Mirka Kinisjärvi

PILVIPOHJAISEN LASKENTASOVELLUKSEN TOTEUTUS LINUX-YMPÄRISTÖSSÄ

PILVIPOHJAISEN LASKENTASOVELLUKSEN TOTEUTUS LINUX-YMPÄRISTÖSSÄ

Mirka Kinisjärvi
Opinnäytetyö
Kevät 2017
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, langattomat laitteet

Tekijä: Mirka Kinisjärvi
Opinnäytetyön nimi: Pilvipohjaisen laskentasovelluksen toteutus Linux-ympäristössä
Työn ohjaaja: Kari Jyrkkä, Jari Hyttinen, Jukka Lämsä
Työn valmistumislukukausi ja -vuosi: Kevät 2017
Sivumäärä: 45 + 7 liitettä

Opinnäytetyön aiheena oli pilvipohjaisen laskentasovelluksen toteutus Linux-ympäristössä. Tavoitteena oli kehittää pilvityyppinen ohjelmisto- ja laiteympäristö algoritmilaskentaan ja luoda laskentasovellus, joka opettaa neuroverkon laskemaan parametreja syötetyn datan perusteella. Työn toimeksiantajana toimi Nokia Oyj.

Toteutus alkoi teoriatiedon kokoamisella pilvipalveluista, klusterin toteutusvaihtoehdoista ja neuroverkoista. Teoriatiedon pohjalta päätettiin sopiva toteutusvaihtoehto klusterin rakentamiseen Linux-käyttöjärjestelmälle. Testausympäristö rakennettiin yrityksen tuotannossa sijaiseviin laboratoriotiloihin. Viimeisenä vaiheena luotiin laskentasovellus TensorFlow-ympäristöä hyödyntäen.

Opinnäytetyön lopputuloksena saatiin ohjelmisto- ja laiteympäristö pystytettyä ja sen toimivuus testattua. Yritys pystyy hyödyntämään klusteria useisiin käyttökohteisiin sen laskentatehon ja skaalautuvuuden ansiosta. Laskentasovellus saatiin luotua ja neuroverkko oppi hyvin muodostamaan muuttujien epälineaariset riippuvuussuhteet havaintoaineistosta. Ohjelmassa käytettiin tuotannossa syntyvää prosessi- ja testidataa ja neuroverkon avulla muodostettiin yhteyksiä näiden välille.

Asiasanat: Hadoop, klusteri, Linux, neuroverkot, pilvipalvelut

ABSTRACT

Oulu University of Applied Sciences
Information technology, Wireless devices

Author: Mirka Kinisjärvi
Title of thesis: Development of a Cloud-based Computation Application in a Linux Environment
Supervisor: Kari Jyrkkä, Jari Hyttinen, Jukka Lämsä
Term and year when the thesis was submitted: Spring 2017
Pages: 45 + 7 appendices

The subject of this thesis was a cluster computation and neural network development. The goal was to develop a cloud type software and hardware environment to algorithm computation. In addition, the objectives included creating application that teaches the neural network to calculate parameters based on input data. The work was commissioned by Nokia Oyj.

The implementation of this thesis began with compiling theoretical knowledge of cloud computing, computer cluster and neural networks. On the basis of theoretical knowledge was chosen a suitable implementation option for building a cluster in a Linux operating system. The testing environment was built in the laboratory located in the company's production. Finally a computing application was created utilizing the TensorFlow environment.

As a result of this thesis, the software and hardware environment was successfully completed and its functionality tested. The company can utilize the cluster for a variety uses due to its computing power and scalability. Computational application was created and neural network learned well to form nonlinear dependencies of the variables from the observational data. The program utilized process and test data from production and the neural network created connections between them.

Keywords: Hadoop, computer cluster, Linux, neural networks, cloud computing

ALKULAUSE

Haluan kiittää Nokia Oyj:tä tämän opinnäytetyön mahdollistamisesta ja mielenkiintoisesta opinnäytetyöaiheesta. Erityisesti haluan kiittää Nokia Oyj:n yhdyshenkilöinä ja valvojina toimineita Jari Hyttistä ja Jukka Lämsää. Kiitokset myös kaikille työkavereille, jotka edesauttoivat tämän työn tekemistä.

Työn valvojana OAMK:n puolesta toimi Kari Jyrkkä. Kiitokset hänelle opinnäytetyön ohjauksesta ja työn edetessä saamastani rakentavasta palautteesta.

Iso kiitos myös poikaystävälle, perheelle ja ystäville kaikesta tuesta ja kannustuksesta kuluneen kevään aikana.

Oulussa 30.4.2017

Mirka Kinisjärvi

SISÄLLYS

SANASTO

API	*(Application Programming Interface)* Ohjelmointirajapinta
BIOS	*(Basic Input-Output System)* Laitteiston alustuksen suorittamiseen käytettävä tietokoneohjelma käynnistysprosessin aikana
Ethernet	Yleisin ja laajasti käytetty pakettipohjainen lähiverkkotekniikka
Hadoop	Datan tallennus- ja analysointialusta
HDFS	*(Hadoop Distributed File System)* Hadoopin oma tiedostojärjestelmä
IaaS	*(Infrastructure as a Service)* Infrastruktuuri palveluna
ISO-levykuva	Tiedosto, johon on tallennettu massamuistin sisältö ja rakenne
Jupyter Notebook	Avoimen lähdekoodin web-sovellus
KVM-kytkin	*(Keyboard, Video, Mouse)* Laite, joka mahdollistaa usean tietokoneen käyttämisen yhden näppäimistön, näytön ja hiiren avulla
LAN	*(Local Area Network)* Lähiverkko
MapReduce	Hajautettu analytiikka
MSE	*(Mean Squared Error)* Keskineliövirhe
Noodi	Klusterin jäsentietokone
PaaS	*(Platform as a Service)* Sovellusalusta palveluna
PC	*(Personal Computer)* Henkilökohtainen tietokone

Rocks Cluster	Linux-pohjainen suurteholaskennan klusteri
RSS	*(Residual Sum of Squares)* Jäännösneliösumma
SaaS	*(Software as a Service)* Sovellukset palveluna
Sigmoid-funktio	Matemaattinen esitys biologisen hermosolun käyttäytymisestä
SSH	*(Secure Shell)* Salattuun tietoliikenteeseen tarkoitettu protokolla
Tanh	*(Hyperbolic Tangent)* Hyperbolinen tangentti
TensorFlow	Avoimen lähdekoodin ohjelmistokirjasto numeeriseen laskentaan
USB	*(Universal Serial Bus)* Sarjaväyläarkkitehtuuri oheislaitteiden liittämiseksi tietokoneeseen

1 JOHDANTO

Datan määrä on lisääntynyt ja monipuolistunut viime vuosina vauhdilla. Nykyään suurien tietomäärien varastoimiseksi vaaditaan paljon tallennustilaa ja niiden prosessoimiseen runsaasti muistia sekä prosessointitehoa. Suurten datamäärien käsittely yksittäisellä tietokoneella ei ole suuren koon vuoksi mahdollista tehokkuudesta puhumattakaan. Ratkaisuna tähän voidaan käyttää tietokoneklustereita ja niissä suoritettavaa hajautettua laskentaa. Klustereissa dataa voidaan hallinnoida tehokkaasti, skaalautuvasti ja pienin kustannuksin.

Usein relaatiotietokannoissa puhutaan ”schema-on-write”-ajattelusta, jossa tiedetään mihin tauluihin ja sarakkeisiin tallennettava tieto viedään. Suurten datamäärien yhteydessä puhutaan ”schema-on-read”-mallista, jossa suuri määrä dataa halutaan tallentaa tallennusjärjestelmään, mutta ei olla kiinnostuneita siitä, millaisiin tietorakenteisiin data menee ja kuinka sitä käsitellään. Suurella nopeudella syntyvää dataa kerätään talteen skaalautuvaan ja varmistettuun tallennusjärjestelmään, josta voidaan myöhemmin hakea, tarkastella ja analysoida dataa eri menetelmin. (1.) Tilaajan tuotantoympäristössä syntyy suuria määriä tämänkaltaista dataa, jota halutaan analysoida ja hyödyntää.

Ajatus opinnäytetyön aiheesta syntyi, kun yrityksessä mietittiin keinoja toteuttaa suurta datamäärää hyödyntävä alusta, jonka käyttöä voitaisiin kuitenkin hallita luotettavasti. Lisäksi alustan avulla voitaisiin toteuttaa muitakin järjestelmiä, jotka eivät suoraan liity datan prosessointiin. Tässä tapauksessa datana toimii tuotannon prosessi- ja testidata. Prosessi- ja testidata on monipuolista ja datan tietotyypit ovat vaihtelevia. Lisäksi eri tietotyyppejä voi tulla jatkuvasti lisää ja alustasta halutaan skaalautuva, koska dataan ja datan prosessointiin liittyviä innovaatioita on paljon ja niitä tulee nopeasti lisää.

Opinnäytetyön tavoitteena on kehittää pilvityyppinen ohjelmisto- ja laiteympäristö algoritmilaskentaan. Lisäksi tavoitteisiin kuuluu laskentasovelluksen tekeminen, joka opettaa neuroverkon laskemaan parametreja syötetyn datan perusteella. Opetusalgoritmi ja neuroverkko luodaan TensorFlow-ympäristöä käyttäen.

Työ aloitetaan tutustumalla pilvipalveluihin, klusterilaskentajärjestelmiin ja neuroverkkoihin liittyvään teoriaan. Työssä tutkitaan mahdollisia laskentajärjestelmiä Linux-alustalle. Teoriatiedon pohjalta päätetään sopiva toteutusvaihtoehto laskentajärjestelmän toteuttamiseen tuotannossa sijaitsevaan testerilaboratorioon. Käytännön osuudessa toteutetaan testausympäristö valitulla järjestelmällä ja luodaan laskentasovellus. Laskentasovelluksena toteutetaan neuroverkkosovellus käyttäen TensorFlow-ympäristöä.

Tämä opinnäytetyö on osa Oulun ammattikorkeakoulun tietotekniikan koulutusohjelman opintoja. Opinnäytetyön tilaajana toimii Nokia Oyj ja työ tehdään kevään 2017 aikana. Nokia on globaali teknologiajohtaja ohjelmoitavassa maailmassa. Nokia luo mullistavaa tulevaisuuden teknologiaa ihmisten muuttuviin tarpeisiin esimerkiksi kehittämällä uudenlaisia virtuaalitodellisuuden ja digitaalisen terveyden sovelluksia. Yritys toimii yli sadassa maassa ja henkilöstö koostuu 160:sta eri kansallisuudesta. (2.)

2 PILVIPALVELUT

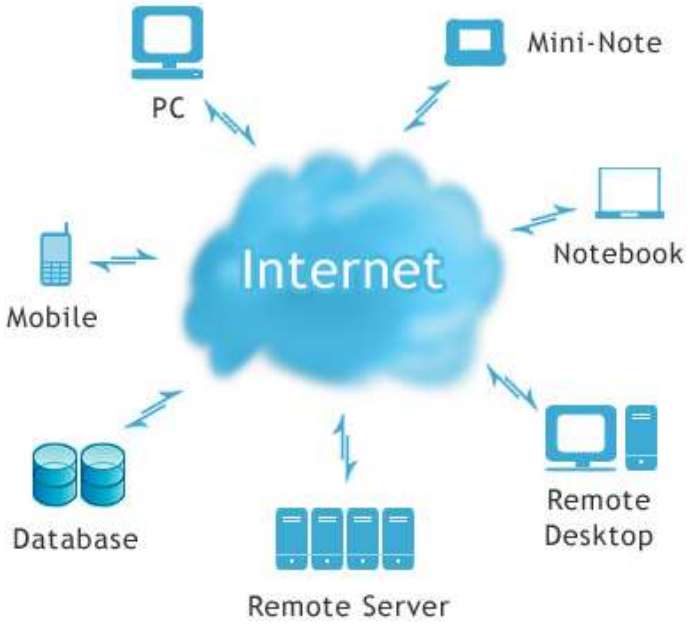
Opinnäytetyössä rakennetaan pilvipalvelualusta, joten aiheen teoriaan tutustuminen oli tarpeellista. Pilvipalveluissa on kyse tietotekniikan palvelullistumisesta, ja ajatuksena on, että tietotekniikka ei tarvitse omistaa, vaan sitä saa käyttöönsä palveluna. Sana cloud eli pilvi on internetin metafora. Pilvipalveluissa internet toimii verkkona, jonka yli palveluita kulutetaan. (3, s. 92.)

2.1 Pilvipalveluiden määritelmä

Yleiskielessä pilvipalveluilla tarkoitetaan internetistä hankittua tietokonekapasiteettia, sovelluksia tai muita palvelusuoritteita (4, s. 32). Käyttäjä voi käyttää palveluita riippumatta palvelun sijainnista tai päivitysajoista (5).

Pilvipalveluilla on useita käyttökohteita. Pilvipalvelut mahdollistavat online-palveluiden käyttämisen, kuten sähköpostin lähettämisen, tiedostojen muokkaamisen, elokuvien ja TV:n katsomisen, pelien pelaamisen tai kuvien ja tiedostojen tallentamisen. Lisäksi pilvipalveluiden ansiosta tavalliset internetin käyttäjät ja yritykset voivat luoda uusia sovelluksia ja palveluita, varastoida ja varmuuskopioida tietoja, suoratoistaa videoita sekä analysoida tietoa ja hyödyntää sitä. (6.) Esimerkkeinä tunnetuista pilvipalveluista ovat Facebook, Dropbox, LinkedIn ja Hotmail.

Pilvipalveluiden ominaispiirteitä ovat itsepalvelullisuus, pääsy palveluihin eri päätelaitteilla, resurssien yhteiskäyttö, nopea joustavuus ja käytön tarkka mittaaminen. Itsepalvelullisuus viittaa siihen, että tietotekniikkaresurssit saa tarvittaessa käyttöönsä ja käytön voi myös lopettaa ilman yhteyttä palveluntarjoajaan. Päätelaiteriippumattomuus tarkoittaa, että palveluiden käyttö onnistuu niin työasemalla, kannettavalla tietokoneella kuin mobiililaitteillakin. Tätä havainnollistaa hyvin kuvassa 1 näkyvä pilvipalvelu. Resurssien yhteiskäyttö tarkoittaa, että asiakas ei tarvitse eikä yleensä myöskään saa tietoa siitä, missä ja millä tavoin palvelu toteutetaan. Nopean joustavuuden ansiosta palvelut skaalautuvat nopeasti ja joustavasti, eikä kapasiteettirajoitteita ole usein lainkaan. Resurssien käyttöä valvotaan ja mitataan tarkasti, asiakas maksaa vain käytetystä kapasiteetista ja palveluntarjoaja laskuttaa sen mukaisesti. (3. s. 93–94.)



KUVA 1. Pilvipalvelu (7)

2.2 Pilvipalvelumallit

Pilvipalvelut on luokiteltu muutamaan päätyyppiin teknisen toteutustavan perusteella. Toteutustapa kertoo, minkälaisia tietojenkäsittelytehtäviä pilvipalveluista saadaan sekä miten kyseessä olevaan koneistoon liitytään. Nämä pääryhmät ovat infrastruktuuri palveluna (IaaS), sovellusalusta palveluna (PaaS) ja sovellukset palveluna (SaaS). (4, s. 50.) Kuvassa 2 näkyy eri pilvipalvelumallit ja sovelluksia, joita kuhunkin malliin kuuluu.

IaaS Infrastructure as a Service	PaaS Platform as a Service	SaaS Software as a Service
Virtuaalipalvelin	Kehitystyökalut	Sähköposti
Tallennusjärjestelmä	Sovelluspalvelin	Pelit
Varmistusjärjestelmä	Tietokantapalvelin	Virtuaalityöpöytä
Esim. Amazon Web Services	Esim. Google Web Toolkit	Esim. Gmail

KUVA 2. Pilvipalveluiden luokittelu (8)

2.2.1 Infrastructure as a Service

IaaS (Infrastructure as a Service) tarkoittaa infrastruktuuria palveluna. IaaS-tyyppisessä pilvipalvelussa palveluntarjoaja ylläpitää internetissä virtuaalista konesalia tai konesaleja, joista asiakas saa käyttöön oman lohkonsa eli tarvitsemansa infrastruktuurin. Asiakas perustaa näihin lohkoihin tarvitsemansa käyttöjärjestelmän ja asentaa sen päälle omat sovelluksensa. (4, s. 52.) Ostamisen ja omistamisen tai pitkäaikaisen sitoutumisen sijaan kapasiteettia voidaan ottaa joustavasti käyttöön tarpeen mukaan. IaaS eroaa perinteisestä ulkoistamisesta joustavuudessa, itsepalvelussa, resurssien yhteiskäytössä, automaatiossa ja käyttöön perustuvassa laskutuksessa. (9, s. 22–23.)

2.2.2 Platform as a Service

PaaS (Platform as a Service) tarkoittaa sovellusalustaa palveluna. PaaS-mallissa palveluntarjoaja tarjoaa alustan, jonka päälle sovelluksia voidaan kehittää ja jolla niitä voidaan testata sekä ylläpitää. Kehitystyöstä tulee yksinkertaisempaa, kun infrastruktuurista ei tarvitse huolehtia ja suuri määrä toiminnallisuuksia on saatavilla valmiina moduuleina ja ohjelmointirajapintoina. Lisäksi kolmansien osapuolten kehittämät maksulliset lisäosat tarjoavat laajennus- ja toiminnallisuusmahdollisuuksia. (9, s. 24.) Tässä opinnäytetyössä käytettiin PaaS-mallia.

2.2.3 Software as a Service

SaaS (Software as a Service) tarkoittaa sovelluksia palveluna. SaaS-tyyppisessä pilvipalvelussa asiakas hankkii itselleen pelkän sovelluksen, joka jaetaan tietoliikenneyhteyden avulla loppukäyttäjän selaimeen. Palveluntarjoaja huolehtii kaikesta muusta. (4, s. 53.) Omistamisen, asentamisen, ylläpidon ja päivittämisen sijaan asiakas ostaa sovellukset käyttöönsä tarvittaessa. Lisenssimaksun sijaan asiakas maksaa esimerkiksi aikaperusteisen käyttäjä- tai konekohtaisen maksun. Toimintamalli alentaa ohjelmistoihin ja niihin liittyvään laitteistoon sitoutuneen pääoman määrää, poistaa ylläpitoon ja päivityksiin liittyvän tuskan sekä vapauttaa henkilöstöresursseja yrityksen kannalta tuottavampiin tehtäviin. (9, s. 25–26.)

2.3 Pilvipalvelutyypit

Pilvipalveluiden tyypit voidaan jakaa neljään kategoriaan, jotka ovat yksityinen pilvi, yhteisöllinen pilvi, julkinen pilvi ja hybridipilvi (9, s. 27).

2.3.1 Yksityinen pilvi

Yksityinen pilvi on yrityksen tai julkisyhteisön oman LAN-lähiverkon tai muulla tavoin järjestetyn luotetun verkon kautta käytettävä pilvipalvelukoneisto, jossa ei tarvita erillistä tietoliikenneyhteyttä. Mallissa asiakas järjestää ja omistaa itse pilvipalvelukoneistonsa ylläpitoprosesseineen ja omistamisen kustannuksineen. (4, s. 55.)

Tässä opinnäytetyössä käytettiin yrityksen omaa LAN-sisäverkkoa, joten pilvipalvelutyyppinä toimi yksityinen pilvi. Tulevaisuudessa on mahdollista, että järjestelmä siirretään joko kokonaan tai osittain kolmannen osapuolen IT-infrastruktuuriin, jolloin pilvipalvelutyyppinä toimisi yhteisöllinen pilvi tai hybridipilvi. Tällä hetkellä ei haluttu vielä tehdä näin, koska haluttiin oppia aiheesta sekä innovoida sovelluksia.

2.3.2 Yhteisöllinen pilvi

Yhteisöllisessä pilvessä infrastruktuuri ja pilvikoneisto ovat useamman organisaation yhteisomistuksessa. Organisaatiot käyttävät pilvipalveluita yhdessä keskitetysti. Laitteisto ja pilvipalvelut voivat olla jonkin näiden organisaatioiden tiloissa tai ulkopuolisen tahon tiloissa ja hallinnoimana. Mallissa pilven pystyttämisen ja omistamisen kustannukset jakautuvat edullisemmin, koska käyttäjiä on yhden asemasta useampia. Yhteisöllinen pilvi -mallin avulla julkinen valta voisi välttyä rakentamasta teknistä ympäristöä erikseen jokaiseen virastoon tai laitokseen. (4, s. 56.)

2.3.3 Julkinen pilvi

Julkinen pilvi on internetyhteyden kautta käytettävä pilvipalvelukoneisto. Laitteisto on palveluntarjoajan tiloissa ja hallinnoimana ja asiakas käyttää palvelua internetyhteyden kautta. Asiakas saa kapasiteettia jaetusta ympäristöstä ilman omassa omistuksessa olevaa laitteistoa tai kapasiteettia. Julkisen pilven tarjoava

yritys vastaa pilvikoneistossa olevien laitteiden ylläpidosta ja omistamisen kustannuksista. (4, s. 54–55.)

2.3.4 Hybridipilvi

Hybridipilvi on yhdistelmä yksityisen ja julkisen pilven toteutuksista. Siinä yrityksen oma yksityinen pilvi yhdistetään pilvipalveluntarjoajan tekniseen ympäristöön tietoliikenneyhteyden kautta. (4, s. 56.)

2.4 Big data ja pilvipalvelut

Big data on käsitteenä epämääräinen, eikä sille ole yhtä täsmällistä selitystä. Se viittaa ilmiöön, jossa datan määrä maailmassa kasvaa eksponentiaalisesti. Toisaalta big data viittaa ratkaisuihin, joita on tarjolla kasvavan datamäärän tallentamiseen, liikutteluun ja ennen kaikkea hyödyntämiseen eli analysointiin. (10.) Puhuttaessa big datasta teknisessä mielessä puhutaan usein klusterimuotoon rakennetuista tallennus- ja laskenta-arkkitehtuureista (3, s. 52).

Suuri osa pilvipalveluntarjoajista on siirtynyt mukaan big datan hyödyntämiseen ja ainakin osin uudelleenkategorisoinut valikoimaansa. Mitä muutama vuosi sitten myytiin pilvipalveluna, on nyt big dataa pilvipalveluna. Käytännössä tällä tarkoitetaan tallennusratkaisua pilvipalveluna tai analytiikkaa palveluna. (3, s.162.)

Pilvipalveluina tarjottavien ratkaisujen edut ovat huomattavat. Ratkaisut eivät vaadi investointeja laitteisiin ja ohjelmistoihin tai pitkäkestoista sopimuksellista sitoutumista, eikä kapasiteettitarvetta tarvitse tietää etukäteen. Big data ja pilvipalvelut tarjoavat mahdollisuuden saavuttaa kilpailuetua, lisätä joustavuutta ja säästää kustannuksissa. Pilvipalvelu- ja big data -ajan menestyvä yritys on se, joka onnistuu keräämään eniten dataa, yhdistämään sen parhaiten muiden organisaatioiden dataan ja analysoimalla kokonaisuutta tuottamaan hyödyksi muutettavaa lisäarvoa. (3, s. 162.)

Tässä opinnäytetyössä käytetty tuotannon prosessi- ja testidata täytti big datan määritelmän. Pilvipalvelutyyppinä käytettiin yksityistä pilveä, jonka etuina on, että

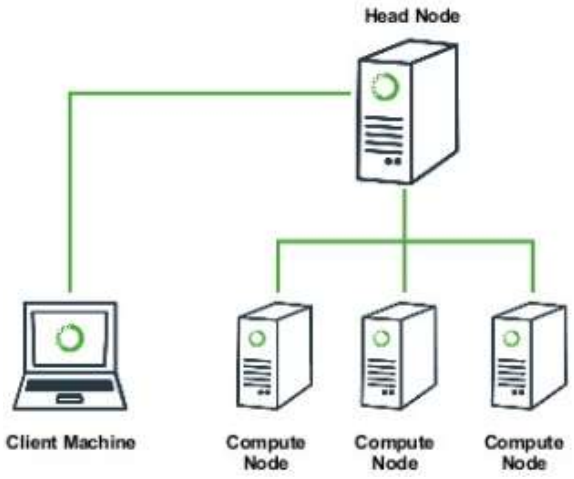
tuotannossa syntyvää big dataa voitiin analysoida ja hyödyntää joustavasti ja luotettavasti. Pilvipalvelua tullaan käyttämään yrityksessä erilaisiin tuotannon prosessi- ja testidatan analytiikkaan.

3 TIETOKONEKLUSTERI

Tietotekniikassa klusteri -termillä tarkoitetaan kahden tai useamman tietokoneen yhdistelmää, jotka ovat verkon kautta kytkettynä toisiinsa. Yksi tietokone toimii palvelimena, joka jakaa muiden tietokoneiden eli noodien kesken tehtäviä. (11.) Tietokoneet toimivat yhtenä paljon tehokkaampana koneena, mutta käyttäjälle näkyy vain yksi järjestelmä (12). Klusterin tehokkuus saavutetaan rinnakkaisprosessoinnilla noodien kesken (13). Kuvassa 3 näkyy klusterin tyypillinen arkkitehtuuri, jossa ”head node” toimii palvelimena ja ”compute nodet” ovat noodeja.

3.1 Klusterin hyödyt

Klusterin hyötyinä ovat edullisuus, toimintavarmuus ja skaalautuvuus. Edullisuus perustuu siihen, että yleensä palvelimen käyttöjärjestelmä ja asennettava alusta ovat avoimen lähdekoodin ohjelmistoja, joten niistä ei joudu maksamaan lisenssimaksuja. Lisäksi etuna on riippumattomuus tietyn laitetoimittajan ratkaisuista. Toimintavarmuus perustuu klusterin toimintalogiikkaan. Kaikki on hajautettua ja se tuo mukanaan vikasietoisuuden. Yhden laitteen vikaantuminen ei johda klusterin toimimattomuuteen tai datan häviämiseen. Skaalautuvuus on yksi keskeisimmistä eduista. Klusterin koneiden määrää voidaan kasvattaa tarpeen vaatiessa. Skaalautuvuus on keskeisessä roolissa myös big dataa ajatellen. Datan tallentamisen ja hyödyntämisen vaihtoehdot vähenevät, kun datamäärät kasvavat. (3, s. 73–74.)



KUVA 3. Klusterin arkkitehtuuri (14)

3.2 Klusterin toteutusvaihtoehtoja

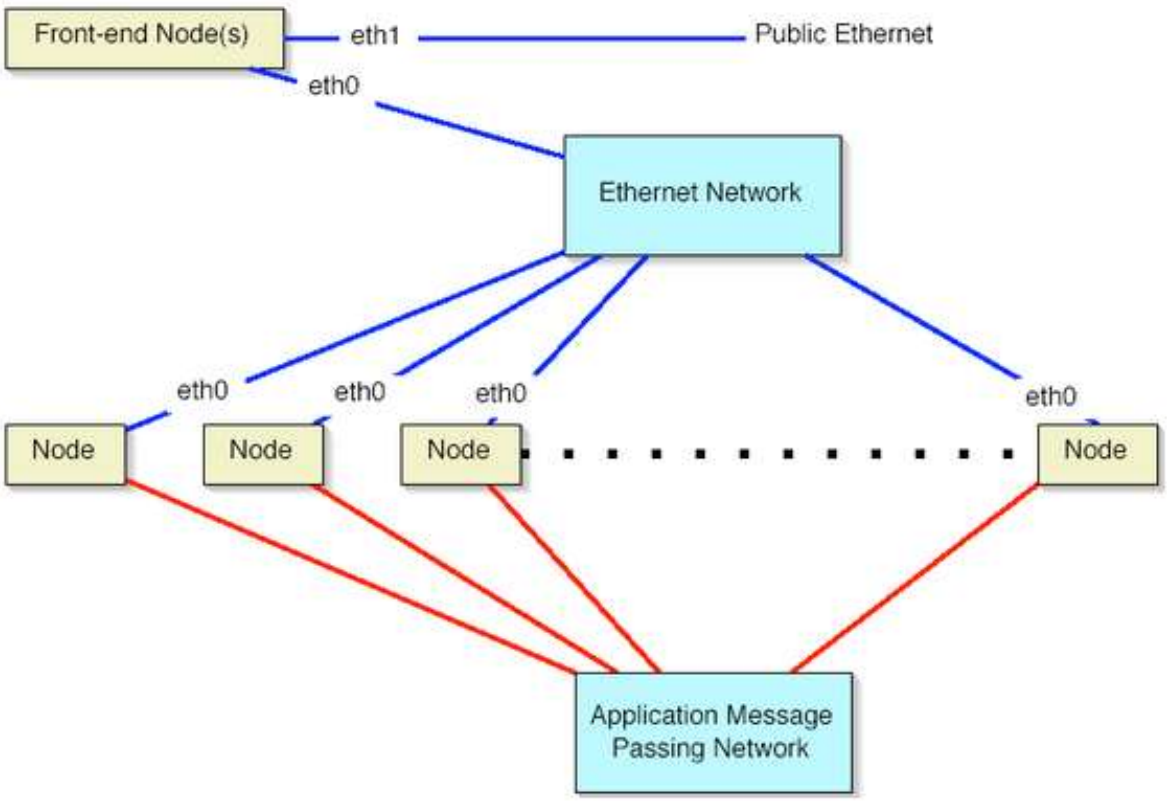
Tässä opinnäytetyössä päädyttiin käyttämään Linux-käyttöjärjestelmää sen muokattavuuden, vakauden, hyvän tietoturvallisuuden ja edullisuuden takia. Valintaa tuki myös aiemmat vastaavat suurteholaskennan toteutukset kyseisellä järjestelmällä.

Klustereiden toteuttamiseen on olemassa useita sekä ilmaisia että maksullisia alustoja. Toteutusta suunniteltaessa tutkittiin avoimen lähdekoodin mukaisia järjestelmiä klusterin toteuttamiseen. Klusterissa haluttiin suorittaa Matlab-skriptejä ja TensorFlow'ta, joten järjestelmistä tutkittiin soveltuvuus kyseisiin ohjelmiin. Eri toteutusvaihtoehtoja tutkittiin ja karsittiin siten, että jäljelle jäi kaksi toteutusvaihtoehtoa ja lopulta niistä valittiin parhaiten sopiva klusterin toteuttamiseen. Tässä kappaleessa on esitetty nämä kaksi toteutusvaihtoehtoa, joista valitaan myöhemmin toinen käytettäväksi alustaksi.

3.2.1 Rocks Cluster Distribution

Rocks Cluster Distribution on yksi Linux-jakeluun tarkoitettu suurteholaskennan klusteri, joka pohjautuu CentOS-Linux-käyttöjärjestelmään. Rocks on ilmainen ja perustuu avoimeen lähdekoodiin. Kehittäjien tärkein päämäärä oli tehdä järjestelmästä helposti käytettävä, hallittava, päivitettävä ja skaalautuva. (15.)

Kuvassa 4 näkyy Rocks-klusterin arkkitehtuuri. Kuten kuvasta näkyy, kaikki klusterin noodit on kytketty Ethernetillä kytkimen kautta sisäverkkoon. Tämä verkko pidetään yksityisenä, jotta kaikki liikenne tässä verkossa on fyysisesti erotettuna ulkoisesta julkisesta verkosta. Klusterin palvelin tarvitsee kaksi Ethernet-rajapintaa. Eth0 on kytkettynä samaan Ethernet-verkkoon kuin noodit ja eth1 on yhdistettynä ulkoiseen verkkoon, esimerkiksi internetiin tai yrityksen intranetiin. (16.)



KUVA 4. Rocks-klusterin arkkitehtuuri (16)

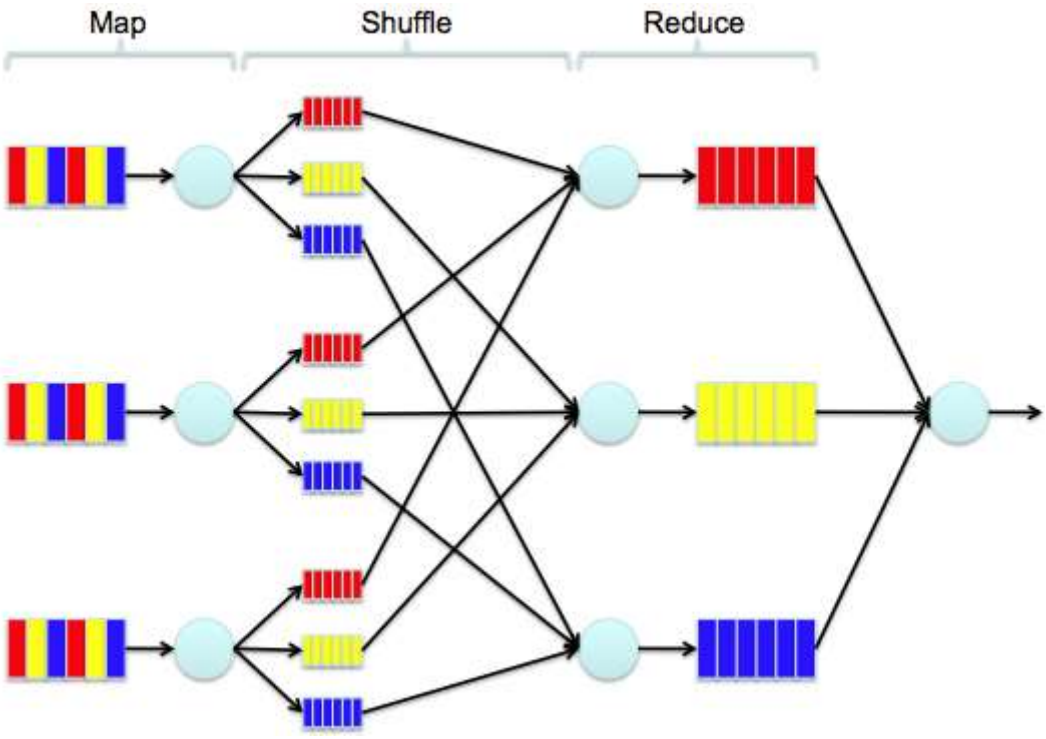
3.2.2 Apache Hadoop

Hadoop on datan tallennus- ja analysointialusta, jolla pystytään tallentamaan ja käsittelemään hajautettuja datamassoja. Tietoliikenneverkon klustereiden palvelimissa olevia datamassoja voidaan käsitellä rinnakkain ja samanaikaisesti yksinkertaisen ohjelmointirajapinnan avulla. (17.) Se on suunniteltu skaalautuvaksi yksittäisistä palvelimista tuhansiin koneisiin, joista jokainen tarjoaa paikallista laskentaa ja tallennusta. (18.)

Hadoopin avulla voidaan luoda palvelinklusteri, eli yhdistää virtuaaliset tai fyysiset palvelimet yhdeksi loogiseksi kokonaisuudeksi. Hadoop ei ota kantaa datan laatuun, eikä sen toiminta ole millään tavalla riippuvainen siitä, missä muodossa data on sen tiedostojärjestelmään tallennettu. Sen tunnetuimmat osa-alueet ovat tällä hetkellä Hadoopin oma tiedostojärjestelmä HDFS ja hajautettu analytiikka MapReduce. (3, s. 72–74.)

HDFS tarjoaa skaalautuvan ja luotettavan tiedostojärjestelmän. Käytännössä HDFS virtualisoi jopa tuhansilla palvelimilla olevan levytilan yhdeksi hakemistorakenteeksi, johon voidaan kirjoittaa mitä tahansa dataa. Big datan ideologiaan kuuluu, että tiedostojärjestelmästä ei poisteta dataa, vaan sinne kirjoitetaan koko ajan uutta dataa. Järjestelmä skaalautuu lisäämällä klusteriin noodeja, ja datan lukeminen on nopeaa, koska se tehdään rinnakkaisprosessoinnilla noodien kesken. (1.)

MapReduce on Java-pohjainen ohjelmointimalli, joka on tarkoitettu suurten datamäärien, erityisesti big datan, käsittelyyn. Sen tarkoituksena on rinnakkaistaa ja hajauttaa analytiikka. MapReducella käsitellään HDFS-tiedostojärjestelmässä olevaa dataa halutulla tavalla, kun ollaan kiinnostuneita datan sisällöstä. (1.) MapReduce koostuu kolmesta vaiheesta, jotka ovat Map-, Shuffle- ja Reduce-vaiheet. Ensimmäisenä on Map-vaihe, joka hoitaa datan suodatuksen. Shuffle-vaiheessa Map-vaiheen suorittaneelta palvelimelta saadut välitulokset lähetetään Reduce-vaiheen palvelimelle. Reduce-vaihe yhdistelee ja järjestelee datan luettavaan ja analysoitavaan muotoon. Kuvassa 5 näkyy MapReducen toiminta.

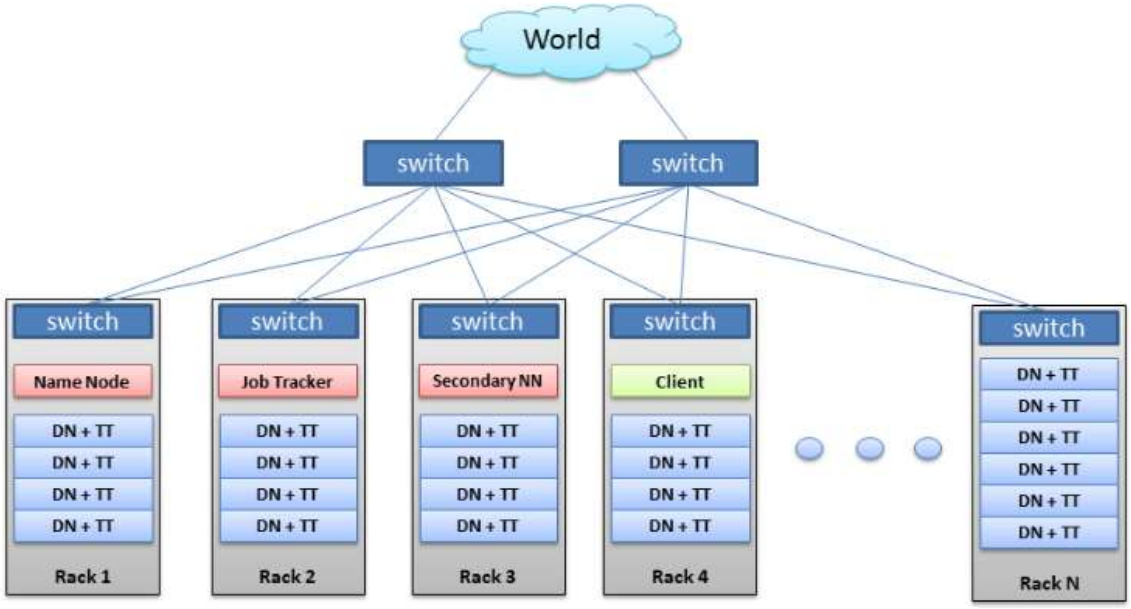


KUVA 5. MapReduce (19)

Kuvassa 6 näkyy Hadoopin arkkitehtuuri. Hadoop-klusterissa on käynnissä useita prosesseja samanaikaisesti. Prosessit ovat NameNode (NN), DataNode (DN), Secondary NameNode (Secondary NN), JobTracker ja Tasktracker (TT). Jokaisella prosessilla on oma rooli klusterin sisällä. NameNode, Secondary NameNode ja JobTracker ovat master-prosesseja ja DataNode sekä TaskTracker ovar slave-prosesseja. (20, s. 20–25.)

Tärkein prosessi Hadoopin sisällä on NameNode, joka toimii HDFS:n masterina ja pitää kirjaa siitä, miten HDFS-tiedostojärjestelmään luotu tiedosto on jaettu datalohkoihin ja mitkä slave-solmut säilövät jaettuja lohkoja. Kun HDFS-tiedosto halutaan lukea tai kirjoittaa, NameNode kertoo, missä klusterin DataNodeissa tiedoston lohkot sijaitsevat. Klusterin slave-solmuissa on käynnissä DataNode-prosessi, joka antaa NameNodelle jatkuvasti tietoa ja kommunikoi muiden DataNodejen kanssa varmuuskopioidakseen omat datalohkonsa. Secondary NameNode on NameNoden apuprosessi, joka valvoo klusterin tilaa sekä kopioi tietyin väliajoin HDFS:n metadatan itselleen. (20, s. 20–25.)

JobTracker toimii MapReducen toimeenpanijana ja määrittää mitä tiedostoja käsitellään, määrää solmuja tietyille tehtäville ja valvoo jokaisen tehtävän suorittamista. Mikäli jokin tehtävä epäonnistuu, JobTracker aloittaa tehtävän automaattisesti uudelleen toisessa solmussa. TaskTracker on jatkuvasti yhteydessä JobTrackeriin ja hoitaa yksittäisten tehtävien toimeenpanon jokaisessa slave-solmussa. (20, s. 20–25.)



KUVA 6. Hadoop-klusterin arkkitehtuuri (21)

Toteutusvaihtoehtoihin perehtymisen jälkeen käytettäväksi alustaksi valittiin Apache Hadoop. Hadoopista löytyi hyvät ja selkeät dokumentaatiot ja sitä oli käytetty useissa muissa klusterin toteutuksissa. Lisäksi Hadoop on ilmainen ja järjestelmä helposti skaalautuva.

4 NEUROVERKOT

Neuroverkolla tarkoitetaan epälineaarista sovitusta tekevää matemaattista ajatusmallia, joka perustuu biologisen eliön hermoston toimintaan (22). Opinnäytetyön osana luotiin neuroverkkoja hyödyntävä sovellus, joten tässä luvussa on käsitelty siihen liittyvää teoriaa.

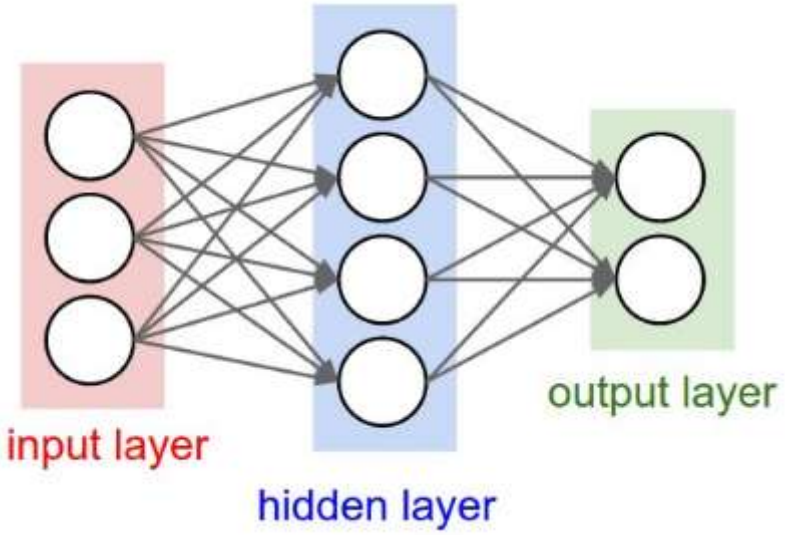
4.1 Neuroverkkojen määritelmä

Tutkijat ovat olleet jo pitkään kiinnostuneita ihmisaivoista ja niiden toiminnasta. Vuonna 1943 kehitettiin ensimmäinen käsitteellinen malli keinotekoisesta neuroverkosta. Keinotekoinen neuroverkko suunniteltiin laskennalliseksi malliksi, joka perustuu aivojen tapaan ratkaista tietynlaisia ongelmia. (23.)

Neuraalilaskennassa käytetään rinnakkaisia yksinkertaisia laskuelementtejä suorittamaan suuria laskentaoperaatioita. Laskuelementtejä kutsutaan neuroneiksi ja koko järjestelmää keinotekoiseksi neuroverkoksi. (24.) Neuronit ovat yksinkertaisia elementtejä, jotka lukevat tulon, prosessoivat sen ja muodostavat lähdön. Kuitenkin useiden neuronien verkosto voi ilmetä uskomattoman älykkäänä käyttäytymiseltään. (23.)

Kuvassa 7 näkyy yksinkertainen neuroverkko, jossa on kolme syöteneuronia, neljä piiloneuronia ja kaksi ulostuloneuronia. Neuronien välissä olevat viivat ovat synapseja, jotka kuvaavat kahden neuronin välistä yhteyttä ja ilmaisevat reitin tiedonkululle. Jokaiselle synapsille määritellään sattumanvarainen painoarvo. Jos verkko muodostaa ”hyvän” ulostulon, painoarvoja ei tarvitse säätää. Jos verkko muodostaa ”huonon” ulostulon, järjestelmä mukautetaan muuttamalla painoarvoja seuraavien tulosten parantamiseksi. Opetusdataa syötetään niin kauan, että oppiminen on saavuttanut halutun tason eli virhe muodostuu riittävän pieneksi. Kasvattamalla piiloneuronien määrää voidaan parantaa approksimaatiota. (25.) Kuitenkin verkon yhteyksien kasvattaminen altistaa verkon ylisovittumiselle. Ylisovitus on yleinen ongelma, jossa malli oppii suoriutumaan hyvin oppimisessa käytetyllä datalla, mutta alisuoriutuu oppimisen aikana näkemättömällä datalla. (26.)

Neuroverkkolaskentaa tullaan käyttämään yrityksessä osana laajempaa tekoäly-aiheista projektia. Hadoop-ympäristö toimii tässä tapauksessa neuroverkon opetusdatan säilytyksessä. Opetusdataa kerätään koko ajan lisää ja neuroverkkoa mukautetaan tällä uudella datalla.



KUVA 7. Yksinkertainen neuroverkko (27)

4.2 Neuroverkon oppiminen

Yksi neuroverkon keskeisiä elementtejä on sen kyky oppia. Neuroverkko on monimutkainen mukautuva järjestelmä, joka pystyy muuttamaan sisäistä rakennettaan läpivirtaavan tiedon perusteella. (23.) Neuroverkkoa ei siis ohjelmoida suoraan vastaamaan tiettyyn syötteeseen tietyllä tavalla, vaan se oppii sille annettujen esimerkkien perusteella (28). Opetus tapahtuu muuttamalla verkon laskentaelementtien ja niiden välisten yhteyksien parametreja verkon saamien syötteiden pohjalta. Yleensä parametrien muuttaminen tapahtuu jollakin iteratiivisella algoritmilla. (24.)

Oppiminen voidaan jakaa ohjattuun (supervised) ja ohjaamattomaan (unsupervised) oppimiseen. Ohjatulla oppimisella tarkoitetaan algoritmeja, joille annetaan opetusdataa, jonka lopputulos on jo tiedossa. Algoritmia opetetaan opetusdatan

syöte-tulosparien avulla siten, että se saavuttaa halutut lopputulokset tietynlaisella syötteellä. Aluksi hankitaan dataa, joka luokitellaan ja jonka avulla algoritmi opetetaan. Opetuksen jälkeen hankitaan uutta dataa, jota ei käsitellä mitenkään, ja sillä testataan algoritmia. Tulokset tarkastetaan ja tarvittaessa algoritmia opetetaan lisää siihen saakka, että saavutetaan halutunlainen lopputulos. (29.)

Ohjaamattomassa oppimisessa tavoitetulosta ei tunneta ennalta ja algoritmin annetaan tehdä omia päätöksiä ilman ulkopuolista avustusta. Ohjaamattomassa oppimisessa ei käytetä syöte-tulospareja oppimiseen. Ohjatun oppimisen tavoitteena on luokitella dataa halutulla tavalla, kun taas ohjaamattomassa oppimisessa etsitään piilossa olevia riippuvuuksia datasta. (29.)

Neuroverkkojen opettamisessa synapsien painot optimoidaan niin, että opetusnäytteiden vasteiden ja haluttujen vasteiden ero minimoituu (30). Yksi yleisimmin käytetyistä optimointimenetelmistä on gradienttimenetelmä. Gradienttimenetelmä on yksinkertainen derivaattoja käyttävä usean muuttujan funktion optimointiin tarkoitettu menetelmä. Menetelmän periaatteena on laskea funktion gradientin perusteella jyrkin laskeva suunta ja optimoida siihen suuntaan otettava askelpituus. (31.)

Neuroverkkojen opettamisessa käytetään sakkofunktiota. Sakkofunktio on mittari, joka mittaa, kuinka hyvin neuroverkko oppii suhteessa sille annettuun opetusnäytteeseen ja odotettuun ulostuloon. Sakkofunktio saa yhden arvon, joka kertoo, miten neuroverkko toimii kokonaisuutena. (32.) Erilaisia sakkofunktioita ovat esimerkiksi keskineliövirhe (Mean Squared Error, MSE) ja jäännösneliösumma (Residual Sum of Squares, RSS). Kaavassa 1 on esitetty keskineliövirhe ja kaavassa 2 jäännösneliösumma.

KAAVA 1. Keskineliövirhe (33)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

missä

n = näytteiden määrä

$\hat{Y}$ = lähdön odotusarvo

Y = approksimaation tulos

KAAVA 2. Jäännösneliösumma (34)

$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

missä

y = approksimaation tulos

x = tulo

f(x_i) = lähdön odotusarvo

4.3 Aktivointifunktiot

Neuronin ulostulon laskennassa käytetään jotain aktivointifunktiota. Neuroverkkojen aktivointifunktioita on useita, mutta tässä opinnäytetyössä perehdytään tarkemmin Sigmoid- ja Tanh-aktivointifunktioihin.

4.3.1 Sigmoid-funktio

Sigmoid-funktio on matemaattinen esitys biologisen hermosolun käyttäytymisestä. Funktio on reaaliarvoinen ja differentioituva. (35.) Sigmoid-funktio on esitetty kaavassa 3. Kuvassa 8 on esitetty Sigmoid-funktion arvo $\sigma(z)$ z-arvon välillä −10–10.

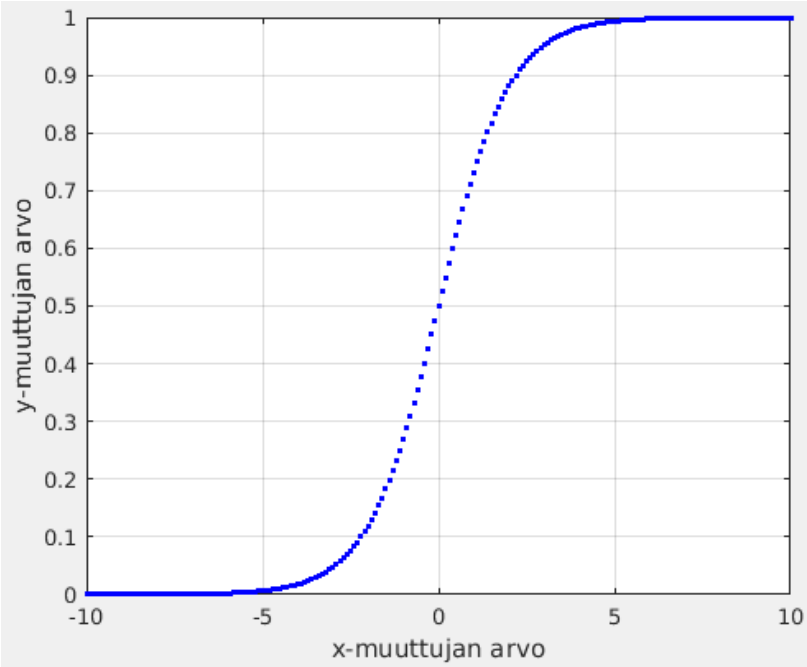
KAAVA 3. Sigmoid-funktio (35)

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

missä

σ = Sigmoid-funktio

z = tulo



KUVA 8. Sigmoid-funktio

Neuronilla voi olla n määrä tuloja x, joiden arvo voi vaihdella tietyllä välillä. Jokaiselle tulolle on painoarvo w1 sekä b eli vakiotermi. Neuronin lähtö tulojen x_n, painoarvon w1 ja vakiotermin b kanssa on esitetty kaavassa 4. (35.)

KAAVA 4. Neuronin lähtö (35)

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

missä

w_j = painoarvo

x_j = tulo

b = vakiotermi

4.3.2 Tanh-funktio

Tanh (Hyperbolic tangent) -funktio antaa arvoja, jotka vaihtelevat välillä (-1,1). Näin ollen vahvasti negatiiviset sisääntulot jäsennellään negatiivisiksi lähdöiksi. Lisäksi vain nolla-arvoiset sisääntulot jäsennellään lähellä nollaa oleviksi lähdöiksi. Nämä ominaisuudet tekevät opetuksen jumittumisesta vähemmän todennäköistä. (36.) Kaavassa 5 näkyy tanh-funktio ja kuvassa 9 on esitetty tanh-funktion arvo $\sigma(z)$ z-arvon välillä −10–10.

KAAVA 5. Tanh-funktio (31)

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

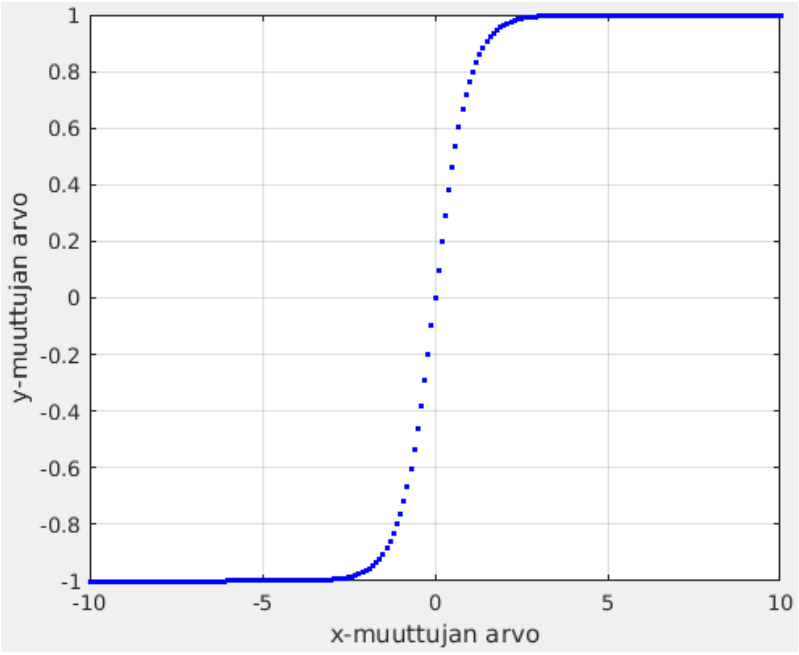
missä

tanh(z) = hyperbolinen tangentti

sinh(z) = hyperbolinen sini

cosh(z) = hyperbolinen kosini

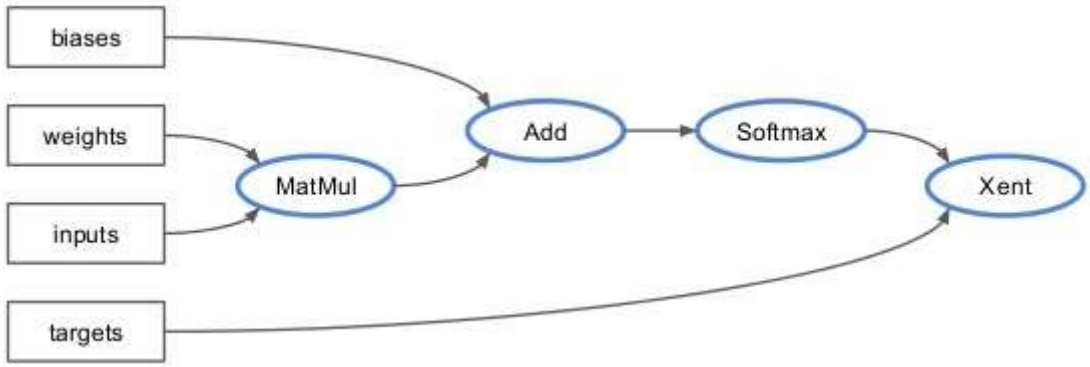
z = tulo



KUVA 9. Tanh-funktio

5 TENSORFLOW

TensorFlow on avoimen lähdekoodin ohjelmistokirjasto numeeriseen laskentaan, joka käyttää tietovirtakaavioita. Kuvassa 10 näkyy TensorFlow'n käyttämä tietovirtakaavio. Kaaviossa esiintyvät noodit esittävät matemaattisia laskutoimituksia ja kuvaajan reunat ovat moniulotteisia datamatriiseja niiden välillä. Joustava arkkitehtuuri mahdollistaa laskemisen yhtä tai useampaa suoritinta tai grafiikkapiiriä käyttämällä työpöydälle, palvelimelle tai mobiililaitteelle yhden rajapinnan välityksellä. Googlen tutkimusorganisaation tutkijat ja insinöörit kehittivät Tensorflow'n alun perin koneoppimista ja neuroverkkojen tutkimista varten, mutta järjestelmää voidaan soveltaa myös monenlaisilla muilla aloilla. (37.)

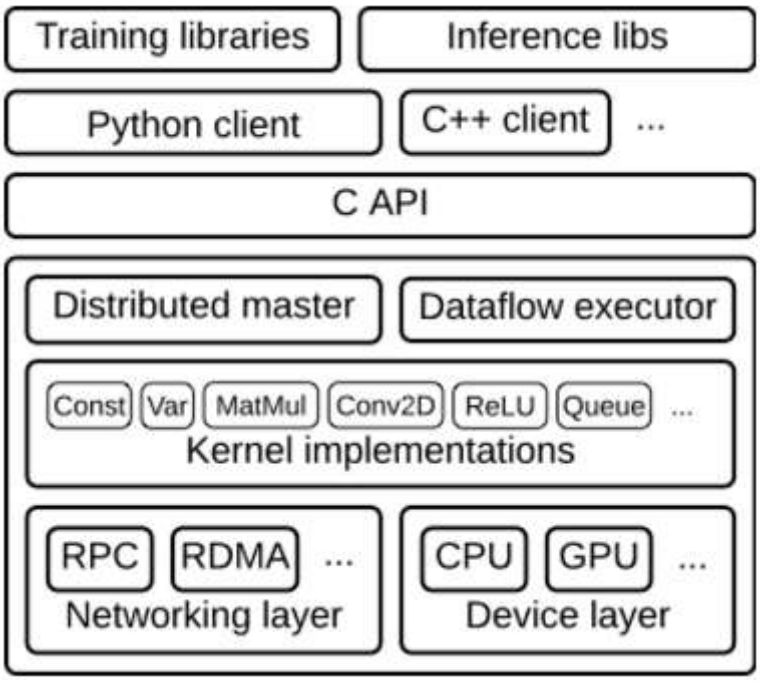


KUVA 10. Tietovirtakaavio (38.)

TensorFlow-ympäristössä voidaan käyttää hajautettua laskentaa. Jokaisella klusterin koneella tulee olla Hadoop ja TensorFlow asennettua, jotta hajautettua laskentaa voidaan hyödyntää. (39.)

Kuvassa 11 näkyy TensorFlow'n yleinen arkkitehtuuri. Tässä opinnäytetyössä käytetään Pythonia ohjelmointikielenä, joten TensorFlow-kuvaaja luodaan Python-clientilla. C API eli ohjelmointirajapinta mahdollistaa eri ohjelmointikielten ja kirjastojen käyttämisen ohjelmassa. Distributed Master karsii ennalta määrätyn alikuvaajan kuvaajasta ja määrittely tapahtuu Session.run()-metodin argumenttien mukaan. Tämän jälkeen Distributed Master jakaa alikuvaajat palasiksi, jotka taas jaetaan Worker Services -lohkoille suoritettaviksi. Worker Services -lohkoja

on yksi joka tehtävään. Lohkot ajoittavat kuvaajan suorituksen käyttäen kernel-toteutuksia (kernel implementations) tarkoituksenmukaisesti käytettävissä olevaan laitteistoon (CPU, GPU). Kernel-toteutukset esittävät laskennan yksittäisen kuvaajan osan toiminnalle; esimerkiksi matmul on matriisin kertolaskun toteutus. (40.)



KUVA 11. TensorFlow-arkkitehtuuri (40)

Klusterissa voi suorittaa useita eri laskentasovelluksia yhtä aikaa. Laskentasovellukset käyttävät yhteistä HDFS-tiedostojärjestelmää, jolloin kaikki data on kaikkien koneiden saatavilla. Tensorflow on yleisesti käytetty koneoppimisympäristö, joka liittyy saumattomasti Hadoop-ympäristöön. Käytännössä tässä opinnäytetyössä Hadoopin HDFS-tiedostojärjestelmästä luetaan TensorFlow'n opetusdataa. Tulevaisuudessa klusteria tullaan hyödyntämään myös TensorFlow'n laskentatöiden hajautetussa suorittamisessa.

6 TESTAUSYMPÄRISTÖN TOTEUTUS

Tässä luvussa kerrotaan testausympäristön toteutuksesta ja testataan valittua alustaa eli Hadoopia oikeassa fyysisessä ympäristössä. Testausympäristö rakennettiin tuotannossa sijaitseviin laboratoriotiloihin.

6.1 Testausympäristön rakentaminen

Klusteri rakennettiin kolmesta TEITO 4U -PC:stä, jotka yhdistettiin Ethernetin kautta toisiinsa. Internet-yhteys jaettiin laskentanoodeille ZyXEL GS-108B v2 -reitittimellä. Koneet yhdistettiin 4-porttiseen Belkin SoHo KVM -kytkimeen, jonka avulla kaikkia klusterin tietokoneita pystyi hallitsemaan yhden näytön, näppäimistön ja hiiren avulla. Kaikki testausympäristön rakentamiseen käytetyt välineet löytyivät valmiiksi yrityksestä. Klusterin fyysinen toteutus näkyy kuvassa 12.

KUVA 12. Klusterin fyysinen toteutus

6.2 Käyttöjärjestelmän asennus

Klusterin tietokoneille asennettiin Red Hat Enterprise Linux 7.1 -jakelupaketti, joka sisälsi yrityksen IT-osaston määrittelemiä asetuksia. Tämä on ymmärrettävää tietoturvan kannalta, mutta tässä tapauksessa aiheutui haasteita esimerkiksi palomuurin ja SSH-yhteyden kanssa. Asennuksen fyysisenä mediana käytettiin USB-muistitikkua. Rufus-ohjelmalla luotiin USB-asennusmedia ISO-levykuvasta. Tietokoneiden BIOS:n asetuksista valittiin ensisijaiseksi käynnistysmediaksi USB. Asennus suoritettiin käyttämällä Nokian omaa asennusopasta (41). Asentaminen aloitettiin laittamalla USB-tikku tietokoneeseen kiinni, jonka jälkeen koneeseen laitettiin virrat päälle.

6.3 Alustan asennus

Hadoop asennettiin asennusoppaan mukaisesti (42). Java on oliopohjainen ohjelmistokieli ja se on ensisijainen vaatimus Hadoopin käynnistämiseen missä tahansa järjestelmässä, joten asennus aloitettiin Javan lataamisella asennusohjeen mukaisesti (43). Javan ja Hadoopin asennukseen käytetyt komennot löytyvät liitteestä 1. Javan asentamisen jälkeen luotiin uusi käyttäjätili ja salasana. Käyttäjän luomisen jälkeen tehtiin avainpohjainen SSH, joka on etäkäyttöohjelmisto, jolla voidaan luoda salattuja yhteyksiä järjestelmästä toiseen. Avaimen toimivuus tarkistettiin kirjautumisella. Avainpohjaisen SSH:n luomisen jälkeen aloitettiin itse Hadoop 2.6.0 lataaminen. Hadoopia asentaessa täytyi konfiguroida useampaa tiedostoa vastaamaan omaa Hadoop-infrastruktuuria. Nämä tiedostot olivat core-site.xml, hdfs-site.xml, yarn-site.xml ja mapred-site.xml. Konfiguroitujen tiedostojen sisällöt on esitetty liitteessä 2. Hadoop 2.6.0 asennettiin yhdelle tietokoneelle ja kovalevy kloonattiin klusterin muille koneille.

6.4 Klusterin testaus

Klusterin toiminta testattiin tekemällä koodi, joka laskee valitusta HDFS-tiedostojärjestelmään tallennetusta tiedostosta sanojen lukumäärän ja laskemiseen kuluneen ajan. Koodi tehtiin Java-ohjelmointikielellä Eclipse-ohjelmointiympäristössä. Koodi ajettiin sekä yhdellä PC:llä että klusterilla. Klusterin todettiin olevan

huomattavasti tehokkaampi kuin yksi PC. Klusterin testaamiseen käytetty koodi on liitteessä 3.

6.5 Tensorflow-ympäristön asennus

Opetusalgoritmia ja neuroverkkoa varten klusterin koneille asennettiin TensorFlow-ympäristö. TensorFlow asennettiin asennusoppaan mukaisesti (44). Tensorflow tukee useaa ohjelmointikieltä, ja aiemman kokemuksen vuoksi päätettiin käyttää Pythonia. Pythonin ajamiseen ladattiin IDLE-editori. Lisäksi asennettiin Jupyter Notebook, joka on avoimen lähdekoodin web-sovellus, jota voidaan käyttää esimerkiksi numeerisessa simuloinnissa, tilastollisessa mallintamisessa ja koneoppimisessa (45). Kaikki TensorFlow-ympäristön asennukseen käytetyt komennot ovat liitteessä 4.

7 LASKENTASOVELLUKSEN LUOMINEN

Laskentasovelluksen tavoitteena oli luoda ohjelma, joka opettaa neuroverkon laskemaan parametreja syötetyn datan perusteella. Aluksi TensorFlow-ympäristöön tutustuttiin testin avulla, jonka jälkeen muodostettiin itse sovellus. Aikataulusyistä johtuen tässä työssä luotiin ainoastaan yksittäisen noodin laskentasovellus. Hajautetun laskennan toteuttaminen on kuitenkin sangen suoraviivainen toteuttaa jatkokehityksenä ja klusteria tullaan tulevaisuudessa käyttämään TensorFlow'n laskentatöiden hajautetussa suorittamisessa.

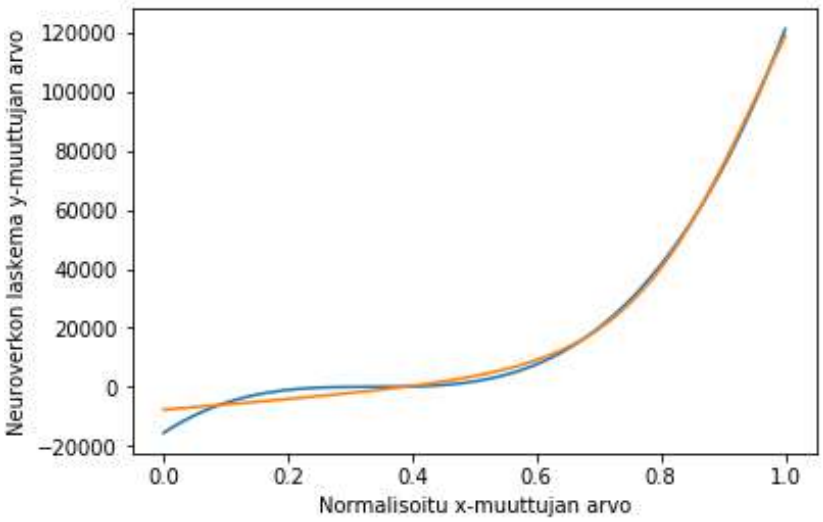
7.1 Ympäristön testaus

Ensimmäiseen testiin luotiin yksinkertainen neuroverkko, joka ratkaisee kolmannen asteen polynomin y-arvon x-arvon avulla. Kolmannen asteen polynomilla testattiin ympäristön toimintaa ja haettiin ymmärrystä TensorFlow-ohjelmointiin. Polynomin sovitusta käytettiin myös erilaisten aktivointifunktioiden sekä optimointimetodien testaamiseen.

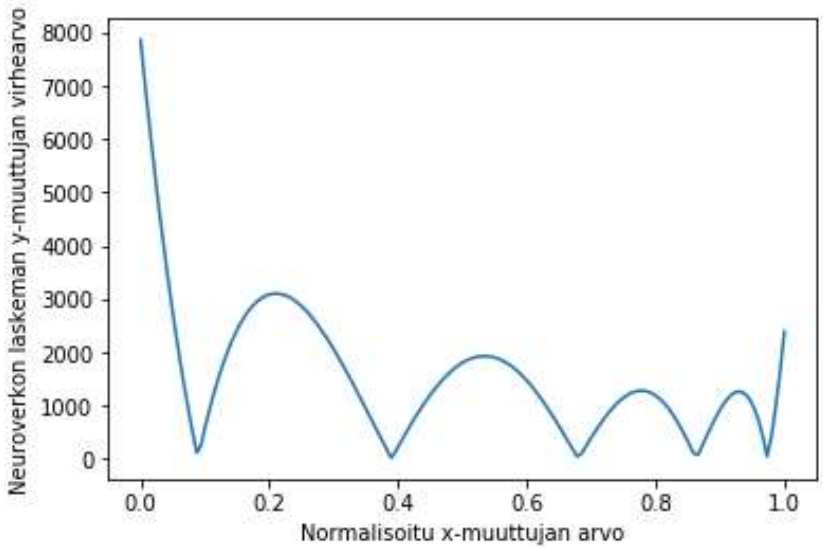
Polynomin arvot luotiin MATLAB-koodin avulla välille −25–49,9 ja arvot kirjoitettiin valittuihin tiedostoihin. Kyseinen MATLAB-koodi löytyy liitteestä 5. Luotujen arvojen avulla muodostettiin opetusalgoritmi ja neuroverkko TensorFlow-ympäristöä ja Python-ohjelmointikieltä käyttäen. Koodissa laskettiin myös virhearvo ja lopuksi muodostettiin Jupyter-sovelluksella kuvaaja polynomista ja neuroverkon sovituksesta sekä virhearvosta. Kommentoitu koodi löytyy liitteestä 6.

Alla on esitetty kuvaajat, miten neuroverkko sovittuu kolmannen asteen polynomiin kahdella eri aktivointifunktiolla, Sigmoid- ja tanh-funktiolla. Lisäksi kummankin tapauksen virhearvon kuvaajat on esitetty. Kuvassa 13 näkyy neuroverkon sovitus ja kuvassa 14 virhearvon kuvaaja Sigmoid-aktivointifunktiota käyttämällä. Sininen viiva kuvaa kolmannen asteen polynomia ja oranssi viiva siihen sovittuvaa neuroverkkoa. Y-akselilla on neuroverkon laskema y-muuttujan arvo ja x-akselilla vastaavasti normalisoitu x-muuttujan arvo. Kuvassa 15 näkyy neuroverkon sovitus ja kuvassa 16 virhearvon kuvaaja tanh-aktivointifunktiota käyttämällä. Kuten kuvaajista huomataan, neuroverkko sovittuu huomattavasti paremmin ja vir-

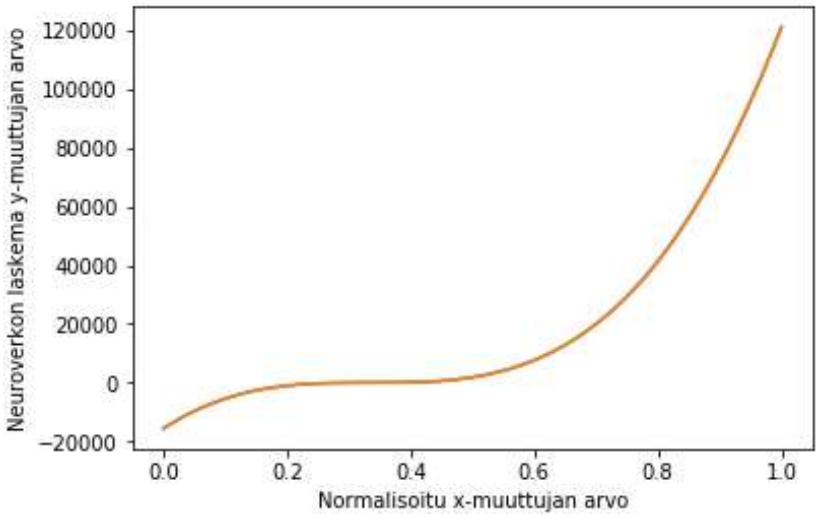
hearvo on pienempi, kun käytetään tanh-aktivointifunktiota Sigmoid-funktion sijaan. Luoduissa arvoissa oli myös negatiivisia arvoja, joten tanh-aktivointifunktio soveltui tähän käyttötarkoitukseen paremmin.



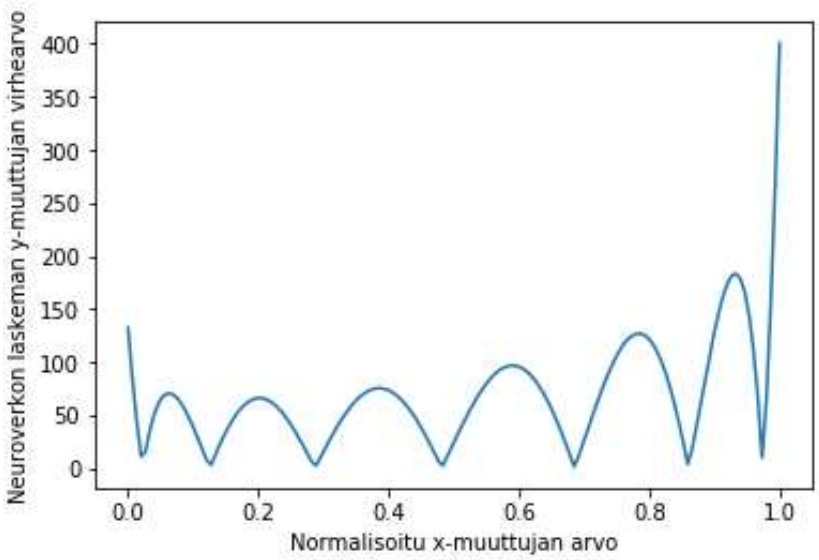
KUVA 13. Kolmannen asteen polynomin ja siihen sovittuvan neuroverkon kuvaaja Sigmoid-aktivointifunktiota käyttäen



KUVA 14. Virhearvon kuvaaja Sigmoid-aktivointifunktiota käyttäen



KUVA 15. Kolmannen asteen polynomin ja siihen sovittuvan neuroverkon kuvaaja tanh-aktivointifunktiota käyttäen

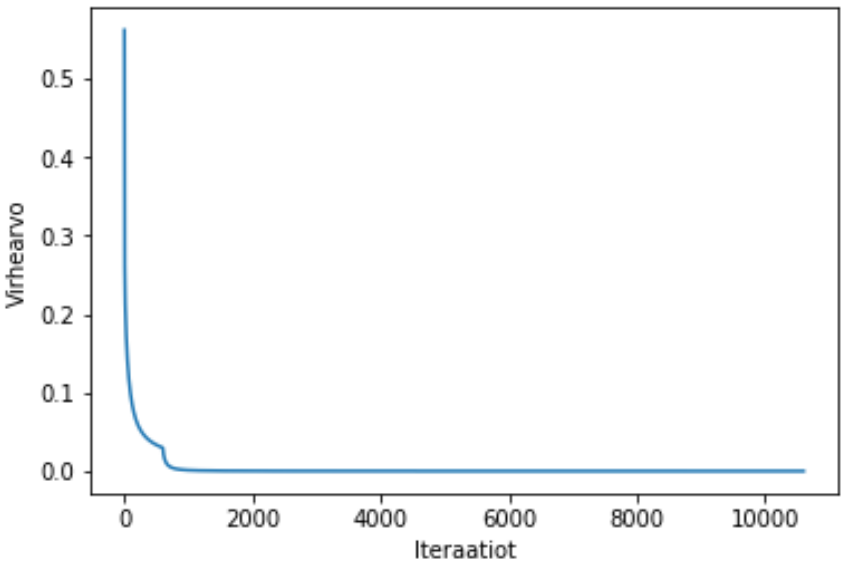


KUVA 16. Virhearvon kuvaaja tanh-aktivointifunktiota käyttäen

7.2 Sovelluksen luominen

TensorFlow-ympäristöön tutustumisen ja erilaisten aktivointifunktioiden ja optimointimetodien testaamisen jälkeen aloitettiin itse laskentasovelluksen luominen. Laskentasovellus opetti neuroverkon laskemaan parametreja syötetyn datan perusteella. Ohjelmassa käytettiin opetusdatana tuotannossa syntyvää prosessi- ja testidataa ja neuroverkon avulla muodostettiin yhteyksiä näiden välille. Opetusdata luettiin Hadoopin HDFS-tiedostojärjestelmästä. Sovelluksessa päädyttiin käyttämään kahta eri optimointimenetelmää, koska testeissä huomattiin virheen olevan pienempi silloin.

Koodissa laskettiin virhearvo optimointi-iteraatioita kohti ja muodostettiin Jupyter-sovelluksella kuvaaja virheestä, joka näkyy kuvassa 17. X-akselilla on iteraatioiden määrä ja y-akselilla sakkofunktion arvo eli virhearvo. Neuroverkko oppi hyvin muodostamaan muuttujien epälineaariset riippuvuussuhteet havaintoaineistosta. Sovelluksen kommentoitu koodi löytyy liitteestä 7.



KUVA 17. Virhearvo optimointi-iteraatioita kohti

8 YHTEENVETO

Työn tavoitteena oli kehittää pilvityyppinen ohjelmisto- ja laiteympäristö algoritmilaskentaan. Lisäksi tavoitteisiin kuului laskentasovelluksen tekeminen, joka opettaa neuroverkon laskemaan parametreja syötetyn datan perusteella. Aikataulutavoitteena oli saada opinnäytetyö valmiiksi kevään 2017 aikana.

Testausympäristön toteutus aloitettiin rakentamalla klusterin fyysinen toteutus yrityksen laboratoriotiloihin. Tämän jälkeen klusterin koneille ladattiin Linux-käyttöjärjestelmä ja palvelinkoneelle asennettiin Hadoop-klusteri, jonka kovalevy kloonattiin noodeille. Klusterin toiminta testattiin koodilla, joka laski valitusta HDFS-tiedostojärjestelmään tallennetusta tiedostosta sanojen lukumäärän ja laskemiseen kuluneen ajan. Tuloksena klusteri suoriutui tehtävästä moninkertaisesti nopeammin verrattuna yhteen PC:hen.

Laskentasovelluksen luomiseen käytettiin TensorFlow-ympäristöä. TensorFlow-ympäristön toimintaa ja eri aktivointifunktioita sekä optimointimetodeita testattiin yksinkertaisella neuroverkolla, joka ratkaisee kolmannen asteen polynomin y-arvon x-arvon avulla. Kolmannen asteen polynomin toteutuksessa todettiin tanh-aktivointifunktion toimivan Sigmoid-funktiota paremmin.

Tuloksena saatiin kehitettyä laskentasovellus, joka opettaa neuroverkon laskemaan parametreja syötetyn datan perusteella. TensorFlow’n opetusdata luettiin Hadoopin HDFS-tiedostojärjestelmästä. Opetusdatana toimi tuotannossa syntyvä prosessi- ja testidata ja neuroverkon avulla muodostettiin yhteyksiä niiden välille. Neuroverkko oppi hyvin muodostamaan muuttujien epälineaariset riippuvuussuhteet havaintoaineistosta.

Yritykselle luotiin tämän opinnäytetyön lisäksi erillinen dokumentti, jossa käytiin yksityiskohtaisesti läpi kaikki asennukseen liittyvät ohjeet ja klusterin toteutuksessa käytetty laitteisto ja ohjelmisto. Dokumentti tehtiin englannin kielellä, jotta sitä voidaan hyödyntää helposti kansainvälisen yrityksen kaikilla toimipisteillä. Dokumentin avulla opinnäytetyössä tehty ohjelmisto- ja laiteympäristö on helppo toteuttaa uudestaan.

Opinnäytetyön aihe oli mielestäni todella mielenkiintoinen ja tarjosi sopivasti haastetta. Aihe oli ajankohtainen ja hyvin laaja. Erilaisia ympäristöjä ja alustoja olisi riittänyt tutkittavaksi, mutta johtuen yrityksessä käynnissä olevasta projektista, päätökset piti tehdä nopeasti ja eteen tuli tilanteita, joissa piti rajoittaa tutkittavaksi otettavia kohteita. Esimerkiksi TensorFlow'n laskentatyön hajautettu toteutus klusterilla jouduttiin aikataulusyistä rajaamaan pois tästä opinnäytetyöstä ja jättämään sovelluksen jatkokehitykseen. Jatkokehitysideoita tuli opinnäytetyön edetessä paljon ja yritys pystyy hyödyntämään klusteria ja laskentasovellusta useisiin käyttökohteisiin tulevaisuudessa.

Kiinnostus työn aiheeseen piti mielenkiinnon yllä ja asetetut tavoitteet saavutettiin ajallaan. Olen päässyt oppimaan paljon uutta ja kokonaisuutena olen erittäin tyytyväinen opinnäytetyön lopputuloksiin. Saavutin kaikki tavoitteeni ja yrityksessä oltiin tyytyväisiä opinnäytetyön lopputuloksiin.

LÄHTEET

1. Hotti, Marko 2012. Pikaperehdytys Big Dataan: Mikä on Apache Hadoop, entä Hive? Tivi. Saatavissa: http://www.tivi.fi/Arkisto/2012-11-28/Pikaperehdytys-Big-Dataan-Mik%C3%A4-on-Apache-Hadoop-ent%C3%A4-Hive-3196648.html. Hakupäivä 22.3.2017.

2. Ohjelmoitavan maailman globaali teknologiajohtaja. Nokia Oyj. Saatavissa: http://www.nokia.com/fi_fi/tietoa-meista/keita-olemme. Hakupäivä 20.1.2017.

3. Salo, Immo 2014. Big data ja pilvipalvelut. Jyväskylä: Docendo.

4. Heino, Petteri 2010. Pilvipalvelut. Hämeenlinna: Alma Talent.

5. Pilvipalvelut. 2013. EDU.fi. Saatavissa: http://www.edu.fi/valo_opas/hankintaopas/pilvipalvelut. Hakupäivä 24.2.2017.

6. What is cloud computing. 2017. A beginner's guide. Microsoft Azure. Saatavissa: https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/. Hakupäivä 24.2.2017.

7. Cloud Computing Concerns and Issues. 2015. Zeendo. Saatavissa: http://zeendo.com/info/what-cloud-computing-means/. Hakupäivä 15.2.2017.

8. NOKIA Cluster Computation thesis document. Internal document.

9. Salo, Immo 2012. Hyötyä pilvipalveluista. Jyväskylä: Docendo.

10. Salo, Immo 2013. Mitä big data on? Big data. Saatavissa: http://www.big-data.fi/artikkelit/mitä-big-data-0. Hakupäivä 9.2.2017.

11. Chowdhry, Lubna Luxmi 2005. Cluster Computing. Code project. Saatavissa: https://www.codeproject.com/Articles/11709/Cluster-Computing. Hakupäivä 9.2.2017.

12. Computer Cluster. 2017. Techopedia. Saatavissa: https://www.techopedia.com/definition/6581/computer-cluster. Hakupäivä 9.2.2017.

13. Cluster Computing / Computer Clusters. BPC, Articles and Glossary. Saatavissa: http://www.bestpricecomputers.co.uk/glossary/cluster-computing.htm. Hakupäivä 22.3.2017

14. Overholt, Kristopher 2015. Distribute Computing on your Cluster with Anaconda. SlideShare. Saatavissa: https://www.slideshare.net/continuumio/distributed-computing-on-your-cluster-with-anaconda-webinar-2015. Hakupäivä 23.2.2017.

15. Introduction. Rocks Cluster Distribution. Saatavissa: http://www.rocksclusters.org/rocks-documentation/4.1/introduction.html. Hakupäivä 9.2.2017.

16. Getting Started. Rocks Cluster Distribution. Saatavissa: http://www.rocksclusters.org/rocks-documentation/4.1/getting-started.html. Hakupäivä 23.2.2017.

17. Hadoop. SAS. Saatavissa: https://www.sas.com/en_us/insights/big-data/hadoop.html. Hakupäivä 16.4.2017.

18. What Is Apache Hadoop. 2017. Apache Hadoop. Saatavissa: http://hadoop.apache.org/. Hakupäivä 9.2.2017.

19. Langit, Lynn 2013. What is Hadoop? Hadoop MapReduce Fundamentals. SlideShare. Saatavissa: https://www.slideshare.net/lynnlangit/hadoop-mapreduce-fundamentals-21427224/3-What_is_Hadoop_Opensource_data. Hakupäivä 16.4.2017.

20. Salo, Immo 2013. Big data - tiedon vallankumous. Jyväskylä: Docendo.

21. Hedlund, Brad 2011. Understanding Hadoop Clusters and the Network. Saatavissa: http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/. Hakupäivä 23.2.2017.

22. Reed, Russell D. - Marks, Robert J. 1999. Neural Smithing. Supervised Learning in Feedforward Artificial Neural Networks. Cambridge: MIT.

23. Shiffman, Daniel. Chapter 10. Neural Networks. The nature of code. Saatavissa: http://natureofcode.com/book/chapter-10-neural-networks/. Hakupäivä 6.3.2017.

24. Himberg, Johan 1997. Neuraalilaskenta ja keinotekoiset neuroverkot. Saatavissa: http://cis.legacy.ics.tkk.fi/jhimberg/dippa/node4.html. Hakupäivä 11.3.2017.

25. Nielsen, Michael 2017. A visual proof that neural nets can compute any function. Neural Networks and Deep Learning. Saatavissa: http://neuralnetworksanddeeplearning.com/chap4.html. Hakupäivä 27.4.2017.

26. Ojala, Timo 2016. Neuroverkkojen regularisointimenetelmät. Tietotekniikan kandidaatintutkielma. Jyväskylä: Jyväskylän yliopisto, tietotekniikan laitos. Saatavissa: https://jyx.jyu.fi/dspace/bitstream/handle/123456789/52722/URN%3ANBN%3Afi%3Ajyu-201701131144.pdf?sequence=1. Hakupäivä 27.4.2017.

27. CS231n Convolutional Neural Networks for Visual Recognition. Saatavissa: http://cs231n.github.io/neural-networks-1/. Hakupäivä 24.3.2017.

28. Honkela, Timo. Neuroverkot: johdatus moderniin tekoälyyn. Saatavissa: http://lipas.uwasa.fi/stes/step96/step96/honkela2/. Hakupäivä 11.3.2017.

29. Kapitanova, Krasimira - Sang H., Son 2012. Machine Learning Basics. Saatavissa: http://www.crcnetbase.com/doi/abs/10.1201/b14300-6. Hakupäivä 27.4.2017.

30. Optimization. CS321n Convolutional Neural Networks for Visual Recognition. Saatavissa: http://cs231n.github.io/optimization-1/. Hakupäivä 16.4.2017.

31. Hallamäki, Antti - Hotokka, Pekka. Epälineaarisen optimoinnin algoritmien toteuttaminen – raportti. Saatavissa: http://users.jyu.fi/~pejuhoto/opiskelu/optimointi/optimointi.html. Hakupäivä 24.3.2017.

32. A list of cost functions used in neural networks, alongside applications. 2015. Keskusteluketju. Saatavissa: http://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications. Hakupäivä 27.4.2017.

33. Khan, Mohammed Emtiyaz 2015. Cost Functions. Saatavissa: https://emtiyaz.github.io/pcml15/cost_function.pdf. Hakupäivä 27.4.2017.

34. Residual Sum of Squares (RSS) - Definition, Formula, Example. EasyCalculation.com. Saatavissa: https://www.easycalculation.com/statistics/learn-residual-sum-squares.php. Hakupäivä 27.4.2017.

35. Nielsen, Michael 2017. Using neural nets to recognize handwritten digits. Saatavissa: http://neuralnetworksanddeeplearning.com/chap1.html#sigmoid_neurons. Hakupäivä 24.3.2017.

36. Derivation: Derivatives for Common Neural Network Activation Functions. The Clever Machine. Saatavissa: https://theclevermachine.wordpress.com/tag/tanh-function/. Hakupäivä 16.4.2017.

37. About TensorFlow. TensorFlow. Saatavissa: https://www.tensorflow.org/. Hakupäivä 3.3.2017.

38. Machine Intelligence at Google Scale: TensorFlow. 2016. SlideShare. Saatavissa: https://www.slideshare.net/HadoopSummit/machine-intelligence-at-google-scale-tensorflow. Hakupäivä 4.4.2017.

39. How to run TensorFlow on Hadoop. 2017. TensorFlow. Saatavissa: https://www.tensorflow.org/deploy/hadoop#hdfs. Hakupäivä 11.3.2017.

40. TensorFlow Architecture. 2017. TensorFlow. Saatavissa: https://www.tensorflow.org/extend/architecture. Hakupäivä 24.3.2017.

41. NOKIA Enterprise Linux 7. Internal document.

42. How to Setup Hadoop 2.6.0 (Single Node Cluster) on CentOS/RHEL and Ubuntu. 2016. TecAdmin.net. Saatavissa: http://tecadmin.net/setup-hadoop-2-4-single-node-cluster-on-linux/#. Hakupäivä 1.3.2017.

43. Kumar, Rahul 2017. How to Install JAVA 8 (JDK/JRE 8u121) on CentOS/RHEL and Fedora. Saatavissa: https://tecadmin.net/install-java-8-on-centos-rhel-and-fedora/. Hakupäivä 27.3.2017.

44. Hoolihan, Tim 2016. Installing TensorFlow on CentOS. Saatavissa: http://hoolihan.net/blog-tim/2016/03/02/installing-tensorflow-on-centos/. Hakupäivä 27.3.2017.

45. The Jupyter Notebook. 2017. Jupyter. Saatavissa: http://jupyter.org/. Hakupäivä 27.3.2017.

LIITTEET

Liite 1 Hadoopin asennuksen komennot

Liite 2 Hadoopin konfiguroitavat tiedostot

Liite 3 Klusterin testaamiseen käytetty koodi

Liite 4 TensorFlow-ympäristön asennuksen komennot

Liite 5 Matlab-koodi arvojen luomiseen

Liite 6 Kolmannen asteen polynomin koodi

Liite 7 Koodi, joka opettaa neuroverkon laskemaan parametreja syötetyn datan perusteella

```
$ su root
```

```
# cd /opt/
```

```
# wget --no-cookies --no-check-certificate --header "Cookie:
gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-
securebackup-cookie" "http://download.oracle.com/otn-
pub/java/jdk/8u121-b13/e9e7ea248e2c4826b92b3f075a80e441/jdk-
8u121-linux-x64.tar.gz"
```

```
# tar xzf jdk-8u121-linux-x64.tar.gz
```

```
# java -version
```

```
# adduser hadoop
```

```
# passwd hadoop
```

```
# su - hadoop
```

```
$ ssh-keygen -t rsa -P ””
```

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
$ chmod 0600 ~/.ssh/authorized_keys
```

```
$ ssh localhost
```

```
$ exit
```

```
$ cd ~
```

```
$ wget http://apache.claz.org/hadoop/common/hadoop-2.6.0/hadoop-
2.6.0.tar.gz
```

```
$ tar xzf hadoop-2.6.0.tar.gz
```

```
$ sudo mv hadoop-2.6.0 hadoop
```

```
$ sudo chown -R hadoop /home/hadoop/hadoopdata/
```

```
$ cd $HADOOP_HOME/sbin/
```

```
$ source ~/.bashrc
```

```
$ nano /home/hadoop/hadoop/etc/core-site.xml
```

```
$ nano /home/hadoop/hadoop/etc/hdfs-site.xml
```

```
$ nano /home/hadoop/hadoop/etc/mapred-site.xml
```

```
$ nano /home/hadoop/hadoop/etc/yarn-site.xml
```

```
$ hdfs namenode -format
```

```
$ start-dfs.sh
```

```
$ start-yarn.sh
```

```
$ jps
```

core-site.xml

```
<configuration>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://master:9000/</value>
                <description>NameNode URI</description>
        </property>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/home/hadoop/hadoop/tmp/hadoop-
                ${user.name}</value>
        </property>
</configuration>
```

hdfs-site.xml

```
<configuration>
        <property>
                <name>dfs.datanode.data.dir</name>
                <value>file:///home/hadoop/ha-
                doopdata/hdfs/datanode</value>
                <description>DataNode directory</description>
        </property>
        <property>
                <name>dfs.namenode.name.dir</name>
                <value>file:///home/hadoop/ha-
                doopdata/hdfs/namenode</value>
                <description>NameNode directory</description>
        </property>
        <property>
                <name>dfs.replication</name>
                 <value>2</value>
        </property>
         <property>
                <name>dfs.permissions</name>
                <value>false</value>
        </property>
        <property>
                <name>dfs.datanode.use.datanode.hostname</name>
                <value>false</value>
        </property>
        <property>
                <name>dfs.namenode.http-address</name>
                <value>master:50090</value>
                <description>Your Secondary NameNode hostname for http ac-
                cess.</description>
        </property>
</configuration>
```

yarn-site.xml

```
<configuration>
        <property>
                <name>yarn.nodemanager.aux-services</name>
                <value>mapreduce_shuffle</value>
        </property>
        <property>
                <name>yarn.nodemanager.aux-services.mapreduce_shuf-
                fle.class</name>
                <value>org.apache.hadoop.mapred.ShuffleHandler</value>
        </property>
        <property>
                <name>yarn.resourcemanager.resource-tracker.address</name>
                <value>master:8031</value>
        </property>
        <property>
                <name>yarn.resourcemanager.scheduler.address</name>
                <value>master:8030</value>
        </property>
        <property>
                <name>yarn.resourcemanager.address</name>
                <value>master:8032</value>
        </property>
        <property>
                <name>yarn.resourcemanager.webapp.address</name>
                <value>master:8088</value>
        </property>
        <property>
                <name>yarn.resourcemanager.admin.address</name>
                <value>master:8033</value>
        </property>
</configuration>
```

mapred-site.xml

```
<configuration>
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
</configuration>
```

```
package WordCount;

import java.io.File;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.commons.lang3.time.StopWatch;
import java.util.StringTokenizer;
import java.util.concurrent.TimeUnit;
import javax.swing.JFileChooser;

public class WordCount1 {

    private static File getFile(){

        JFileChooser chooser = new JFileChooser(".");
        chooser.setDialogTitle("Select File To Count Words:");
        int retval = chooser.showOpenDialog(null);
        File f =null;
        chooser.grabFocus();
        if (retval == JFileChooser.APPROVE_OPTION)
            f = chooser.getSelectedFile();
        return f;
    }

    public static void main(String [] args) throws Exception{
    Configuration c = new Configuration();
    Path output =  new Path("hdfs://master:9000/words.txt");
    Path outputResults =  new Path("hdfs://master:9000/results.txt");
    StopWatch timer = new StopWatch();
    c.set("fs.default.name", "hdfs://master:9000");
    File inputFile = getFile();
    Path input = new Path(inputFile.getAbsolutePath());
    FileSystem hdfs = FileSystem.get(c);
    timer.start();
    hdfs.copyFromLocalFile(input, output);
    Job j= Job.getInstance(c);
    j.setJobName("WordCount");
    j.setJarByClass(WordCount.class);
    j.setMapperClass(MapForWordCount.class);
    j.setReducerClass(ReduceForWordCount.class);
    j.setOutputKeyClass(Text.class);
    j.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(j, output);
    FileOutputFormat.setOutputPath(j, outputResults);
    timer.stop();
    System.out.println("Computation time: " + timer.getTime(TimeUnit.SECONDS) + " s");
    System.exit(j.waitForCompletion(true)?0:1);

    }
```

```
    public static class MapForWordCount
        extends Mapper<LongWritable, Text, Text, IntWritable>{

    public void map(LongWritable key, Text value, Context con
                    ) throws IOException, InterruptedException{
    final IntWritable one = new IntWritable(1);
        Text word = new Text();
            StringTokenizer itr = new StringTokenizer(value.toString().toLowerCase());
            while (itr.hasMoreTokens()){
                word.set(itr.nextToken());
                con.write(new Text (word), one);
        }
    {
      Text outputKey = new Text(word.toString().toUpperCase().trim());
      IntWritable outputValue = new IntWritable(1);
      con.write(outputKey, outputValue);
    }
    }
    }
    public static class ReduceForWordCount
        extends Reducer<Text, IntWritable, Text, IntWritable>{

    public void reduce(Text word, Iterable<IntWritable> values, Context con
                       ) throws IOException, InterruptedException{
        int sum = 0;
        for(IntWritable value : values){
            sum += value.get();
        }
        con.write(word, new IntWritable(sum));
        System.out.println(word.toString() + " " + sum);
        }
    }
}
```

```
$ su root
# yum install epel-release
# yum install gcc
# yum install python-pip
# yum install python-devel
# yum install atlas
# yum install atlas-devel
# yum install gcc-gfortran
# yum install openssl-devel
# yum install libffi-devel
# pip install --upgrade virtualenv
# virtualenv --system-site-packages ~/venvs/tensorflow
# source ~/venvs/tensorflow/bin/activate
# pip install --upgrade tensorflow
# yum install python-tools
# idle
# pip install jupyter
# jupyter notebook
```

```
x = -25:0.5:49.9;
y = x.^3;

fileID = fopen('testi.txt', 'w');
for k = 1:length(x)
    xn = x(k);
    fprintf(fileID,'%f\n',x(k));
end

fclose(fileID);

fileID = fopen('output.txt', 'w');
for k = 1:length(y)
    yn = y(k);
    fprintf(fileID,'%f\n',y(k));
end

fclose(fileID);
```

```
# import libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# load text files
X_data = np.loadtxt(r'/home/hadoop/Documents/Python/testi.txt',delimiter=None)
T_data = np.loadtxt(r'/home/hadoop/Documents/Python/output.txt',delimiter=None)

# transpose of a vector
X_data = X_data[:,None]
T_data = T_data[:,None]

# print the shape of an array
print(X_data.shape)
print(T_data.shape)

# initialize variables
n_samples = X_data.shape[0]
n_input = 1
n_hidden = 3
n_output = 1
```

```
# normalize training data
X_max = np.amax(X_data, axis=0)
X_min = np.amin(X_data, axis=0)
T_max = np.amax(T_data, axis=0)
T_min = np.amin(T_data, axis=0)
X_data = (X_data - X_min) / (X_max - X_min)
T_data = (T_data - T_min) / (T_max - T_min)

# cost function and optimizer
cost = tf.reduce_mean(tf.square(t - y))
optimizer1 = tf.train.AdamOptimizer(0.001).minimize(cost)

# initialize TensorFlow session
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

# run optimizer and cost function
for epoch in range(120000):
    _, c = sess.run([optimizer1, cost], feed_dict={x: X_data, t: T_data})
    if (epoch % 100) == 0:
        print([epoch, c])

# compute and print neural network output
result = sess.run([y], feed_dict={x: X_data})
result = result * (T_max - T_min) + T_min
print (result)

# close TensorFlow session
sess.close()

# denormalize data and calculate error between the desired and obtained value
T_data = T_data * (T_max - T_min) + T_min
errorValue = np.absolute(T_data - result)
```

```
# plot neural network fit and error value
plotresult = np.reshape(result,(150,1))
xdataPlot = np.reshape(X_data,(150,1))
plt.figure(1)
plt.plot(X_data, T_data)
plt.plot(xdataPlot, plotresult)
plt.xlabel('Normalisoitu x-muuttujan arvo')
plt.ylabel('Neuroverkon laskema y-muuttujan arvo')
plt.figure(2)
tdataPlot = np.reshape(T_data,(150,1))
errorValuePlot = np.reshape(errorValue,(150,1))
plt.plot(xdataPlot, errorValuePlot)
plt.xlabel('Normalisoitu x-muuttujan arvo')
plt.ylabel('Neuroverkon laskeman y-muuttujan virhearvo')
plt.show()
```

```
# import libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pydoop.hdfs as hdfs

# loading text files
input = []
with hdfs.open('hdfs://master:9000/testi2.txt') as f:
    for line in f:
        input.append(map(float, line.split(',')))
input = np.asarray(input)

output = []
with hdfs.open('hdfs://master:9000/output2.txt') as f:
    for line in f:
        output.append(map(float, line.split(',')))
output = np.asarray(output)

# print the shape of an array
print(input.shape)
print(output.shape)

# initialize variables
n_samples = input.shape[0]
n_input = input.shape[1]
n_hidden1 = 80
n_output = output.shape[1]
```

```
# the definition of variables and initialization of neural network layers
x = tf.placeholder(tf.float32, [None, n_input])
tf.expand_dims(x,1)
W1 = tf.Variable(tf.random_normal([n_input, n_hidden1], stddev=0.1))
b1 = tf.Variable(tf.random_normal([n_hidden1], stddev=0.1))
h1 = tf.nn.tanh(tf.matmul(x,W1) + b1)
W2 = tf.Variable(tf.random_normal([n_hidden1, n_output], stddev=0.1))
b2 = tf.Variable(tf.random_normal([n_output], stddev=0.1))
y = tf.matmul(h1, W2) + b2
t = tf.placeholder(tf.float32, [None, n_output])

# cost function and optimizers
cost = tf.reduce_mean(tf.square(t - y))
optimizer1 = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
optimizer2 = tf.train.AdamOptimizer(0.0001).minimize(cost)
optimizer1_iter = 600
optimizer2_iter = 5000
errors = np.zeros(optimizer1_iter + optimizer2_iter)
check = 0

# initialize TensorFlow session
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

# run optimizer1 and cost function
for epoch in range(optimizer1_iter):
    _, c = sess.run([optimizer1, cost], feed_dict={x: input, t: output})
    errors[check] = c
    check = check + 1
    print([epoch, c])
```

```
# run optimizer2 and cost function
for epoch in range(optimizer2_iter):
    _, c = sess.run([optimizer2, cost], feed_dict={x: input, t: output})
    errors[check] = c
    check = check + 1
    if (epoch % 10) == 0:
        print([epoch, c])

# print neural network output
result = sess.run(y, feed_dict={x:input})
print(result)

# close TensorFlow session
sess.close()

# plot error value per optimizer iterations
plt.figure(1)
plt.plot(errors)
plt.xlabel('Iteraatiot')
plt.ylabel('Virhearvo')
plt.show()
```