat som är hämtade från *Proben1*, där hyperparametrar har bestämts manuellt.

Tabell 4. Här redovisas resultaten från problemet ”Cancer1” då den genetiska algoritmen användes för att bestämma hyperparametrar till neurala nätverk.

	Medelvärde	Standardavvikelse
Träningsdata (SEP)	3.49	(0.83)
Valideringsdata (SEP)	2.78	(0.40)
Testdata (SEP)	2.64	(0.56)
Testdata (Klassificering)	1.21	(0.65)

Tabell 5. Här redovisas resultaten från problemet ”Cancer1” som är hämtade från *Proben1*, där hyperparametrar har bestämts manuellt.

	Medelvärde	Standardavvikelse
Träningsdata (SEP)	2.83	(0.15)
Valideringsdata (SEP)	1.89	(0.12)
Testdata (SEP)	1.32	(0.13)
Testdata (Klassificering)	1.38	(0.49)

4.4.3. ”Glass1”

Nedan redovisas de resultat som erhölls då neurala nätverk har använts på klassificeringsproblemet ”Glass1”. I Tabell 6 redovisas resultat då den genetiska algoritmen har använts för att optimera hyperparametrar till neurala nätverk. I Tabell 7 redovisas de resultat som är hämtade från *Proben1*, där hyperparametrar har bestämts manuellt.

Tabell 6. Här redovisas resultaten från problemet ”Glass1” då den genetiska algoritmen användes för att bestämma hyperparametrar till neurala nätverk.

	Medelvärde	Standardavvikelse
Träningsdata (SEP)	3.50	(0.28)
Valideringsdata (SEP)	9.16	(0.36)
Testdata (SEP)	8.07	(0.49)
Testdata (Klassificering)	29.25	(3.51)

Tabell 7. Här redovisas resultaten från problemet ”Glass1” som är hämtade från *Proben1*, där hyperparametrar har bestämts manuellt.

	Medelvärde	Standardavvikelse
Träningsdata (SEP)	7.16	(0.15)
Valideringsdata (SEP)	9.15	(0.12)
Testdata (SEP)	9.24	(0.13)
Testdata (Klassificering)	32.70	(0.49)

5. Slutsatser

Syftet med detta arbete har varit att undersöka om det med hjälp av genetiska algoritmer är möjligt att automatisera processen att bestämma hyperparametrar till neurala nätverk. Den implementerade metoden bedöms utifrån fyra aspekter; robusthet, automatiseringsgrad, generalitet, samt precision.

Då den implementerade algoritmen användes för att klassificera problemen från *Proben1,*så användes samma standardparametrar till den genetiska algoritmen som under testfasen, se Tabell 1.

När standarparametrarna till den genetiska algoritmen används erhålls en fullt automatiserad modell, där användaren endast behöver importera och normalisera datan innan denna algoritm kan användas till att anpassa en prediktiv modell i form av ett neuralt närverk.

Precisionen av den implementerade modellen har i detta arbete utvärderats genom att applicera modellen på tre klassificeringsproblem hämtade från *Proben1*. Resultaten jämfördes även med resultat erhållna från neurala nätverk där hyperparametrar bestämts manuellt. På samtliga problem erhölls ett lägre medelvärde av det procentuella antalet klassificeringsfel på testdatan, då den genetiska algoritmen användes för att bestämma hyperparametrar. Minskningen av medelvärdena är dock relativt liten. Studeras standardavvikelsen av mätningarna då den genetiska algoritmen användes, så är denna större än skillnaderna mellan de två medelvärdena för samtliga problem. Detta indikerar att det inte finns någon statistiskt säkerställd minskning i medelvärdet av klassificeringsfelen, när den genetiska algoritmen användes.

Att resultaten från den implementerade modellen är relativt lika de resultat som erhölls från *Proben1* innebär att denna metod beter sig robust på samtliga testade problem. Det inträffar inte att metoden spikar och ger betydligt bättre resultat på något av de enskilda problemen eller vise versa, att den ger betydligt sämre resultat på något problem.

Då ”Squared error percentage”(SEP) jämförds mellan de två metoderna, för att bestämma hyperparametrar, finns däremot inget tydligt samband. Det skiljer från varje problem vilken metod som ger lägst SEP för de olika dataseten. Som nämnts i kapitel 4 är SEP mätt på träningsdatan inte direkt jämförbar mellan de två metoderna. Resultaten från *Proben1* förväntades ge ett lägre värde på träningsdatan eftersom att detta beräknas med modellparametrar anpassade endast till träningsdatan och därmed antagligen överanpassade till just träningsdatan. Detta samband stämmer för problemen ”Diabetes1” och ”Cancer1”, men inte för problemet ”Glass1”.

Studeras SEP för de två metoderna på problemet ”Glass1” ses att de är nästan identiska på valideringsdatan, medan metoden som bestämmer hyperparametrar med den genetiska algoritmen erhåller ett betydligt lägre medelvärde av SEP på träningsdatan. Detta resulterade även i ett lägre medelvärde av både SEP och det procentuella antalet klassificeringsdel på testdatan.

Att bedöma generaliteten av den implementerade algoritmen är svårt utan att göra ett betydligt bredare test. Eftersom att den implementerade algoritmen i nuvarande form är relativt beräkningsintensiv tar det lång datortid att utföra tester och ett bredare test var därför inte möjligt inom ramen för detta arbete. Men det faktum att medelvärdet av antalet klassificeringsproblem minskade i jämförelse med resultaten från *Proben1* för samtliga klassificeringsproblem, där klassificeringsproblemen varierade mellan olika domäner och antal möjliga klassvärden varierade, indikerar generalitet.

6. Diskussion

Att optimera hyperparametrar till neurala nätverk är ett komplext problem och risken att lösningen konvergerar mot en suboptimal lösning är stor. Det är viktigt att poängtera att även då den implementerade metoden gav till synes tillfredställande resultat på de tre klassificeringsproblemen, så garanterar inte denna metod att hyperparametrarna som tas fram är optimala, i det avseende att det inte finns parametrar som medför högre klassificeringsprecision. Det är snarare troligt att denna algoritm resulterar i lösningar som är suboptimala. Om denna modell alltid hade resulterat i optimala lösningar skulle alla lösningar ge samma antal klassificeringsfel på samma klassificeringsproblem, vilket inte är fallet, detta hade i så fall resulterat i att standardavvikelserna redovisade i kapitel 4 aldrig hade varit skilda från 0.

Även om risken att lösningarna konvergerar mot suboptimala lösningar med den implementerade metoden finns det ändå stor nytta med att använda den. Bara det att den implementerade modellen inte gav sämre resultat på de tre klassificeringsproblemen än resultaten presenterade i *Proben1*, som är resultat framtagna för att kunna användas som riktmärken för utvecklare av neurala nätverk, visar på hur svårt det är att bestämma de optimala parametrarna. Och om resultaten blir likvärdiga när den implementerade modellen används till att bestämma hyperparametrar som när dessa bestäms manuellt, har en stor vinst erhållits genom att det svåra och tidskrävande arbetet för personen som måste optimera dessa manuellt har överförts till arbete i form av datorkraft.

I implementationen av den utvecklade modellen har fokus delvis lagts på att optimera så många beräkningar som möjligt för att minska tidskomplexiteten. När modellen testades på problemen från *Proben1* tog en körning för att erhålla hyperparametrar ca 30-60 minuter, då de flesta beräkningar skedde parallellt på 8 processorer. Skulle större dataset användas resulterar detta naturligtvis i att körningstiden för datorn ökar. I det fall väldigt stora dataset ska användas kommer användaren att behöva ta ställning om det finns tillräckligt med datorresurser för att använda denna modell.

En av de stora fördelarna med att använda genetiska algoritmer för detta ändamål är att det är väldigt enkelt att parallellisera denna process. Detta gör det möjligt att utan några större justeringar skala upp modellen för att utföra beräkningarna på ännu fler processorer, om detta finns tillgängligt.

Som nämndes i det tidigare kapitlet innehåller den framtagna algoritmen parametrar som enkelt kan justeras och som kan ha inverkan på prestandan. Detta medför självklart mer

flexibilitet i modellen och kan rimligtvis medföra att något bättre lösningar kan erhållas om dessa parametrar justeras för varje enskilt problem. Dock leder detta till en konflikt med det ursprungliga syftet, som var att använda genetiska algoritmer för att automatisera processen att bestämma hyperparametrar till neurala nätverk. Eftersom att detta endast förflyttar behovet av optimering av parametrar direkt i neurala nätverket till optimering av parametrar till den genetiska algoritmen. Därför rekommenderas att denna modell användas med de standardparametrar som angivits, och är de som har använts under testarna i detta arbete.

7. Rekommendation om vidare forskning

I diskussionen ovan diskuteras framför allt två svagheter med den implementerade metoden. Den ena är att konvergens mot globalt optimum inte kan garanteras och den andra är modellens tidskomplexitet, som medför att stor datorkraft krävs för att hantera stora dataset. Ett naturligt nästa steg för forskning inom detta område är därför något som kan bidra till en förbättring på dessa punkter.

En åtgärd som kan leda till både snabbare träning av de neurala nätverken samt att modellparametrarna konvergerar mot bättre lösningar är hur de initiala modellparametrarna väljs när dem optimeras med gradient decent. I den implementerade modellen sprids modellparametrarna ut deterministiskt utan hänsyn till vilka värden de borde anta. Ett mer lämpligt sätt att välja dessa kan vara genom intelligenta gissningar om vad modellparametrarna borde vara utifrån den underliggande datan. Detta skulle exempelvis kunna ske genom att använda någon typ av algoritm som bestämmer hur stor inverkan varje invariabel har till klassificeringen, som exempel skulle en random forest kunna användas för detta ändamål.

En annan åtgärd som kan ha stor inverkan på tidskomplexiteten för den implementerade modellen är att tillåta GPU-beräkningar. Genom att utnyttja GPU-beräkningar för att utföra multrismultiplikationerna inom det neurala nätverket kan stora tidsbesparingar göras. Flertalet forskare har testat detta med stor framgång. (Ly, Paprotski, & Yen, 2008), (Luo, Liu, & Wu, 2005) De resulterande nätverken kunde genomgå träningsprocessen många gånger snabbare än traditionella nätverk där beräkningarna skedde sekventiellt genom CPU-beräkningar.

För att öka sannolikheten att den implementerade algoritmen konvergerar mot optimala hyperparametrar krävs förändringar i den genetiska algoritmen. Det naturliga skulle vara att utföra åtgärder som medför att en högre diversitet bibehålls. Detta skulle exempelvis kunna utföras genom en metod som heter ”Fitness Sharing”. Denna metod bibehåller diversitet inom populationen genom att straffa individer som är tillräckligt lika varandra.

8. Litteraturförteckning

A. Nielsen, M. (2014). *Neural Networks and Deep Learning.* Determination Press.

Böiers, L.-C. (2010). *Mathematical Methods of Optimization.* Malmö: Studentlitteratur AB.

Chen, Y.-p. (2006). *Extending the Scalability of Linkage Learning Genetic Algorithms.* Springer Berlin Heidelberg.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, 303-314.

Davidson-Pilon, C. (2015). *Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference.* Crawfordsville, Indiana: Addison-Wesley.

Diaz-Gomez, P. A., & Hougen, D. F. (2007). Initial Population for Genetic Algorithms: A Metric Approach. *International Conference on Genetic and Evolutionary Methods* (ss. 43-49). Las Vegas, Nevada, USA: DBLP.

James, B., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research 13*, 281-305.

Kurt, H., Maxwell, S., & Halbert, W. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 359 - 366.

Loshchilov, I., & Hutter, F. (2016). *CMA-ES for Hyperparameter Optimization of Deep Neural Networks.* eprint arXiv:1604.07269.

Luo, Z., Liu, H., & Wu, X. (2005). Artificial neural network computation on graphic process unit. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. (Volume:1)* (ss. 622 - 626). IEEE.

Ly, D. L., Paprotski, V., & Yen, D. (2008). *Neural Networks on GPUs: Restricted Boltzmann.* Toronto: Department of Electrical and Computer Engineering, University of Toronto.

Melanie, M. (1999). *An Introduction to Genetic Algorithms.*

Petalas, Y., & Vrahatis, M. (2004). Memetic Algorithms for Neural Network Training in Bioinformatics. *In European symposium on intelligent technologies, hybrid systems and their implementation on smart adaptive systems* (ss. 41-46). Aachen, Germany: EUNITE.

Prechelt, L. (1994). *Proben1: A set of neural network benchmark problems and benchmarking rules.* Technical Report 21/94.

Radcliffe, N. J. (1991). Forma Analysis and Random Respectful Recombination. *In Proc. 4th Int. Conf. on Genetic Algorithms.* San Mateo: Morgan Kauffman.

Randy, L. H., & Sue Ellen, H. (2004). *Practical Genetic Algorithms.* wiley-Interscience.

Ronald, J. W., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation Volume 1 Issue 2*, 270-280.

Teuvo, K., & Panu, S. (October 2002). How to make large self-organizing maps for nonvectorial data. *neural Networks 15*, ss. 945–952.

Trevor Hastie, R. T. (2009). *The Elements of Statistical Learning.* Springer Series in Statistics.

V.Kapoor, S. A. (2010). Empirical Analysis and Random Respectful Recombination of Crossover and Mutation in Genetic Algorithms. *International Journal of Computer Applications* , 25-30.

Wu, C.-H., Tzeng, G.-H., Goo, Y.-J., & Fang, W.-C. (2007). A real-valued genetic algorithm to optimize the parameters of support vector machine for predicting bankruptcy. *Expert Systems with Applications*, 397-408.