Teckenigenkänning med maskininlärning

Viktor Sjölind

Kandidatavhandling i datateknik

Handledare: Mats Aspnäs

Institutionen för informationsteknologi

Åbo Akademi

Våren 2018

Referat

Denna avhandling behandlar ämnet maskininlärning och hur man kan använda det för att implementera ett datorsystem som känner igen tecken utifrån bilder. Detta bygger på att man matar systemet med inlärningsdata som anpassar en modell iterativt varefter systemet blir bättre och bättre på att analysera okända data och slutligen kan identifiera tecken med en hög noggrannhet. Ämnet maskininlärning har valts för att det är en teknik som kan användas för att lösa otaliga problem.

Nyckelord

Maskininlärning, neurala nätverk, teckenigenkänning, OCR

Innehållsförteckning

1 Inledning

Identifiering av vad en bild föreställer kan verka enkelt för en människa eftersom även ett litet barn kan förstå om en bild t.ex. föreställer en fisk eller en katt. Detta kan man anta är möjligt på grund av att man omedvetet har skapat sig en inre föreställning om hur ett visst djur ser ut. En fisk har fenor och en katt har svans och ben. Om man ska lösa problemet med att identifiera ting med mjukvara blir det genast svårare eftersom det klassiska sättet att skriva algoritmer är att programmeraren försöker beskriva ett problem så noggrant som möjligt och specifikt för varje enskilt problem som ska lösas.

Maskininlärning erbjuder en annan möjlighet, genom att lösa problem med algoritmer som iterativt analyserar data, kommer mjukvaran fram till gemensamma kännetecken för ting, programmeraren kan då skapa en mera generell algoritm och applicera den på ett stort antal problem. Jämfört med klassisk mjukvaruutveckling var programmerare själva försöker beskriva lösningen till ett problem så specifikt som möjligt. Det är en metod för att lära datorer att gissa sig fram till slutsatser genom att analysera data.

Detta ger en generell lösning som kan appliceras till många möjliga problem inom IT-världen. En algoritm för att klassificera djur eller elektronisk skräppost kan även användas för att känna igen bokstäver och tecken vilket denna avhandling kommer att diskutera.

1.1 Mål

Målet med min kandidatavhandling är att jag ska beskriva och visa hur man kan använda maskininlärning för att identifiera en bild med handskriven text. För att detta ska vara möjligt att lösa med maskininlärning krävs det en stor mängd data för att lära algoritmen. Jag kommer att använda mig av färdiga databanker eftersom att själv samla in data är oerhört tidskrävande och ett stort problem i sig själv. Företag som Google och Amazon håller hårt i sitt data och de har insett att det ger dem stora möjligheter att man har inlärningsdata för sina mjukvaror som utnyttjar maskininlärning. Som exempel kan man ta att Google Images använder

sig av maskininlärning för att känna igen vad en bild föreställer vilket ger företaget en mera exakt tjänst än om man bara hade analyserat beskrivningarna till bilderna på hemsidorna. Spotify är ett annat exempel som analyserar den musik man lyssnar på och tillhandhåller liknande musik åt användaren så att den ska använda tjänsten mera och ta in mera reklam.

2 Metoder och verktyg

Detta kapitel kommer att beskriva vilka metoder som kan användas för maskininlärning och ge förklaringar till grundläggande begrepp och teoretisk bakgrund inom ämnet som man måste förstå.

Jag kommer börja med att beskriva maskininlärning mera generellt och sen dyka ner i textigenkänning och lösa problemet som gavs i inledningen. Meningen är att endast grundläggande kunskaper inom informationsteknik ska krävas för att kunna förstå begreppsförklaringarna och följa med i texten.

2.1 Maskininlärning

Maskininlärning kan beskrivas som att använda ett system med artificiell intelligens att lära sig själv och bli bättre på att lösa problem över tiden, utan att programmeraren uttryckligen beskriver en algoritm för att lösa ett visst problem [1]. Inlärningen börjar med observation av data som är märkt d.v.s. som man på förhand vet vad det föreställer, systemet försöker sen identifiera mönster och likheter i. I praktiken betyder detta att programmet manipulerar förutbestämda variabler tills optimala värden hittas. Inlärningen kan delas in i två övergripande områden av algoritmer: övervakad respektive oövervakad inlärning [2].

2.1.2 Inlärning

Övervakad inlärning innebär att all inlärningsdata är märkt och det måste finnas information om vad det är. Målet är att mjukvaran skall utifrån inlärningsdata lära sig att skapa en funktion som sen ger korrekt resultat på okända data. Exempelvis kunde man mata en algoritm med data av hundar och katter med rätt etiketter varefter mjukvaran efter en tid skulle kunna känna igen omärkta bilder av de tidigare nämna djuren [3].

Märkning av data kräver ofta att människor gör det för hand och är därför tidskrävande. Med *oövervakad inlärning* finns det inget färdigt rätt resultat eller kategorisering av inlärningsdata. Problemet löses då med att programmet

observerar likheter i data och kategoriserar det med klustring. Med oövervakad inlärning kunde man till exempel utifrån en kunddatabas med kundprofiler och deras köp få fram information om vad personer med ett visst kön och ålder konsumerar mest och anpassa reklam utifrån detta [3].

2.3 Mått på likheter

För att mäta likheter mellan data som man analyserar använder man sig av begreppen *korrelation* och *regression*. Korrelation är ett mått på positivt eller negativt samband mellan två faktorer. Exempel ju dyrare bil man har desto rikare är man, detta är stark positiv korrelation. Negativ korrelation skulle istället vara att ju rikare man är desto mindre pengar har man satsat på bilen. Noterbart är att korrelation säger inget om orsaken bakom sambandet.

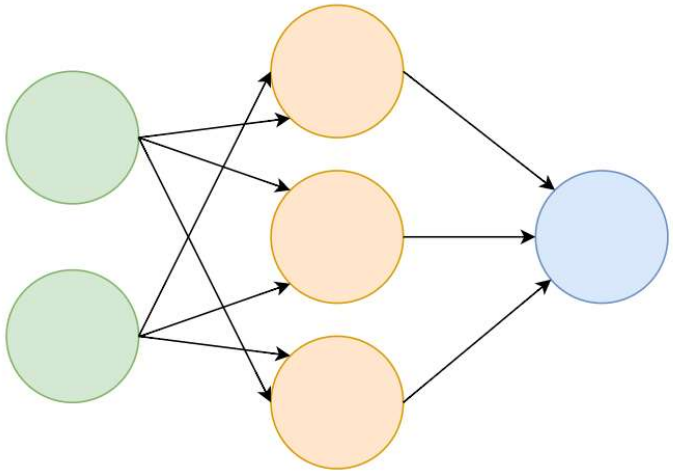
Regression används för att förutsäga en kontinuerlig mängd utdata, linjär regression är att man kan anpassa en rak linje utifrån en mängd diskreta värden enligt den raka linjens ekvation.

Indelning av data kan ske genom *klustring* och *klassificering*. Klustring innebär att identifiera speciella särdrag av data och gruppera dessa utan vidare kunskaper om data i sig. Klassificering i sin tur betyder att man har indata som är med etiketter och kategoriserar dessa i åtskilda kategorier som sen används för att träna algoritmer.

2.4 Algoritmer för inlärning

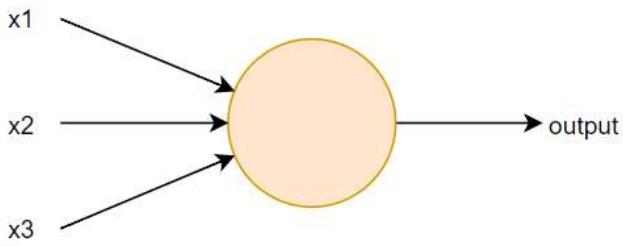
Neurala nätverk är en klass av maskininlärningsalgoritmer [9]. Dessa algoritmer löser problem genom att man först tränar dem med indata vars rätta klassificering är känt, avvikelserna från rätta märkningen används för att justera parametrar och varefter iteration utförs. Detta används för att lära känna igen mönster t.ex. ansikten eller kontokortsbedrägeri genom att känna igen avvikelser från normala aktiviteter. *Modeller* definierar matematiskt sambandet mellan indata och dess märkning.

Neurala nätverk är uppbyggda av lager av artificiella neuroner. Det första lagret benämns input lager med input neuroner. Mellanlagret kallas för det dolda lagret och det sista är output lager med output neuroner.



Input, dolda och output lager

Meningen är att man ska avbilda biologiska neuronnät. Det finns flera typer av artificiella neuroner varav den äldsta är *perceptroner* som är en typ av artificiella neuroner utvecklade på 50-talet. De tar in ett antal binärtal och ger ut ett enskilt binärttal i form av 0 eller 1.



Neuron

För att beräkna output används vikter w som med naturliga tal anger hur viktig varje input x är. Output bestäms av huruvida summan av varje $w \cdot x$ är större eller mindre än tröskelvärdet som också är ett naturligt tal. Perceptron är ett objekt som tar beslut genom att väga bevis [14].

$$output = \begin{cases} 0, & \sum_j w_j x_j \ \leq \ tröskelvärde \\ 1, & \sum_j w_j x_j \ > \ tröskelvärde \end{cases}$$

Dessa kan kombineras i flera nivåer för mera abstrakt och komplext beslutsfattandet. Likaså kan ett output ges till flera perceptroner i nästa nivå. Den matematiska modellen kan även skrivas om enligt bilden, var $b = bias = -tröskelvärde$ [15].

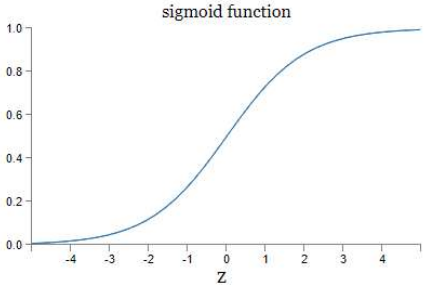
$$output = \begin{cases} 0, & w \cdot x + b \ \leq \ 0 \\ 1, & w \cdot x + b \ > \ 0 \end{cases}$$

För att bättre kunna klassificera data måste man introducera en ny typ av neuroner kallade *sigmoid neuroner*, dessa skiljer sig från perceptroner genom att de kan anta värden mellan noll och ett d.v.s. flyttal och inte enbart binära värden. Sigmoid neuroner tillåter därför en mera specifik evaluering och output.

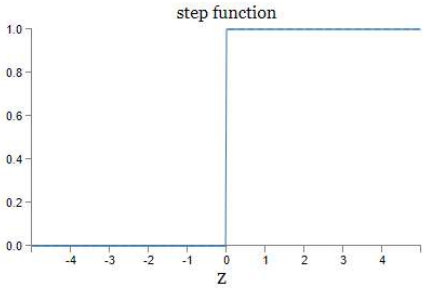
För att output inte ska anta värden mellan -INF och INF, utan istället begränsas till ett värde mellan noll och ett kan man använda en *aktiveringsfunktion* eller *stegfunktion* som normaliserar output och tillsamman med ett tröskelvärde bestämmer om ett neuron aktiveras eller inte. Jag kommer dock att använda *Sigmoid funktionen* $\sigma\,(w \cdot x + b)$ som normaliserar output och ger ett flyttal, funktionen definieras enligt $\sigma(z) = \frac{1}{1+e^{-z}}$. Output fås då av $\frac{1}{1+\exp(-\sum_j w_j x_j - b}$.

Om z är ett stort positivt nummer kommer summan av vikterna, input och bias att anta ett värde kring noll vilket ger funktionen $\sigma(z)$ värde av ungefär ett, med

andra ord om z är stort och positivt fås output ett, likt tidigare nämnda perceptron exempel. Detta ger en stor fördel för att kunna lösa problemet som denna avhandling ställer. Man kan nu matematisk ge ett värde för sannolikheten för att en bild föreställer ett visst tecken och låta en algoritm välja det tecken som är mest troligt att en bild föreställer [4].



This shape is a smoothed out version of a step function:



Figur 1: jämförelse mellan steg och sigmoid funktionen [4]

3 Implementation

I detta kapitel behandlas hur man i praktiken kan lösa avhandlingens frågeställning d.v.s. hur man bygger upp ett neuralt nätverk vars uppgift är att identifiera handskrivna tecken.

3.1 Utförande

För att implementera ett fungerande exempellösning har jag använt mig av Python programmeringsspråket med Numpy biblioteket för matematiska funktioner. Detta motiveras med att Python är välanvänt inom forskning och det tillåter en att snabbt bygga prototyper p.g.a. av dess simpla syntax och struktur – hela programmet är under 200 rader kod. Jag har avsiktligen valt att inte använda mig av specifika högnivåramverk och bibliotek för maskininlärning för att jag ville visa att det går någorlunda enkelt att utveckla ett neuralt nätverk utan att använda sig av verktyg som TensorFlow eller scikit och ändå få ett gott resultat. Likaså ville jag även försöka lära mig att förstå hur neurala nätverk är uppbyggda från grunden.

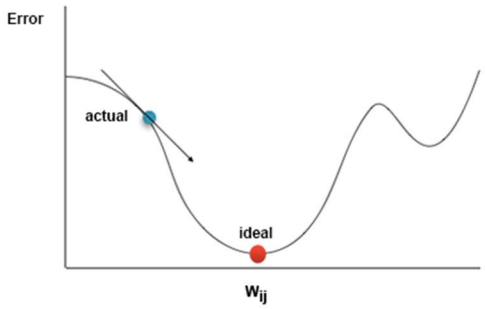
3.2 Inlärningsdata och evaluering

För att träna modellen kommer jag att använda mig av MNIST databasen som innehåller 60 000 centrerade handskrivna gråskalebilder av siffror i formatet 28x28 pixlar. Det neutrala nätverket kommer bestå av tre lager: input, dolt och output lager. Varje träningsinput representeras av en $28 \cdot 28 = 784$ dimensionell kolumnvektor. Det sannolikaste resultatet fås av en funktion $y = y(x)$ vilket är en tiodimensionell radvektor var var enskilda element representerar sannolikheten för att indata är en specifik siffra t.ex. $y = (0, 0.1, 0, 0.2, 0, 0, 0, 0.9, 7, 0)$ vilket skulle tyda på att den analyserade siffran med stor sannolikhet är en åtta.

För att evaluera hur mycket indata påminner om en viss siffra använder jag nedanliggande kvadratiska kostnadsfunktionen vilket estimerar hur modellen presterar:

$$C(w,b) = \frac{1}{2n}\sum_{x} ||y(x) - a||^2$$

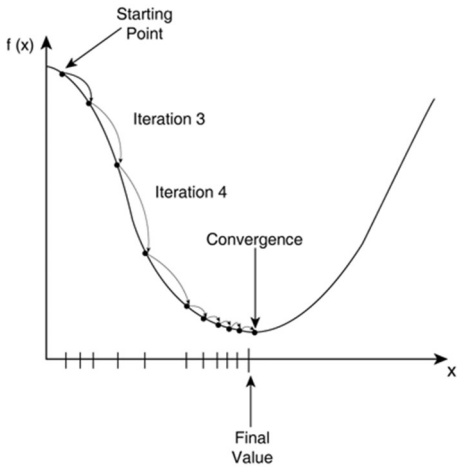
I formeln anger n det totala antalet träningssampel, summan över enskilda sampel, w alla vikter i nätverket, b alla biaser, a aktiveringsfunktionens output vektor för input x. Vid en lyckad identifiering kommer C att ge ett positivt värde nära noll. Detta betyder att problemet egentligen går ut på att hitta värden för vikt och bias som minimerar kostnaden, vilket jag kommer att lösa med en version av gradient nedstigning [16] [17]. Kostnadsfunktionen kan visualiseras enligt Figur 2.



Figur 2:exempel på kraftig och ideal felnivå av kostnadsfunktion [4]

3.3 Gradient nedstigning

Inom matematiken beskrivs gradient som en multivariabel generalisering av derivatan, det vill säga derivata för funktioner baserade på flera variabler. Inom maskininlärning används gradient nedstigning av modellen för att justera vikterna vid varje iteration (epok) av inlärningsdata för att minimera kostnadsfunktionen.



Figur 3: iterativ minimering [5]

Det kan vara väldigt tidskrävande för algoritmen att gå igenom all inlärningsdata. Ett effektivare sätt för att minimera funktionen är stokastisk gradientnedstigning som baserar sig på att vikterna uppdateras efter varje enskilt inlärningssampel. Ett mellanting bland dessa är mini-batch gradient nedstigning vilket jag kommer att använda mig av. Detta innebär att istället för att uppdatera modellen på en eller alla inlärningsdata sampel beräknar man gradient på $1 < k < n$ inlärningssampel som väljs ut slumpmässigt vid början av varje epok. I nästa kapitel analyseras resultatet av olika mini-batch storlekar.

```
model = initialization(...)
n_epochs = ...
train_data = ...
for i in n_epochs:
        train_data = shuffle(train_data)
        X, y = split(train_data)
        predictions = predict(X, model)
        error = calculate_error(y, predictions)
        model = update_model(model, error)
```

Figur 4: pseudokod för gradient nedstigning [6]

3.4 Bakåtspridning

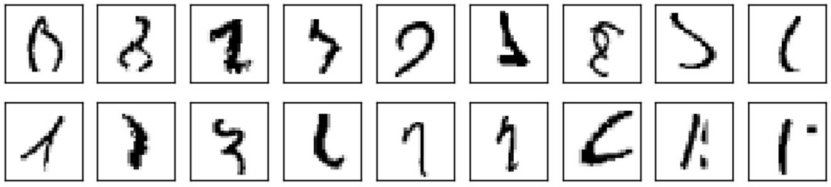
Är en algoritm för att beräkna enskilda gradienter utifrån kostnadsfunktionen genom att arbeta sig bakåt i närverket. När det neurala nätverket ger fel output beräknas derivatan av felnivån först utgående från output lagret. Eftersom detta lager baserar sig på det dolda lagret måste samma beräkning även utföras där, algoritmen arbetar sig bakåt vilket även dess namn anspelar på [7].

Algoritmens huvudsakliga funktion är partiell derivering av kostnadsfunktionen med avseende på vikten och bias vilket berättar hur snabbt och i vilken riktning kostnaden ändrar i förhållande till justering av vikt och bias.

$$\frac{\partial C}{\partial w^2} = \frac{\partial \sum \frac{1}{2}(y - \hat{y})^2}{\partial w^2}$$

4 Diskussion

För att testa mjukvaran kördes den med varierande inställningar på 50 000 bilder av MNIST inlärningsdata varefter nätverket evalueras på 10 000 valideringsbilder. Värt att notera är att världsrekordet i identifiering av MNIST bilderna utfördes av Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus år 2013. Deras implementation kunde uppnå korrekt resultat på 9979 utav 10 000 bilder. De 21 bilder som inte kunde identifieras av deras implementation är sådana som även för människor är svåra att tyda [11].



Figur 5: de 21 bilder som världsrekodsimplementationen inte lyckades tyda

Min implementation baserar sig på Neural Networks and Deeplearning ebookens implementation [8] och med den kommer jag upp i korrekt resultat på ca 9500 bilder med en variation på +- 100 bilder beroende på inställningar och slumpen som man kan anta spelar en roll eftersom diverse slumpgenerator funktioner används i implementationen. Inlärningstakten var satt mellan 1.0 och 3.0 medans antalet epoker låg mellan 30 och 100.

Optimeringsalgoritm	Stokastisk gradient nedstigning
Sampelstorlek	10
Inlärningstakt	3.0
Epoker	100
Igenkänningsresultat	95.4%

Optimeringsalgoritm	Stokastisk gradient nedstigning
Sampelstorlek	50
Inlärningstakt	3.0
Epoker	100
Igenkänningsresultat	94.7%

Figur 5: inställningar och resultat

Utifrån detta kan man dra slutsatsen att det inte räcker med att bara justera nätverkets inställningar för att förbättra resultatet med någon större procentenhet utan man måste ta till andra medel.

Avbildar man kostnadsfunktionen av epoker i en graf ser man också att inlärningen kan skilja sig avsevärt bara genom att ändra inställningarna men resultatet efter ett en stor mängd epoker kommer ändå att vara likadant. Inlärningen behöver inte följa ett tydligt mönster.

4.1 Möjliga förbättringar

Mjukvaran kan som noterat förbättras både genom bättre prestanda och högre igenkänningsresultat. Lösningar till detta kunde vara att skriva om programmet i ett annat programmeringsspråk, öka mängden inlärningsdata, analysera hur olika inlärningsalgoritmer påverkar och att optimera de nuvarande algoritmerna. Fokus kommer nu dock att vara på att optimera den befintliga implementationen och försöka förbättra den.

4.1.1 Cross-entropy kostnadsfunktionen

Om inlärningen är långsam betyder det att den partiella deriveringen är liten eftersom det är den funktionen som ger nya värden för vikten och bias och detta är hur neuroner lär sig. Problemet är då att förstå varför deriveringen ger små värden. Ser man på sigmoid funktionens graf kommer det fram att dess kurva är

väldigt plan när neuronens output närmar sig ett och derivatan är som störst med input noll [10]. Derivatan av kostnadsfunktionen med avseende på vikten och bias kan skrivas om enligt:

$$\frac{\partial C}{\partial w} = (a - y)\sigma'(z)x = a\sigma'(z)$$

$$\frac{\partial C}{\partial w} = (a - y)\sigma'(z)x = a\sigma'(z)$$

Var $a = \sigma(z),\ z = wx + b$

Därifrån härstammar problem med långsam inlärning d.v.s. när neuronens output är nära värdet ett.

En möjlig lösning för att snabba upp inlärningen är att byta ut den kvadratiska kostnadsfunktionen mot cross-entropy kostnadsfunktionen. Likt tidigare baserar sig också cross-entropy kostnadsfunktionen på fallet att man har en radvektor var enskilda element representerar sannolikheten för att indata är en specifik siffra. Cross-entropy Output från ett neuron är också som tidigare $a = \sigma(z)$ var $z = \sum_j w_j x_j + b$.

Denna kostnadsfunktion bygger på att under inlärningen är det rätta svaret känt och detta kan utnyttjas på kolumnvektor datastrukturen genom att ”styra” algoritmen i riktning mot rätt resultat. Cross-entropy kostnadsfunktionen definieras enligt:

$$C = -\frac{1}{n}\sum_x [y \ln a + (1 - y)\ln(1 - a)]$$

Summan av träningsdata och önskade resultatet.

Utifrån denna formel fås att om träningsdata output är nära det önskade resultatet kommer cross-entropy att ligga kring noll.

Från denna formel är det möjligt att definiera en partiell derivata formel som ger ett större värde när felnivån är större, $\sigma'(z)$ upphävs. Detta är vad som skiljer cross-entropy från kvadratiska kostnadsfunktionen [12].

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z) - y)$$

Hastigheten som vikter ändras kontrollerad av $\sigma'(z) - y$

4.1.2 Förbättringar på inlärningsdata

För att analysera hur mängden inlämningsdata påverkar resultatet kördes mjukvaran även med träningsdata av olika storlekar på mängden bilder och följande parametrar:

Optimeringsalgoritm	Stokastisk gradient nedstigning
Sampelstorlek	10
Inlärningstakt	3.0
Epoker	100
Igenkänningsresultat	74.4%
Träningsdata storlek	1000

Optimeringsalgoritm	Stokastisk gradient nedstigning
Sampelstorlek	10
Inlärningstakt	3.0
Epoker	100
Igenkänningsresultat	91.7%
Träningsdata storlek	10 000

Optimeringsalgoritm	Stokastisk gradient nedstigning
Sampelstorlek	10
Inlärningstakt	3.0
Epoker	30
Igenkänningsresultat	94.2%
Träningsdata storlek	25 000

Igenkänningsresultatet blev sämre med mindre träningsdata och därför kan man dra slutsatsen att mera inlärningsdata leder till bättre igenkänningsresultat i viss mån, skillnaden mellan 50 000 bilder och 25 000 är en ungefär procentenhet bättre resultat. Detta är även en balanseringsfråga, större mängd inlärningsdata ger bättre resultat men kräver mera datorkraft och införskaffning av inlärningsdata är även tidskrävande och dyrt. *Överanpassning* är också något man måste ta hänsyn till när man ökar mängden träningsdata, detta innebär att modellen inte utvecklas mera fastän träningen fortsätter. Värsta fallet är att modellen bara lär sig att känna igen träningsdata med 100% korrekthet och sen underpresterar när mjukvaran körs på testdata [13].

Litteraturförteckning

[1] W. L. Hosch, ”Britannica,” [Online]. Available: http://www.britannica.com/EBchecked/topic/1116194/machine-learning. [Använd 20 02 2018].

[2] A. Geitgey, ”Medium,” [Online]. Available: https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471. [Använd 18 02 2018].

[3] L. Tagliaferri, ”Digital Ocean,” Digital Ocean, [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-machine-learning. [Använd 15 02 2018].

[4] J. Fox, ”Neural-Networks-101,” 06 06 2017. [Online]. Available: https://github.com/cazala/synaptic/wiki/Neural-Networks-101. [Använd 21 03 2018].

[5] C. McDonald, ”Towards Data Science,” Medium, 27 11 2017. [Online]. Available: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220. [Använd 21 03 2018].

[6] J. Brownlee, ”Machine Learning Mastery,” 21 07 2017. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/. [Använd 21 03 2018].

[7] D. G. o. S. Shih, ”The official blog of machine learning at Berkeley,” 04 02 2017. [Online]. Available: https://ml.berkeley.edu/blog/2017/02/04/tutorial-3/. [Använd 24 03 2018].

[8] M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[9] D. Korbut, ”Medium,” 26 08 2017. [Online]. Available: https://blog.statsbot.co/machine-learning-algorithms-183cc73197c. [Använd 38 03 2018].

[10] N. M. Vega, ”Prismalytics,” 27 07 2011. [Online]. Available: http://prismalytics.com/2011/07/27/logistic-regression-machine-learning/. [Använd 29 03 2018].

[11] L. Wan, M. Zeiler, S. Zhang, Y. LeCun och R. Fergus, *Regularization of Neural Network using DropConnect.* [Performance]. International Conference on Machine Learning(ICML), 2016.

[12] R. DiPietro, 02 05 2016. [Online]. Available: https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/. [Använd 01 04 2018].

[13] ”EliteDataScience,” 07 09 2017. [Online]. Available: https://elitedatascience.com/overfitting-in-machine-learning. [Använd 01 04 2018].

[14] M. Humphrys, ”School of Computing. Dublin City University,” [Online]. Available: http://computing.dcu.ie/~humphrys/Notes/Neural/single.neural.html. [Använd 02 04 2018].

[15] N. Udyavar, ”Towards Data Science,” Medium, 03 27 2017. [Online]. Available: https://towardsdatascience.com/a-beginners-guide-to-neural-networks-part-two-bd503514c71a. [Använd 02 04 2018].

[16] M. Lachlan, ”Medium,” 10 01 2018. [Online]. Available: https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5fe69815fd. [Använd 01 04 2018].

[17] C. McCormick, 04 03 2014. [Online]. Available: http://mccormickml.com/2014/03/04/gradient-descent-derivation/. [Använd 02 04 2018].