jamk.fi

Tekoäly ja koneoppiminen

Jonne Okkonen, TTV18S3
Joonas Niinimäki, TTV18S3

Harjoitustyö
Tietorakenteet ja algoritmit, Sampo Kotikoski
8.12.2019
Tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

Sisältö

1 Johdanto ... 2

2 Tutkimustyö ja testausmetodit ... 2

3 Neuroverkon opettaminen ... 4

3.1 ReLu ... 6

3.2 Sigmoid ... 7

3.3 ReLu vs Sigmoid ... 8

4 Neuroverkon optimointi ... 9

4.1 Mallin optimointi ... 9

4.2 CPU vs. GPU ... 11

5 Ylioppiminen ... 12

6 Demo ... 12

Lähteet ... 13

Liitteet ... 14

Kuviot

Kuvio 1 Heatmap numerosta ... 3

Kuvio 2 Neuroverkolla tunnistettujen numeroiden tuloksia ... 3

Kuvio 3 Esimerkki neuroverkon output taulukosta. ... 4

Kuvio 4 Neuroverkon tarkkuuden kasvu opetuksessa ... 4

Kuvio 5 Neuroverkon tarkkuuden kehitys ReLu käytettäessä ... 6

Kuvio 6 Neuroverkon tarkkuuden kehitys Sigmoidia käytettäessä ... 7

Kuvio 7 Neuroverkon tarkkuuden kehitys ReLu ja Sigmoid käytettäessä ... 8

Kuvio 8 Hidden Layerien optimointi tuloksia ... 9

Kuvio 9 Arvioitu neuroverkon suorituskyky ... 10

Kuvio 10 CPU vs. GPU eri neuroni määrillä ... 11

Kuvio 11 Ylioppimisen demonstrointi ... 12

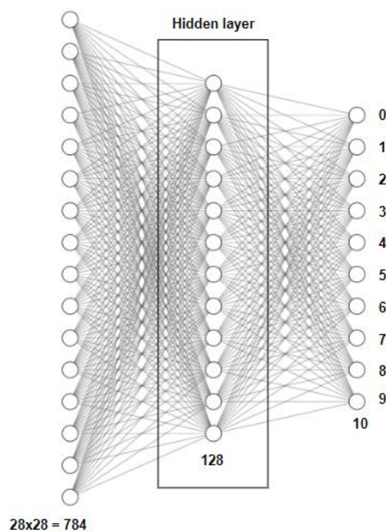
1 Johdanto

Tekoäly on yksi kasvava resurssi ja puheenaihe informaatioaikakautena. Sen mahdollisuudet automatisoida monia monimutkikkaita tehtäviä avaa monia uusia tutkimustieteen-, kehityksen- ja huolenaiheita.

Vaikka huolenaiheet eivät ole turhia, niihin tulee kiinnittää huomiota ja tekoäly onkin todettu luotettavaksi apulaiseksi monenlaisissa tehtävissä. Tieto – ja viestintätekniikan alalla se avaa kokonaan uudenlaisia tapoja ajatella ja toteuttaa ohjelmakoodia, sekä uuden tavan ajatella tietorakenteita ja algoritmeja sovellettuna.

2 Tutkimustyö ja testausmetodit

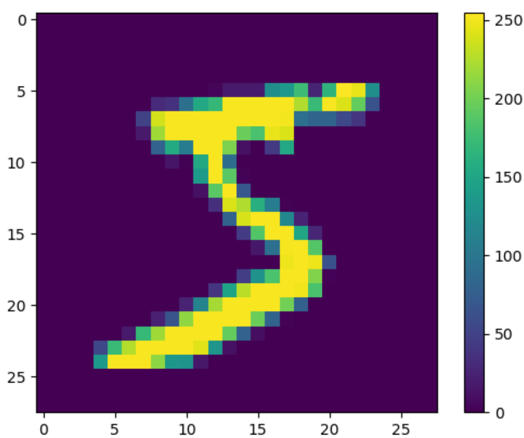
Tutkimuksen kohteena on käytetty yksinkertaista 0-9 numeroiden tunnistuksen neuroverkkomallia, jossa neuroverkon toiminnallisuutta on lähdetty tutkimaan sen eri tietorakenteiden ja algoritmien näkökulmasta.



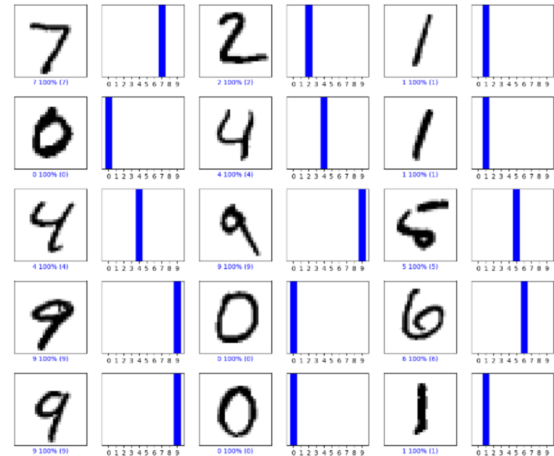
Tutkimusdata on saatu MNIST:in tietokannasta ja se sisältää 60 000 numeroa 28x28 resoluutiolla sekä 10 000 numeroa neuroverkon testaukseen. Neuroverkon koulutukseen käytimme Googlen Tensorflow ja sen Keras kirjastoa sekä python ohjelmointikieltä. Perusohjelmakoodi löytyy raportin liitteistä. Käytimme sitä pohjana kaikissa testeissä ja muokkasimme sitä testin tarkoituksen mukaan. Ajan mittaukset saamme Tensorflow:n palauttamasta datasta tai mittaamme manuaalisesti pythonin time kirjastoa hyödyntämällä.

Neuroverkon ensimmäisellä kerroksella on 784 neuronia, joihin syötetään tunnistettavan kuvan 784 pikseliä, nämä neuronit taas yhdistyvät ensimmäiselle hidden layerille, jossa on 128 neuronia ja lopuksi nämä neuronit yhdistyvät viimeiselle kerrokselle, jossa on 10 neuronia, jotka kuvastavat output arvoja nollasta yhdeksään.

Myöhemmin neuroverkkojen optimoimisosiossa tutkimme useamman hidden layerin ja niiden neuroneiden lukumäärää vaikutusta verkon toimintaan.



Kuvio 1 Heatmap numerosta



Kuvio 2 Neuroverkolla tunnistettujen numeroiden tuloksia

Tutkimuksessa on hyödynnetty algoritmin manipulaatiota muuttamalla sen hidden layereitä, ReLu funktiota Sigmoid funktioksi, solmujen määrän vaihtaminen hidden layereissä, ajettavien testikierrosten määrää, sekä laskennan suorittamista CPU:n sijasta GPU:lla.

Hidden layer on neuroverkkomalleissa osio, jonka neuronien tehtävänä on laskea painoarvojen summa, sen saamista neuroneista ja sijoittaa tulos aktivointifunktioon, eli esimerkiksi ReLu tai Sigmoid.

ReLu ja Sigmoid ovat tilastomatemaattisia laskentakaavoja, jota käytetään koneoppimisessa. Neuronien määrän muutos vaikuttaa neuroverkoissa sen yksityiskohtien laskemiseen, eli kuinka monta kiintopistettä tunnistettavaan dataan saadaan.

Ajettavien testikierrosten määrällä saadaan laskettua tarkemmat painoarvot, jotka vaikuttavat kokonaistuloksen tarkkuuteen ja niitä on hyvä ajaa käyttötarkoitukselle sopiva määrä, jotta saatu tulos on tarpeeksi luotettavaa. Liian suurella opetusmäärällä vaarana on yliopettaminen, eli neuroverkko tunnistaa testattavat alkiot hyvin, mutta ei enää kykene tunnistamaan uutta tietoa samalla tarkkuudella, ja tarkkuus uudessa tiedossa on siksi kehnoa.

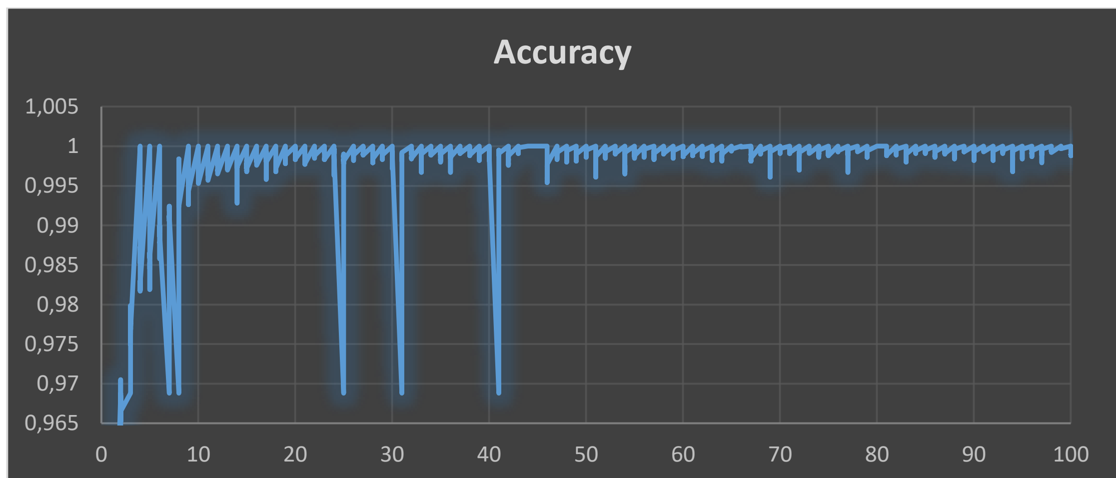
3 Neuroverkon opettaminen

Neuroverkon opettaminen tapahtuu vastavirta (eng. backpropagation) algoritmilla, jossa lasketaan neuronien painoarvoja ja minimoidaan edelliseltä kierrokselta saadun virheen määrää. Asianmukaisella painoarvojen hienosäädöllä saadaan pienempi virheen määrä, joka taas kasvattaa mallin luotettavuutta ja korkeampaa todennäköisyyttä oikealle ulostulo arvolle.

Tätä kutsutaan ohjeistetuksi koneoppimiseksi. Tuloksena neuroverkosta ulos saadaan todennäköisyyksiä asetetuille tulos vaihtoehdoille, tutkimuksen neuroverkon tapauksessa esimerkiksi, jos syötetty arvo on kaksi, saadaan ulostuloksi taulukko, jossa kerrotaan kuinka monella prosentilla syötetty arvo vastaa 0, 1, 2, 3…9.

```
[1.0917730e-14% 4.8915454e-16% 5.3028426e-15% 3.1428860e-07% 8.0108260e-17%
 2.3940569e-15% 2.3558858e-25% 9.9999964% 3.2570477e-13% 2.5445628e-09%]
```

Kuvio 3 Esimerkki neuroverkon output taulukosta.



Kuvio 4 Neuroverkon tarkkuuden kasvu opetuksessa

Testasimme testattavan neuroverkon tarkkuuden kehitystä nollasta sataan iteraatio välillä, kuten kuviosta 4 selviää. Se kuvaa verkon lasketun tarkkuuden kehitystä testatun tarkkuuden sijaan. Käytimme testissä ReLu funktiota ja kuten kuvassa näkyy arvon lähestyessä 100% tippuu tarkkuus hetkellisesti ja alkaa nousta taas takaisin 100% kohti.

Toisaalta poikkeavuuksia voidaan havaita 25, 32 ja 42 kierroksien kohdalla. Testissä käytetyn ReLu funktion takia muutokset ovat jyrkempiä alussa, ja siitä johtuvat tarkkuuden pudotukset näkyvät merkitsevästi. Tarkkuuden vaihtelu jatkuu, mutta tarkkuuden vähenemisen suuruus vähenee ajan kuluessa. Testien perusteella mallin tarkkuus tällä datasetillä on mahdollista saada 98%.

Tilastomatemaattisesti laskemisessa ei koskaan saada absoluuttisia totuuksia, eli 100% tarkkuutta vaan se on aina relatiivista suhteutettuna käyttötarpeeseen. Yleisesti ottaen kuitenkin luotettavuusluku p-arvoa, eli ns. hypoteesin testauksen todennäköisyyden arvoa: p(x), kun epäyhtälöt $0 \leq p(x) \leq 1$ kaikilla $x \in R$ pidetään tärkeänä, että luotettava tieto on p=0,5* eli vähintään 99,5% tarkka, ja vielä parempi on jos p=0,05** tai p=0,005***.

Sen lisäksi datan keskihajonta $s=\sigma_x=\sqrt{\sigma_x^2}$ vähenee mitä enemmän tilastomatemaattisessa datassa on alkioita N ja otantaan otetaan enemmän hyödyllistä dataa otanta n. Hyödyllisen datan laskemiseksi kannatta hyödyntää Pearsonin ja Spearmanin korrelaatiokertoimia. Pearsonin korrelaatiokerroin tunnetaan myös nimellä otoskorrelaatio.

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y},$$

Missä otoshajonnat s_x ja s_y lasketaan otoshajonnan kaavalla:

$$s_x = \sqrt[2]{\sum_{i=1}^{n} \frac{(x_{i-\bar{x}})}{n-1}}$$

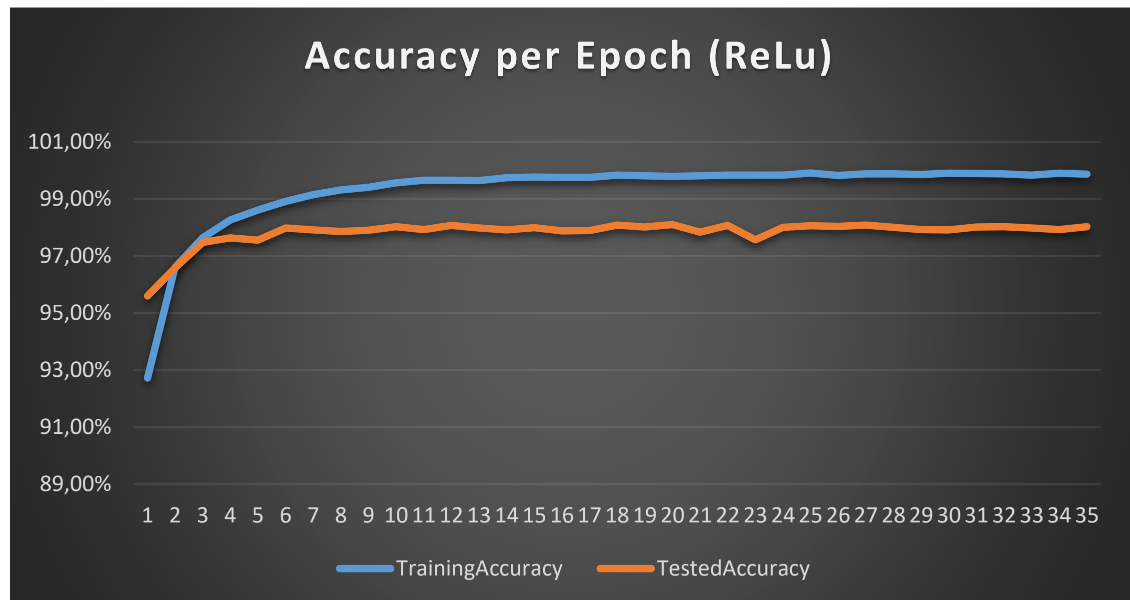
Pearsonin korrelaatiokertoimen avulla voidaan ottaa joukko otantoja ja laskea niiden relatiivinen suhde summien keskiarvosta. Spearmanin korrelaatiokerroin eli toisella nimellä tunnettu järjestyskorrelaatiokerroin lasketaan kaavalla:

$$\rho = 1 - \frac{6\sum d_i^2}{(n^2-1)n}$$

Sen avulla voidaan laskea otannan alkion tilastollinen merkitsevyystaso datassa. Se mahdollistaa tilastollisesti merkitsevän datan rajoittamista ja poikkeuksien näkyvyyden merkitystä tilastollisessa lopputuloksessa.

3.1 ReLu

ReLu eli Rectified Linear Unit on yleinen neuroverkoissa käytetty funktio. Se luo yksinkertaisen ja lineaarisen opettamisprosessin, joka mahdollistaa nopeamman neuroverkon opettamisen sigmoid:iin verrattuna, koska ReLu on helpompi ja nopeammin laskettava kuin sigmoid. Ja kuten kuviosta 5 näkyy tarkkuus kasvaa aluksi hyvin nopeasti, mutta hidastuu tietenkin loppua kohden, mutta myös tarkkuuden heittelyt ovat paljon suurempia kuin sigmoidilla, johtuen siitä, että ReLu ei ole yhtä tarkka.



Kuvio 5 Neuroverkon tarkkuuden kehitys ReLu käytettäessä

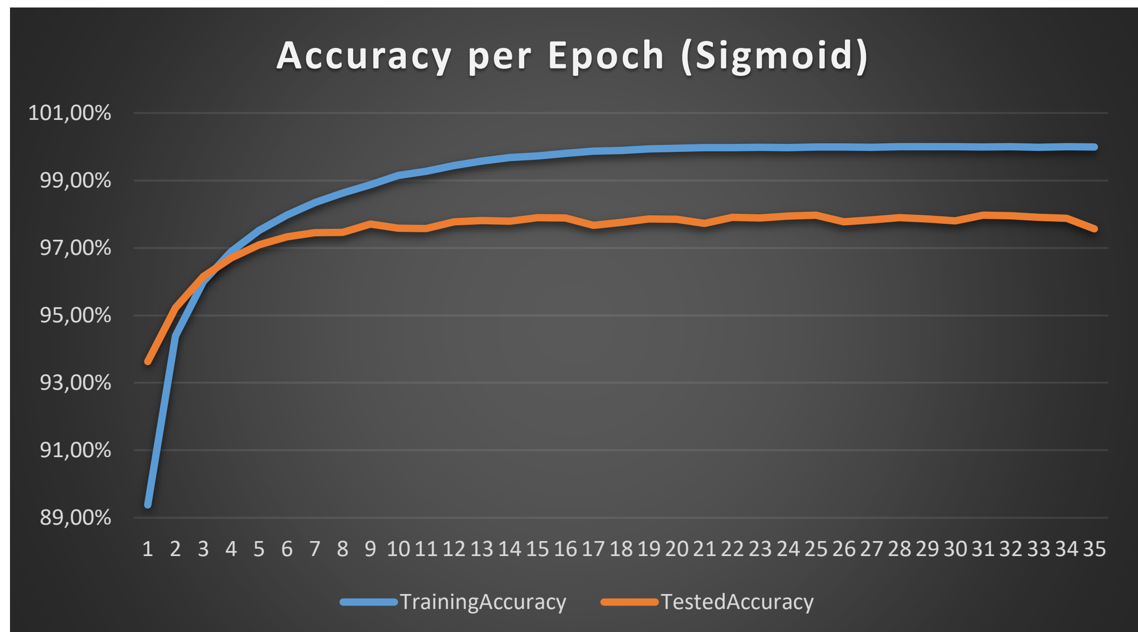
ReLu kaavana on: $f(x) = x^+ = \max(o, \bar{x})$. Analyyttisena funktiona käytetään Softplus kaavaa, josta smooth eli sulava versio on yleisin algoritmi $f(x) = \log(1 + e^x)$ ja sen derivaatan kaava on: $f'(x) = \frac{1}{1+e^{-x}}$, eli logistinen funktio.

Testitapauksesta voidaan havaita, että ReLu:n opetustarkkuus kasvaa jyrkästi ensimmäisten 15 opetuskierroksen, eli ns. epochin aikana, jonka jälkeen se tasautuu. Mer-

kittävä tehokkuuden muutos testatussa mallissa 2 kierroksella. Testattu tarkkuus verrannollisesti aloittaa korkeammasta arvosta ja tasautuu 8 kierroksen kohdalla ja pysyy relatiivisesti 97%-98% marginaalissa, eli 2-3% virhemarginaalissa.

3.2 Sigmoid

Sigmoid funktio on matemaattinen funktio, jota käytetään neuroverkkojen opetuksessa. Se on tarpeeksi nopea ja tarkka useimpiin vaadittaviin käyttötarkoituksiin, mutta myös hankala ja kallis laskea. Sigmoidin funktion kaava on: $S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$, ja toisin kuin ReLu, Sigmoidin funktio toteuttaa Sigmoidin käyrän kuvaajan.

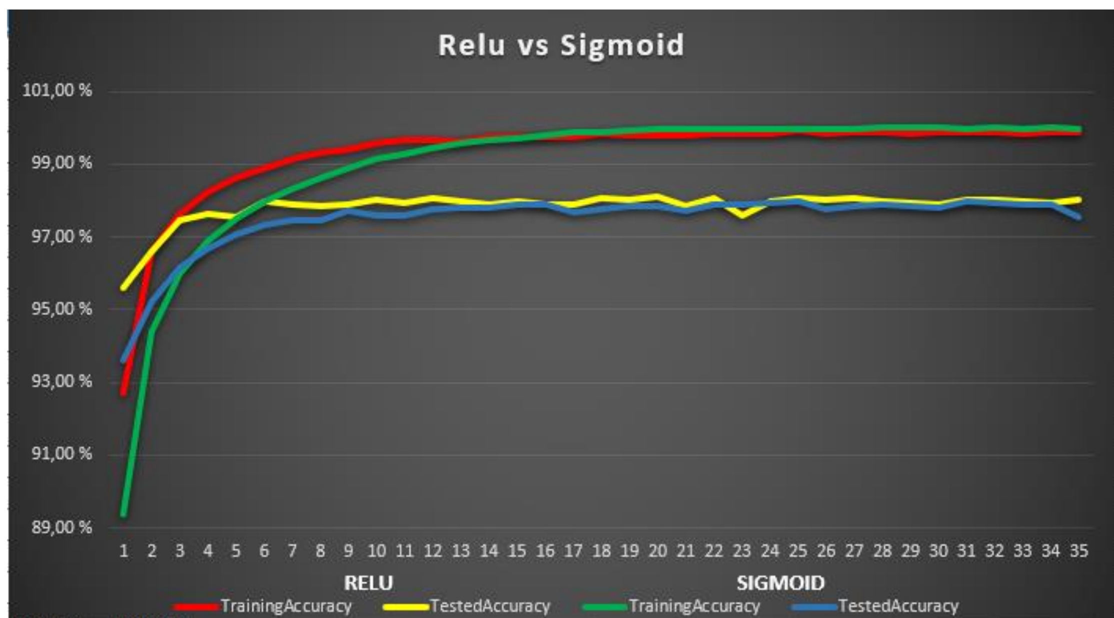


Kuvio 6 Neuroverkon tarkkuuden kehitys Sigmoidia käytettäessä

Kuten kuviosta 7 näkyy sigmoid on tasaisempi, tarkempi ja hitaampi kuin ReLu. Tarkkuus on aluksi huonompi, mutta se kasvaa hyvin lineaarisesti ja lähestyy 100% hyvin lähelle, ja koska sigmoidilla laskettu arvo on paljon tarkempi ei myöskään tarkkuuden pudotukset ole yhtä suuria kuin ReLu:a käytettäessä, vaan ne pysyvät todella pieninä. Sigmoid soveltuu paremmin silloin kun vaatimuksessa on tarkkuuden ja ajankäytön ideaalinen suhde. Toisin sanoen Sigmoid tasapainottaa tarkkuuden ja nopeuden relatiivisen suhteen keskiarvoksi.

3.3 ReLu vs Sigmoid

ReLu on nopeampi, helpompi ja halvempi laskea verrattuna sigmoid:iin, mutta ReLu:n tarkkuus on myös huonompi. Jos katsotaan kuviota 7 nähdään, että ReLu on alussa huomattavasti nopeampi ja tarkempi kuin sigmoid. Sigmoid taas aloittaa huonommalla tarkkuudella ja alkaa hyvin lineaarisesti nousta. Sigmoid saavuttaa ReLu:n tarkkuuden vasta noin 15 epochin kohdalla, mutta tämän jälkeen nähdäänkin kuinka sigmoid jatkaa tasaisesti nousuaan ja sen pudotukset tarkkuudessa ovat paljon pienempiä kuin ReLu:n. ReLu:n tarkkuuden heittelyt näkyvät erityisen hyvin testatussa tarkkuudessa.



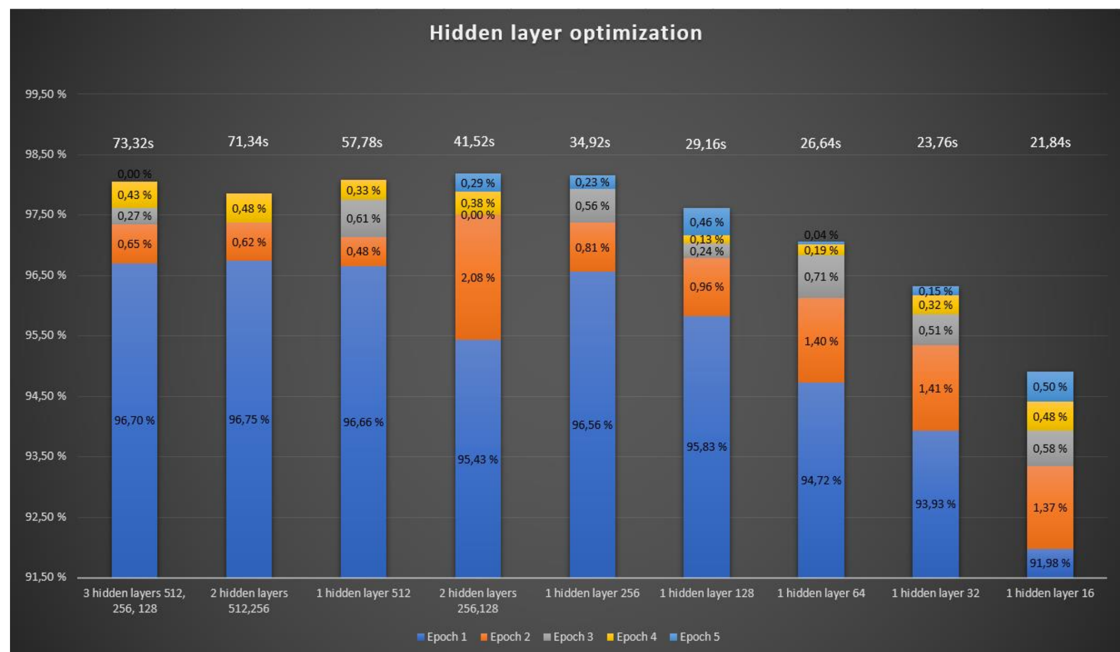
Kuvio 7 Neuroverkon tarkkuuden kehitys ReLu ja Sigmoid käytettäessä

Sigmoid on parempi silloin kun halutaan optimoida tarkkuuden ja ajankäytön resursseja, eli silloin kun tarkkuudella on myös väliä, mutta se ei kuitenkaan saa vallata liikaa ajankäytön resursseja myöskään. Sigmoid on luotettavampi ja tarkempi kuin ReLu, mutta sitä hitaampi ja kalliimpi toteuttaa. ReLu:n tarkkuus ja nopeus ovat kuitenkin niin hyviä, että sitä kannattaa käyttää pääasiallisesti.

4 Neuroverkon optimointi

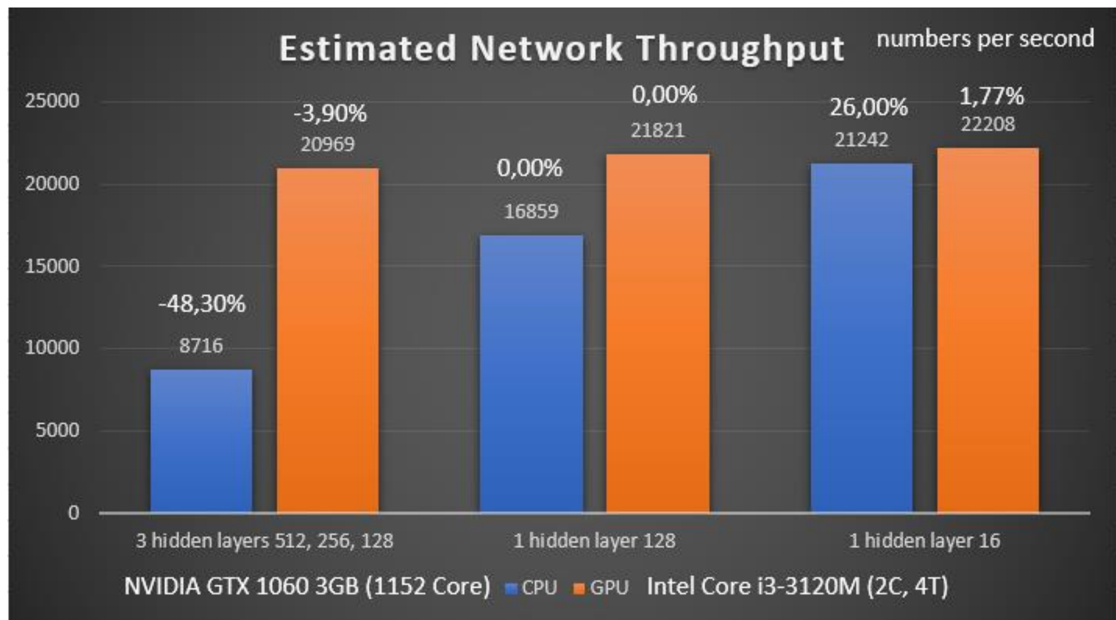
4.1 Mallin optimointi

Teimme erilaisia testauksia neuroverkon optimoimiseksi. Testaukseen käytimme samaa numerontunnistus neuroverkkoa ja testasimme sitä muuttamalla hidden layereitä ja niiden neuronien lukumäärä. Mittasimme verkon tarkkuutta, koulutusaikaa ja tehokkuutta käsitellä numeroita.



Kuvio 8 Hidden Layerien optimointi tuloksia

Kuten kuviosta 8 nähdään kun verkon monimutkaisuutta lisätään lisäämällä kerroksia ja neuroneita, tarkkuus paranee, mutta myös opetusaika nousee. Kuvion perusteella optimaalisin verkko ratkaisu nopeuden ja tarkkuuden suhteen olisi 1 hidden layer 256 neuronilla. Tällöin verkosta saataisiin tarkka, mutta se olisi myös suhteellisen nopea opettaa. Hidden Layerien määrän ja neuronien määrän valintaan vaikuttaa myös toinen asia. Halutaanko laskentatehoa ja aikaa käyttää ennemmin opetuksen vai verkon käytön puolella? Sillä mitä monimutkaisempi verkko on, sen hitaampi sen käyttäminen on, mutta tällöin se on myös tarkempi ja helpompi opettaa, kun taas yksinkertaisemmalla verkolla on vähemmän tarkkuutta, kouluttaminen kestää kauemmin, mutta sen käyttäminen on nopeampaa.



Kuvio 9 Arvioitu neuroverkon suorituskyky

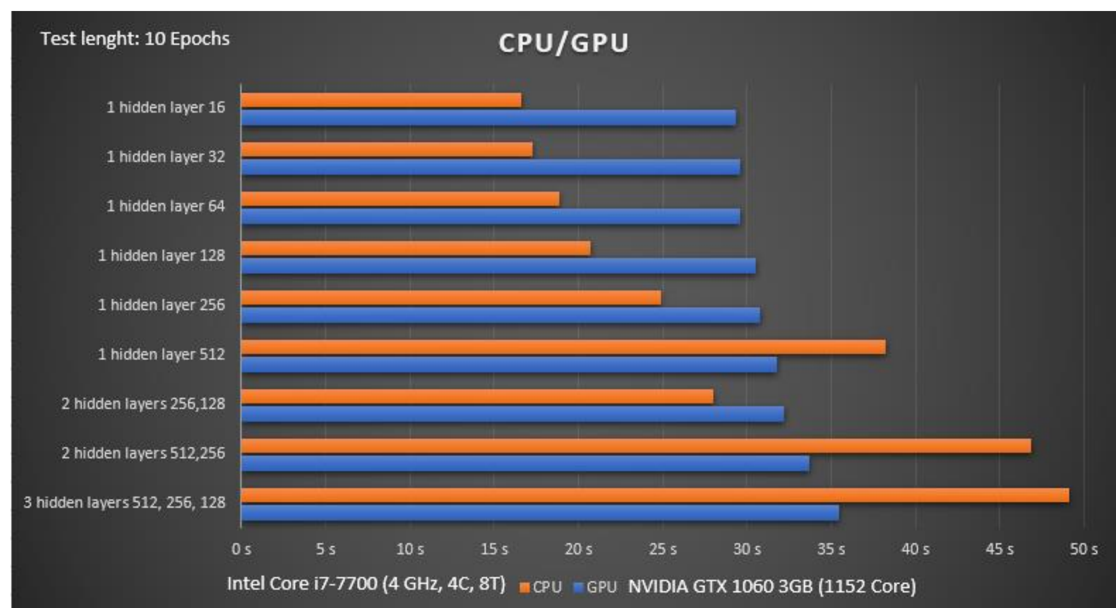
Mittasimme verkon suorituskykyä kolmella eri mallilla, monimutkikkain 3 hidden layerillä ja suurimmalla määrällä neuroneja sekä pienimmällä määrällä hidden layereitä ja neuroneita ja alkuperäisellä mallilla, jota käytimme muissa testeissä. Lisäksi mittasimme tulokset CPU:lla ja GPU:lla. Tulokset mitattiin mittaamalla kulunut aika 10 000 numeron testisarjan tunnistuksessa.

Kuten kuviosta 9 näkyy, monimutkikkaampi malli oli melkein puolet huonompi CPU:lla ja GPU:n tehojen ansiosta vain noin 4% huonompi suorituskyvyltään kuin perusarvona käytetty 1 hidden layerin ja 128 neuronin testimalli. Yksinkertaisempi malli oli taas noin 26% tehokkaampi CPU:lla ja noin 2% tehokkaampi GPU:lla kuin perusarvona käytetty malli.

Tuloksista voidaankin nähdä, että hidden layerien määrään ja neuronien määrään ei ole suoraa vastausta, vaan ne täytyy valita käyttökohteen mukaan riippuen siitä kuinka paljon tarkkuutta, nopeutta, opetusaikaa ja suorituskykyä halutaan neuroverkolta.

4.2 CPU vs. GPU

CPU:n ja GPU:n erona on se, että GPU voi hyödyntää sen tuhansia ytimiä laskemaan tuhansien neuronien summat kerralla, kun taas CPU voi laskea vain muutamia kymmeniä kerralla ja siksi GPU:n käyttö koneoppimisessa on laskelmallisesti merkitsevästi nopeampaa. Isot tietomäärät kuitenkin vaativat paljon VRAM:ia näytönohjaimelta ja isoimpien mallien kouluttamiseen tarvitaankin hyvin kalliita näytönohjaimia, joissa on todella paljon muistia. Esimerkiksi testauksessa käytetyn mallin opetukseen GPU:lla vaati melkein 3GB muistia.

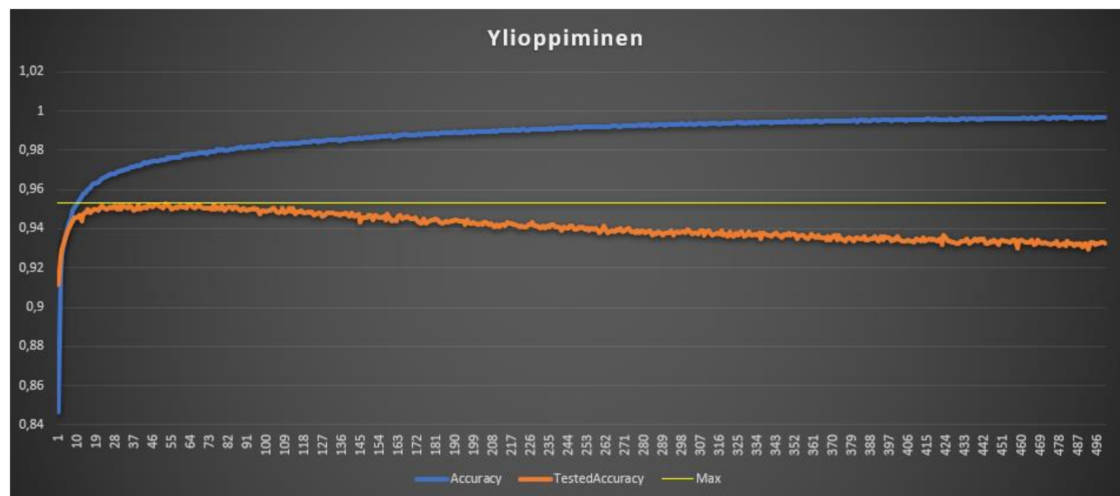


Kuvio 10 CPU vs. GPU eri neuroni määrillä

Testasimme CPU ja GPU 10 epochin pituisessa opetuksessa eri kokoisilla malleilla. Ja kuten kuviosta 10 näkyy, monimutkikkaimmissa verkoissa GPU on noin 30% nopeampi kuin CPU, mutta kun aletaan siirtyä yksinkertaisempiin verkkoihin, joissa on vähemmän neuroneita GPU alkaa hävitä CPU:lle jopa 76 %. Tämä johtuu siitä, että CPU:n ytimet ovat paljon nopeampia kuin GPU:n ytimet ja koska laskettavia neuroneita on vähemmän ei GPU pysty hyödyntämään sen tuhansia ytimiä vaan vain muutamia, jolloin CPU pystyy vähemmillä, mutta tehokkaammilla ytimillä tekemään laskut nopeammin. GPU:n todellinen etu tulee vasta syvemmissä ja monimutkaisemmissa verkoissa.

5 Ylioppiminen

Ylioppiminen tarkoitta sitä, kun ylioppinut neuroverkko osaa hyvin opetusaineiston, mutta ei osaa käsitellä muuta aineistoa oikein. Ylioppiminen heikentää siis neuroverkon toimivuutta. Kuviossa 11 voimme nähdä kuinka mallin tarkkuus jatkaa tasaisesti nousuaan koko ajan, mutta noin 90 epochin jälkeen testattu tarkkuus alkaa tippua tasaisesti alaspäin. Tästä näemme, että verkko on alkanut ylioppimaan ja testisarjan numeroiden tunnistus alkaa heikentyä.



Kuvio 11 Ylioppimisen demonstrointi

6 Demo

Linkki Demoon: https://jonneokkonen.com/HandWrittenNumbersDetection/

Loimme demon, jossa käyttäjä pääsee itse piirtämään numeron väliltä 0-9 ja tämän jälkeen neuroverkko suorittaa numeron tunnistuksen. Verkko on koulutettu python ympäristössä 30 epochin verran ja muuten verkon malli on samanlainen kuin 2 osiossa esitelty malli. Koulutuksen jälkeen veimme mallin json-muotoon ja latasimme sen tensorflown JavaScript kirjaston avulla. JavaScriptin puolella teimme tarvittavat muutokset canvas-elementistä saadulle kuvalle ja annoimme sen neuroverkolle tunnistettavaksi. Tunnistuksen jälkeen neuroverkkopalauttaa taulukon, josta näkee mikä numero syötetty arvo oli todennäköisemmin. Lopuksi vielä tallennamme syötetyn arvon ja oikean vastauksen tietokantaan, jotta arvoja voisi käyttää myöhemmin neuroverkon parantelussa.

Lähteet

How Does Back-Propagation in Artificial Neural Networks Work?. 2019. Artikkeli. Viitattu 5.12.2019. https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7.

Tensorflow image classification example. 2019. Tensorflown kuvan tunnistus esimerkki. Viitattu 5.12.2019. https://www.tensorflow.org/tutorials/keras/classification.

Liitteet

```
1  from __future__ import absolute_import, division, print_function, unicode_literals
2  
3  # TensorFlow and tf.keras librarys
4  import tensorflow as tf
5  from tensorflow import keras
6  from keras.datasets import mnist
7  
8  print("Tensorflow Version: " + str(tf.__version__))
9  
10 # Load MNIST Data
11 (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
12 
13 class_names = ['0','1','2','3','4','5','6','7','8','9']
14 
15 print("Train_images: " + str(train_images.shape))
16 print("Train_labels: " + str(len(train_labels)))
17 
18 print("Test_images: " + str(test_images.shape))
19 print("Test_labels: " + str(len(train_labels)))
20 
21 # Preprocessing
22 # Scale 0 to 255 RGB values to range 0 to 1 for training and testing images
23 train_images = train_images / 255.0
24 test_images = test_images / 255.0
25 
26 # Create Network Model
27 model = keras.Sequential([
28     keras.layers.Flatten(input_shape=(28, 28)),
29     keras.layers.Dense(128, activation='relu'),
30     keras.layers.Dense(10, activation='softmax')
31 ])
32 
33 # Compile Model
34 model.compile(optimizer='adam',
35               loss='sparse_categorical_crossentropy',
36               metrics=['accuracy'])
37 
38 # Train Model
39 model.fit(train_images, train_labels, epochs=10)
40 
41 # Save Model
42 model.save('model')
43 
44 # Test 10 000 number
45 test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
46 print('\nTest accuracy:', test_acc)
47 
```

Demon lähdekoodi: https://gitlab.labranet.jamk.fi/M2235/handwrittennumberdetection