Avoimen lähdekoodin neuroverkot: Työkaluina Ml5.js ja Knime

Neuroverkkojen luominen käyttäen Knime-sovellusta ja Ml5.js-kirjastoa

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus
Kevät, 2021

Joonas Solkinen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Hämeenlinnan korkeakoulukeskus

Tekijä Joonas Solkinen Vuosi 2021

Työn nimi Avoimen lähdekoodin neuroverkot: Työkaluina Ml5.js ja Knime

Neuroverkkojen luominen käyttäen Knime-sovellusta ja Ml5.js-kirjastoa

Ohjaajat Lasse Seppänen

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli vertailla neuroverkkojen luomiseen saatavilla olevia avoimen lähdekoodin järjestelmiä. Työhön valittiin kaksi järjestelmää, Knime ja Ml5.js-kirjasto. Vertailua varten molemmilla järjestelmillä luotiin neuroverkot.

Opinnäytetyön tietopohja koostui aiheeseen liittyvästä kirjallisuudesta, verkkosivuista ja käytettyjen järjestelmien dokumentaatiosta. Opinnäytetyön teoriaosuudessa avataan neuroverkkojen toimintaa ja käyttömahdollisuuksia. Toiminnallisessa osuudessa rakennetut neuroverkot käydään läpi vaihe vaiheelta. Opinnäytetyö oli toiminnallinen.

Neuroverkkojen opettamiseen käytettiin jääkiekkopelaajien tilastoja. Neuroverkkojen tavoitteena oli ennustaa pelaajien tulevan kauden pisteet. Saaduista tuloksista voidaan todeta, että järjestelmissä on hyvin paljon samaa ja suurin ero löytyykin niiden käyttömahdollisuuksissa. Ml5.js toimii hyvin kevyenä ja helposti käytettävänä lisänä verkkosivuilla. Knime taas toimii hyvän laajennus mahdollisuutensa puolesta hyvin neuroverkkojen käyttämiseen analysoinnin apuna.

Avainsanat Neuroverkot, Ml5.js, Knime, Tekoäly, Jääkiekko

Sivut 38 sivua ja liitteitä 4 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Abstract

Author Joonas Solkinen

Year 2021

Subject Open-source neural networks with Ml5.js and Knime

Building neural networks using Knime software and Ml5.js library

Supervisors Lasse Seppänen

ABSTRACT

The purpose of the thesis was to compare open-source systems that provide tools to build neural networks. Selected systems for the thesis were Knime and Ml5.js library. To help in comparison, neural networks were built with both systems.

Theory of this thesis was based on literature and websites about the thesis subject. Also, documentation from the systems used in the thesis were used. In the theory part of the thesis goal was to give basic understanding of how neural networks and what are their possibilities. In functional part covers neural network build process with both systems. The thesis was functional.

Hockey player stats were used to teach neural networks. Goal with the neural networks was to predict players next seasons points. Results indicated that there is lot of similarities and the be found in how they can be used. Result obtained from the neural networks had only slight difference in them. Ml5.js works great as a light and easy to use add-on for websites. Knime on the other hand works great in more analysis-based work, thanks to its good extension possibilities.

Keywords Neural network, Ml5.js, Knime, AI, Hockey

Pages 38 pages and appendices 4 pages

Sanasto

HTML	HyperText Markup Language, verkkosivujen määrittelykieli
JavaScript	Koodikieli
JSON	JavaScript Object Notation, Dataformaatti
CSV	Comma-separated Values, Dataformaattii, jossa tiedot on eroteltu pilkulla
Python	Ohjelmointikieli

Sisällys

Kuvat, ohjelmakoodit ja taulukot

Liitteet

1 Johdanto

Tekoäly on aiheena kiinnostava sen mahdollisuuksien vuoksi. Nykypäivänä tekoälyä käytetään paljon eikä peruskäyttäjä välttämättä tiedä, että hyödyntää tekoälyä. Tekoäly helposti kuulostaa kaukaiselta ja pelottavalta. Ensimmäisenä mieleen saattaa tulla terminaattorit ja muut uhkakuvat, joita näkee paljon populaarikulttuurissa. Tekoäly ei kuitenkaan aina tarkoita isoja järjestelmiä, jotka tekevät kaiken itse ja joita käyttävät vain robotit ja tiedemiehet. Hyvä esimerkki yleisestä käyttötarkoituksesta on sellainen, jonka näkee lähes päivittäin, kuten Chatbot, joita on nykyään lähes kaikilla nettisivuilla. Chatbotit käyttävät tekoälyä määritelläkseen, minkä kysymyksen käyttäjä on kirjoittanut ja mikä olisi siihen oikea vastaus. Tällaisen toiminnallisuuden koodaaminen ilman tekoälyä on aikaa vievää ja lähes mahdotonta, kun ottaa huomioon, kuinka monella tapaa ihmiset kysyvät samaa asiaa.

Lähdin siis etsimään erilaisia mahdollisuuksia lähestyä aihetta. Ensiksi vastaan tuli Ml5.js-kirjasto, joka on kehitetty olemaan helppo askel tekoälyn maailmaan. (Ml5.js kotisivut). Halusin kuitenkin saada lisää näkökulmaa tekoälyn mahdollisuuksista. Tutkittuani lisää erilaisia mahdollisuuksia tulin siihen tulokseen, että Knime olisi hyvä vastapaino JavaScript-kirjastolle, jolla pystytään painottamaan enemmän visuaalista puolta kuin Knimellä. Valintoihini vaikutti myös se, että molemmat ovat avoimen lähdekoodin ilmaisia vaihtoehtoja.

Tekoälyn luomiseen on paljon erilaisia tapoja (Kananen ym., 2019, s 27). Opinnäytetyössä keskityn kuitenkin vain yhteen menetelmään ja yhteen opetustapaan. Menetelmänä toimii neural network eli suomeksi neuroverkko. Työn tarkoituksena on antaa pinnallinen ymmärrys neuroverkkojen toiminnasta, jonka avulla pystyy rakentamaan toimivan neuroverkon.

Tutkimuskysymykset:

- Miten neuroverkko toimii ja mitä se vaatii?
- Millä tavoin Knimellä tehty sovellus toimii verrattuna Ml5.js tehtyyn?
- Millä tavoin Knime ja Ml5.js ovat samanlaiset tai erilaiset?
- Miten tehtyjä järjestelmiä voidaan hyödyntää?

2 Neuroverkot teoriassa ja työssä käytettävät järjestelmät

Opinnäytetyön teoria jakaantuu kahteen osaan. Ensiksi käydään läpi neuroverkkojen toimintaa ja teoriaa sen taustalla. Toinen osa koostuu toiminnallisessa osassa käytettävistä työkaluista.

Teoriaosuuden tavoitteena on avata käsitteitä ja teoriaa toiminnallisessa osassa käytettävien sovellusten taustalla. Teoria saattaa vaikuttaa aluksi haastavalta ja monimutkaiselta. Tavoitteena on kuitenkin selittää teoria, niin että se olisi ainakin pintapuolisesti ymmärrettävissä ilman aikaisempaa tuntemusta aiheesta, jotta kaikkien olisi mahdollista saada yleiskäsitys aiheesta.

2.1 Tekoäly

Tekoäly on käsitteenä laaja ja sen alle lukeutuu useita erilaisia tapoja ja teorioita. Usein uutisissa ja populaarikulttuurissa näkee robotteja tai muita hyvin monimutkaisia ja futuristisia tekoälyn muotoja. Tekoälyä hyödyntääkseen saavutettavan työn ei kuitenkaan tarvitse olla monimutkainen kuten robotti, jotta siinä voitaisiin hyödyntää tekoälyä. Tekoäly mahdollistaa tietokoneiden ja muiden laitteiden toimimisen ja itsenäisen oppimisen. Tekoäly mahdollistaa työn automatisoinnin, johon muuten tarvittaisiin ihmisen päättelykykyä tai älykkyyttä, esimerkiksi kirjoitetun tekstin tunnistaminen (Marr, B., 2019, s. 3–4).

Koneiden tekoäly hyödyntää toiminnassaan sääntöjä tai algoritmeja, joiden avulla kone tekee päätöksen siitä, mikä on haluttu lopputulos tai miten tulisi toimia. Jotta koneet voisivat oppia tekemään päätöksen, täytyy niille syöttää paljon tietoa, jota analysoimalla ne voivat luoda oppimismallin, johon ne hyödyntävät annettuja sääntöjä tai algoritmeja voidakseen antaa oikean vastauksen. Ongelma sääntöihin ja algoritmeihin tukeutuvissa tekoälyissä on se, että joskus tällaisten oppimiseen tarvittavien sääntöjen luominen on vaikeaa ihmisille. Esimerkiksi tunnistettaessa eläinlajeja osan eläimistä pystyy hyvin erottamaan yksinkertaisilla säännöillä, kun taas sääntöjen keksiminen kissaeläinten erottamiseksi toisistaan on jo haastavampaa. Tällaisiin toimintoihin voidaan käyttää esimerkiksi neuroverkkoja, joita hyödynnetään tässä opinnäytetyössä (Marr, B., 2019, s. 3–4).

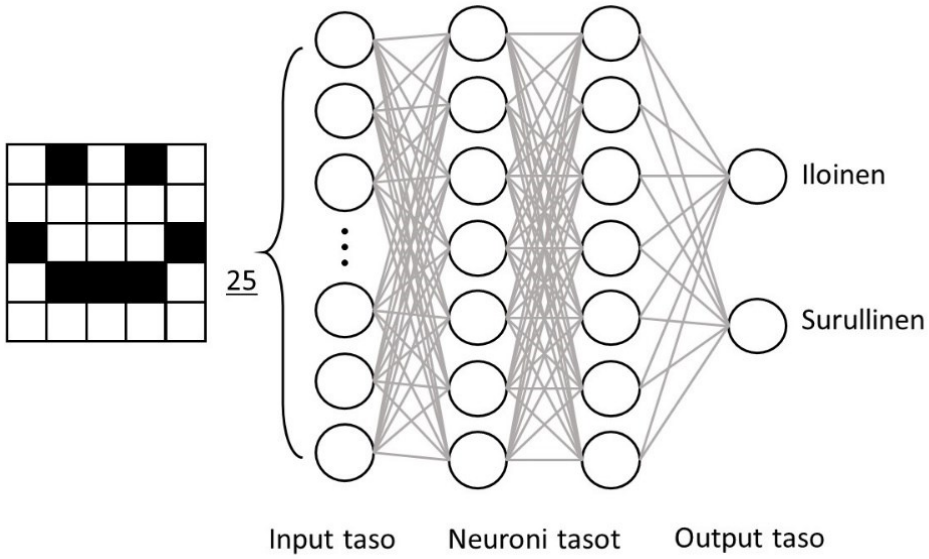
Nykyään tekoälyä on kaikkialla, ja sitä käytetään ja hyödynnetään päivittäin, usein tietämättä. Monet hyvin yksinkertaisiltakin tuntuvat työt tarvitsevat tekoälyä hyvin toimiakseen. Esimerkiksi Instagramissa olevat kuvafiltterit, joiden avulla voi vaikka suurentaa silmiäsi, hyödyntävät tekoälyä tunnistaakseen, missä kasvot kuvassa ovat, jotta halutut efektit tulisivat oikeille paikoilleen (SparkAr, 2021). Tallainen toimivuus vaatii paljon materiaalia, eli tässä tapauksessa erityisesti paljon kasvokuvia. Ottaen huomioon, kuinka monta kasvokuvaa Facebookilla on käytettävissä tekoälyn opettamiseen, ei yllätä, että tämän tyyppiset työkalut toimivat todella hyvin (Marr, B., 2019 s. 49 -- 51).

2.2 Neural Network

Neural network eli neuroverkko on yksinkertaistettu versio aivojen toiminnasta, jossa neuronit lähettävät signaalin eteenpäin toisiin neuroneihin. Yksinkertaistettuna neuronien aktivoituminen määrittää, miten reagoimme tai miksi tunnistamme jonkin asian, jonka näemme. Neuroverkot eivät ole yhtä tehokkaita ja monimutkaisia kuin aivot. Tämä ei kuitenkaan tarkoita, että ne olisivat huonoja tai käyttökelvottomia. Neuroverkot ja niiden muunnokset ovat hyvin monipuolisia, ja niitä voidaan hyödyntää useissa eri tehtävissä. (Keller ym., 2016, s 7–9)

Neuroverkot koostuvat neuroneista, joista koostuu neuronien verkko. Yksinkertaisuudessaan neuroni sisältää vain arvon, joka siirtyy eteenpäin. Tieto kulkee neuroverkossa neuronien välillä tasosta toiseen. Jokaisella neuronien välisellä yhteydellä ja sen arvolla on vaikutus saatavaan lopputulokseen. Neuroverkkojen toiminnan voi jakaa kolmeen eri osaan, jolla neuronit toimivat: input eli tiedonsyöttövaihe, neuronitaso tai hidden layer, eli taso, jossa niin sanotusti punnitaan vaihtoehdot, ja kolmantena osana on output eli lopputulos. (Kananen ym., 2019, s. 127–133)

Kuva 1 Monitasoisen neuroverkon rakenne.



Input-vaiheessa neuroverkko saa tiedon, josta se laskee sille opetetun lopputuloksen. Kuvan esimerkissä neuroverkko arvioi, onko kuvan hymiö iloinen vai surullinen. Koska kuvan koko on 25 pikseliä, myös vastaaviin kuviin opetetun neuroverkon input-taso on 25 neuronin kokoinen. Input-tason neuroneiden arvo vastaa yhtä kuvan pikselin arvoa. Neuroverkot voidaan kehittää vastaanottamaan lähestulkoon mitä vain tietoa ja missä vain muodossa (Kananen ym., 2019, s. 127–133).

Input-vaiheesta siirrytään neuronitasolle. Kaikilla neuroneilla on tietynlainen painoarvo, jonka vuoksi arvo vaihtelee saadun datan mukaan. (Kananen ym., 2019, s 127–133) Arvon suuruus vaihtelee yleensä arvon 0 ja 1 välillä. Neuronin sisällä tapahtuu siis laskutoimitus, joka määrittää arvon, jonka neuroni lähettää eteenpäin. Tämä laskukaava tarvitsee erilaisia lukuja. Jokaiseen syötteeseen, joka saapuu neuroniin, liittyy kaksi arvoa: syötteen arvo ja paino. Näistä luvuista neuroni laskee uuden arvon, jonka lähettää eteenpäin. Lasku yksinkertaisuudessaan on Paino1 x Syöte1 = uusi arvo. (Warwick, K., 2011, s.94–98) Neuronista arvon matka jatkuu eteenpäin joko seuraavalle neuronitasolle tai output-tasolle.

Output-tasolla neuroneista saadut arvot kertovat, mikä lopputulos on tai pitäisi olla. Esimerkissä (Kuva 1) output-tason neuronin, joka vastaa iloista hymiötä, pitäisi olla arvoltaan suurempi kuin surullisen hymiön neuroni.

Monitasoisessa neuroverkossa on useita neuronitasoja ja jokaisessa tasossa on useita neuroneita. Viimeiseen tasoon, eli aiemmin mainittuun output-tasoon, kerätään kaikkien mahdollisten lopputulosten arvo. Se lopputulos, jolla on suurin arvo, vastaa neuroverkon opetteleman mallin mukaan todennäköisintä vaihtoehtoa. (Warwick, K., 2011, s.97–98)

Neuroverkkojen ideaa on kehitetty jo jonkin aikaa. Yksi vanhimmista neuroverkoista on Perceptron, jota testattiin käytännössä jo vuonna 1958. Perceptronissa on vain yksi neuroni ja vain yksi neuronitaso. (Graupe, D., 2013, s 17–18). Kun tietokoneet kehittyivät ja tehoa riitti useiden neuronien käsittelemiseen, pystyttiin testaamaan monitasoista Perceptronia eli Multi-layer-Perceptronia, jota oli ennen tutkittu vain teoriassa. Multi-layer-Perceptron koostui useista neuroneista ja mahdollisesti myös useista neuronitasoista. Moni nykyajan neuroverkko pohjautuu edelleen perinteiseen Perceptroniin. (Keller ym., 2016, s 9–10)

Nykyään on kehitetty useita erilaisia neuroverkkorakenteita, joita käytetään eri tehtäviin. Eri rakenteiden toiminta saattaa olla hyvinkin erilaista verrattuna toisiinsa. Tästä syystä eri neuroverkoissa on tiettyjä toiminnallisuuksia, jotka ovat samanlaisia tai hyvin samanlaisia.

2.3 Neuroverkkojen opettaminen

Neuroverkkoja opetetaan yleisesti parilla eri tavalla. Tässä luvussa tullaan käsittelemään tarkemmin kahta yleisintä tapaa opettaa neuroverkkoja ja kerrotaan myös lyhyesti miksi tekoälyä pitää opettaa.

Tekoälylle pitää opettaa työ, jota varten se luodaan. Oppiakseen tekoäly tarvitsee paljon tietoa ja sitä voidaan opettaa usealla eri tavalla. Tavat opettaa vaihtelevat paljon riippuen siitä, mihin tekoälyä hyödynnetään. Esimerkiksi jos halutaan tietokoneen osaavan ennustaa urheilusuorituksia, täytyy sille kerätä paljon informaatiota urheilusuorituksista.

Työssä käytetään valvotun oppimisen tapaa, koska se on yksinkertainen tapa opettaa neuroverkkoja ja demonstroida oppimista. Valvotun oppimisen käyttämiseksi, joudutaan määrittelemään etukäteen mitä tietoa halutaan oppia annetusta materiaalista ja mitä siitä tulevaisuudessa halutaan saada irti, jonka vuoksi tuloksista on helpompi tutkia, kuinka hyvin tekoäly on asian oppinut.

Neuroverkot, kuten muutkin tekoälyt, vaativat suuria määriä dataa. Dataa, jota käytetään opettamisessa, kutsutaan nimellä dataset. Eri opetustyylit vaativat erilaisia datasettejä, koska tavat käsitellä materiaaleja eroavat paljon. Se asettaa myös vaatimuksia sille, millaisessa muodossa tieto voidaan antaa.

2.3.1 Valvottu oppiminen

Valvotun oppimisen periaate on hyvin yksinkertainen. Otetaan esimerkiksi kuvilla oppiminen. Kaikki kuvat, joita käytetään valvotussa oppimisessa, tarvitsee luokitella etukäteen. Esimerkiksi jos halutaan, että neuroverkko pystyy kertomaan meille kuvassa olevan koiran rodun, tällaisen toimivuuden opettamiseksi ei riitä vain kuva koirasta, vaan tarvitsee myös kertoa, minkä rotuinen koira kuvassa on. Oppimisvaiheessa tietokone antaa arvauksen siitä, mitä kuvassa on, ja vertaa arvaustaan annettuun oikeaan vastaukseen. Vertailun jälkeen neuroverkossa olevien neuronien painoja muutetaan, jotta oltaisiin lähempänä oikeaa vastausta. Tietokone toimii näin jokaisen annetun oppimateriaalin kanssa ja muokkaa neuroneiden painoja sen mukaan, millainen arvaus oli suhteessa oikeaan vastaukseen. (Kananen ym., 2019, s 48–51)

Valvotun oppimisen yleisin käytettävä algoritmi, on back propagation -algoritmi. Back propagation kehitettiin jo vuonna 1986. Algoritmi mahdollisti hidden layerissä sijaitsevien neuronien painojen säätämisen. (Graupe, d., 2013, s. 59) Opettamista voidaan joko nopeuttaa tai hidastaa vaikuttamalla siihen, kuinka paljon painoja korjataan kerralla. (Lamm, ym., 2011, s. 159) Nopea oppiminen kuitenkin voi aiheuttaa neuroverkon epävakautta, sillä se vaikuttaa neuronien painoon siten, että painoarvon vaihtelu on suurempaa. (Keller ym., s 43) Jos neuroverkkoja opetetaan liikaa ja opettaminen lopetetaan vasta, kun vastaukset ovat aivan täydellisiä, syntyy riski, että

neuroverkko on niin sanotusti opetellut ulkoa tiedon, jota se opettelee. Tällöin neuroverkko ei välttämättä osaa enää vastata oikein, kun sille syötetään tietoa, joka ei ole ollut opetusmateriaalissa. (Keller ym., 2016 s. 50)

Opettamista varten materiaali jaetaan kahteen osaan yleensä niin, että noin 70—80 % käytetään opettamiseen ja loppu datasta käytetään oppimismallin tarkistamiseen. (Kattan ym., 2011, s 19)

2.3.2 Ohjaamaton oppiminen

Ohjaamaton oppiminen eroaa valvotusta oppimisesta siten, että ohjaamattomassa oppimisessa ei kerrota haluttua lopputulosta, jonka avulla neuroverkko voisi oppia ja säätää painojaan sen mukaan. Tiedot siis vain syötetään ilman minkäänlaista etukäteisorganisointia tai -luokittelua, mutta tiedon on sisällettävä yhteisiä tekijöitä, jotta tietoa voidaan luokitella. Tällä tavoin neuroverkolla pystytään etsimään yhteisiä tekijöitä ja organisoimaan tieto. Tämä auttaa automatisoimaan työtä, jossa tavoitteena on analysoida suurta määrää tietoa tai luokitella sitä. Tietokone voi siis löytää yhtenäisyyksiä tai poikkeamia, joita ihminen ei välttämättä huomaisi. Ohjaamatonta oppimista voidaan käyttää esimerkiksi, kun etsitään tietyn sairauden sairastaneiden potilaiden yhteisiä tekijöitä. (Kananen ym., 2019, s. 51—53)

2.4 Knime

Knime on avoimen lähdekoodin data-analytiikkasovellus (Bakos, g., 2013, s 24). Knime kuvailee sivuillaan (2020) sovelluksen olevan nopea ja helppo tapa lähestyä data-analytiikkaa. Knimen desktop-sovellus on saatavilla ilmaiseksi, mutta saatavilla oleva Knime server on maksullinen.

Knimessä tiedostoja käytetään Workspacessa. Workspacelle tallentuvat myös omat asetukset ja lokit. Workspacessa voi olla myös workflow-settejä. Workflow-setit pitävät sisällään nodeja tai muita Knimen työkaluja käyttämällä luotuja sovelluksia ja prosesseja. Tallaisia sovelluksia ja prosesseja käytetään analysoimaan ja visualisoimaan dataa. (Bakos, g., 2013, s 27)

Knimessä hyödynnetään paljon nodeja, joita on monia erilaisia eri prosesseja varten. Nodet ovat osia, joista workflow koostuu, ja Workflowssa jokainen node suorittaa yhtä osaa koko prosessista.

Nodet voivat vastaanottaa dataa ja antaa prosessoitua dataa eteenpäin. Jotkin nodet sisältävät node views -ominaisuuden, jolla voi esikatsella datan visualisointia. Jokaisella nodella on dokumentaatio, joka kertoo, mitä node voi tehdä. Knimessä on myös meta-nodeja, joiden sisällä on muita nodeja. Tämä auttaa organisoimaan workflow'ta. Workflow voi myös sisällyttää muuttujia, joita voidaan käyttää nodejen kanssa tai osana loop-rakennetta. Näitä muuttujia kutsutaan nimellä flow variables. (Bakos, g., 2013, s 28–32)

Knimessä nodeilla on portteja, joiden läpi tieto kulkee joko nodeen tai nodesta pois. Näistä yleisin on Data table -portti, joka on perinteisen taulukon tyyppinen. Data table koostuu siis riveistä ja sarakkeista. Kunkin sarakkeen tiedot voivat olla vain yhdenlaista esimerkiksi numeroita. (Bakos, g., 2013, s 30–32)

Knime sisältää paljon erilaisia nodeja ja muita osia. Knimeä voi myös laajentaa ulkopuolisten tuottajien luomilla paketeilla, joissa on lisää työkaluja workflow'n rakentamiseen. (Knime, 2021. -c) Nodeja ja muita mahdollisia osia yhdistelemällä pystytään luomaan juuri sellainen workflow kuin halutaan. Workflow voi valmiina näyttää monimutkaiselta ja mahdottomalta ymmärtää, kun useita nodeja on yhdistettynä toisiinsa viivoilla, eikä oikein tiedä, mistä toiminta alkaa tai mihin se loppuu. Tämä ei kuitenkaan tarkoita, että niiden luominen olisi yhtä haastavaan kuin miltä lopputulos näyttää.

Tutkimuksessa tullaan luomaan Knimessä workflow, joka sisältää toimivan neuroverkon, jolle voidaan antaa dataa ja joka tuottaa siitä haluttua tietoa. Tätä varten Knimeen täytyy asentaa lisäosa, joka sisältää tarvittavat nodet neuroverkon rakentamiseen. Lisäosaksi valikoitui Keras, koska siitä löytyi hyvin dokumentaatiota ja esimerkkejä. Keras on ohjelmointirajapinta, joka on rakennettu Tenserflow -ohjelmointirajapinnan päälle. Keras tarjoaa lähestyttävän tavan käyttää Tenserflow'n tarjoamia palveluja. Kerasta voi hyödyntää tutkimustarkoituksissa ja käyttää Keras-malleja verkossa ja mobiilissa.

2.5 ML5.js

Ml5.js-sivuilla (2020) kerrotaan kirjaston tavoitteena olevan luoda JavaScript-kirjasto, jonka avulla koneoppiminen olisi helposti lähestyttävämpää myös ihmisille, jotka eivät välttämättä ole koodanneet aikaisemmin. Ml5.js-kirjastoa ylläpitää ja kehittää New Yorkin yliopisto. Ml5.js-kirjasto on ottanut paljon vaikutteita p5.js- ja Processing-koodikielistä.

Ml5.js on JavaScript kirjasto, joka on rakennettu Tenserflow-tekoälyohjelmointirajapinnan päälle (ml5.js, 2021. -b). JavaScript-kirjasto on kokoelma erilaisia valmiita funktiota ja koodin pätkiä. JavaScript-kirjastojen koodeja voi käyttää koodissa aina tarpeen mukaan. Kirjasto on siis tietynlainen työkalulaatikko, josta voi ottaa työkalun, kun sitä tarvitsee. (Morris, 2021)

Ml5.js-kirjasto sisältää erilaisia koneoppimisen hyödyntämiseen tarkoitettuja valmiita osia, jotka on kirjoitushetkellä jaettu viiteen eri kategoriaan. Helpers-osio koostuu erilaisista paloista, joita voi hyödyttää datan kanssa työskentelyssä. Osio pitää myös sisällään osat neuroverkkojen luomiseen ja opettamiseen. Loput osiot pitävät sisällään nimensä mukaiset valmiit koneoppimisen osat. Image sisältää kuviin ja videoihin liittyvät osat, Sound ääneen liittyvät osat, Text tekstiin liittyvät osat ja Utils taas sisältää apuvälineet datan käsittelyyn. (ml5.js, 2021. -a)

2.6 Tenserflow

Tenserflow on avoimen lähdekoodin ohjelmointirajapinta, jonka Google julkaisi vuonna 2015. (Bernico, M., 2018, s. 15). Tenserflow tarjoaa käytettäviä kirjastoja useille eri ohjelmointikielille kuten Python, C++, Java, Go, JavaScript ja Swift. Moni muu koodikieli on myös tuettu käyttäjäyhteisön puolesta. (TenserFlow, (2021). -a) Tenserflow on käytettävissä myös Androidilla ja käytettävyyttä iPhone-laitteille kehitetään. TenserFlow’ta hyödyntävät useat eri yritykset, kuten Twitter, Coca-Cola, Intel, Paypal ja Lenovo. Näistä esimerkiksi Twitter käyttää TenserFlow’ta twiittien arvottamiseen. (TenserFlow, (2021). -b)

3 Kehittämistyön tavoite ja tarkoitus

Työn tavoitteena on luoda kaksi toimivaa sovellusta, jotka hyödyntävät neuroverkkoa. Sovellukset luodaan kahdella eri tavalla, mutta ne hyödyntävät samanlaista rakennetta. Molemmat neuroverkot tulevat olemaan Multi-layer-Perceptroneja tai toiselta nimeltään Feed-Forward Neural Networkeja (Kattan ym., 2011, s 3). Ensimmäisen sovelluksen kohdalla hyödynnetään Ml5.js-kirjastoa, jonka painotus on enemmän visuaalisella puolella, mutta joka tarjoaa hyvän mahdollisuuden myös muuhun neuroverkon hyödyntämiseen. Koska kyseessä on JavaScript-kirjasto, voi sitä helposti hyödyntää monessa eri paikassa. Toiseksi luodaan neuroverkko käyttämällä data-analytiikkaan luotua sovellusta Knimeä. Knime on painottunut voimakkaammin puhtaan datan analysoimiseen ja taulukointiin, mutta Knimellä voi silti tutkia esimerkiksi kuvia. Koska Knime ei ole Javascript-kirjasto kuten ml5.js, toimii sen neuroverkko pääsääntöisesti vain Knime-sovelluksen sisällä.

Tässä työssä neuroverkojen tavoitteena on ennustaa jääkiekkopelaajien tulevan kauden pisteet. Ennustettavana kautena toimii kausi 2018/2019, jotta vältettäisiin COVID-19-pandemian aiheuttamat vaikutukset lajin toimintaan. Jo pelatun kauden käyttäminen antaa myös mahdollisuuden tutkia, kuinka lähelle ennustukset osuivat.

4 Neuroverkkojen opettamiseen ja testaamiseen käytettävä materiaali

Neuroverkkojen opettamiseen, testaamiseen ja käyttämiseen käytettävänä materiaalina toimi SM-Liiga Oy:n (liiga.fi) nettisivuilta saatavilla olevat pelaajien tiedot. Pelaajatietojen käyttämiseen vaikutti kiinnostuksen lisäksi tiedon saatavuus, sillä pelaajatiedot ovat julkista tietoa. Myös ennustamisen käsite on helposti ymmärrettävissä. Nämä tukevat hyvin opinnäytetyön toiminnallisen osuuden ymmärrettävyyttä.

Opetusmateriaaliin keräsin 100 pelaajan tiedot. Pelaajien tiedoista otettiin materiaaliksi viisi kautta pelaajaa kohden. Jokaisesta kaudesta valittiin käytettäväksi kaudella pelatun joukkueen nimen, ottelumäärän, maalien määrän ja syöttöjen määrän. Jokaisen pelaajan tiedoista poistettiin nimi ja lisättiin pelipaikka. Jos pelaaja oli pelannut samalla kaudella useassa eri joukkueessa, joukkueen nimen tilalle laitettiin merkintä ”Mult”. Käytettävinä kausina toimivat kaudet 2014/2015–2018/2019, sillä haluttiin kaikkien tiedoissa käytettävien pelaajien kausien olevan samat. Tähän oli kaksi syytä. Ensimmäinen syy oli, että haluttiin ottaa pois kaudet joihin, korona on mahdollisesti vaikuttanut. Toinen syy oli vertailtavuus: kun kaikkien testattavien pelaajien ja opetukseen käytettävien pelaajien tietoihin käytettyjen kausien tiedot ovat samat, on vertaileminen helpompaa. Joiltain pelaajilta puuttui kausien tuloksia annetulta ajalta, koska he saattoivat pelata jossain toisessa liigassa. Pelaajien tietoja ei otettu mukaan opettamiseen, jos käytettävällä ajanjaksolla pelaajalla oli vain yksi käytettävä kausi. Kaudet, joista ei ollut saatavilla tietoa täytettiin nollilla, ja joukkueen tilalle kirjattiin merkintä ”Ei”. Tällä pyrittiin siihen, että merkityksettömillä riveillä olisi yhteinen tekijä, jotta neuroverkko mahdollisesti oppisi olemaan välittämättä näistä riveistä.

Pelaajatietojen opettamiseksi neuroverkolle annettiin joukkueen nimi sekä pelien, maalien ja syöttöjen määrä neljästä ensimmäisestä kaudesta (2014/2015–2017/2018). Tämän lisäksi viidennestä (2018/2019) eli ennustettavasta kaudesta annettiin opeteltavaksi joukkue, jossa pelaaja pelaa kyseisen kauden. Tiedot annettiin neuroverkoille taulukko muodossa (Liite 3).

5 Neuroverkon luominen käyttäen Ml5.js-kirjastoa

Opinnäytetyön tavoitteena oli luoda toimiva neuroverkko käyttäen ml5.js kirjastoa, jota voidaan opettaa ja tämän jälkeen käyttää haluttuun tehtävään. Ml5.js-kirjasto oli jo ennestään hieman tuttu ja vaikkei käyttökokemusta ollutkaan, tiedettiin sen olevan helposti lähestyttävä. Kirjaston käyttäminen ei vaatinut pohjustusta muuten kuin sen asentamisessa React.js-projektiin. Asentaminen oli hyvin yksinkertaista, ja siihen pystyi käyttämään NPM-pakettimanagerointijärjestelmää, jonka avulla on helppo asentaa ja lisätä kirjastoja projekteihin. (npm, 2021)

Aloitettiin siis luomalla React.js-projekti. React.js valittiin pohjaksi, koska se oli ennestään tuttu ja React.js-projektin ajaminen ja testaaminen selaimessa on helppoa, sillä sille ei tarvitse luoda paikallista hostia vaan se luo sen itse. Lisäksi koska Ml5.js-kirjasto on suhteellisen uusi, ei se toimi kaikkien muiden kirjastojen kanssa, mutta testaus React.js:n kanssa osoitti niiden toimivan yhdessä hyvin. Ainoa huono puoli oli, että ml5.js ei tunnistanut React.js:n komponentteja kuten esimerkiksi kuvakomponenttia, mutta tällaiset ongelmat olivat kierrettävissä helposti.

Kiitos ml5.js-kirjaston yksinkertaisuuden ja hyvän luokittelun, ei ennen kirjaston kanssa työskentelyä tarvinnut opiskella kirjaston toimintaa vaan se oli helppo oppia tehdessä. Ml5.js-verkkosivuilta löytyvät esimerkit ja funktioiden toiminnot olivat hyvänä apuna neuroverkkoa luodessa.

5.1 Sovelluksen rakentaminen

Aluksi Ml5.js-kirjasto piti saada käyttöön. Käyttöönotto tapahtui kuten muidenkin kirjastojen käyttöönottaminen Reactilla (Ohjelmakoodi 1). Tuomisen jälkeen käytössä oli koko Ml5.js-kirjasto ja neuroverkon rakentaminen oli mahdollista aloittaa.

Ohjelmakoodi 1 Kirjaston käyttöönotto.

```
import * as ml from 'ml5';
```

Neuroverkolle piti ensin määrittää asetukset, joita haluttiin käyttää. Piti luoda siis objekti, joka sisältää halutut asetukset. Määriteltäviä asetuksia olivat Inputs, Outputs, Task ja Debug. Input-asetuksen avulla pystyttiin määrittelemään input-tason neuronien määrän. Outputs-asetuksella puolestaan määritellään output-tason koko. Ml5.js-kirjasto mahdollistaa input- ja output-asetuksen määrittelyn nimeämällä taulukkoon kaikki siihen käytettävien tietojen sarakkeet, eli kertomalla, mitä tietoa annetaan. (Ohjelmakoodi 2) Inputin ja outputin voi myös määrittää antamalla vain luvun, joka vastaa inputin määrää. Työssä käytettiin jälkimmäistä tapaa eli numeroita, jotta koodi olisi selkeämpi ymmärtää. Neuroverkkoa rakentaessa käyttäen Ml5.js-kirjastoa kannattaa myös muistaa, että jos käyttää ensin mainittua tapaa, täytyy silloin syötettävän tiedon sarakkeiden vastata inputs-asetukseen laitettuja nimiä. Sama pätee outputs-asetuksen kohdalla.

Task-asetus kertoo neuroverkolle, mitä tehtävää verkon tulee suorittaa. ML5.js-kirjastolla voi valita tehtäväksi joko regression, classificationin tai imageClassificationin. ImageClassification on nimenomaan kuvien luokittelua vastaava tehtävä. (Ml5, 2021, -a) Classification-asetuksella neuroverkko pyrkii luokittelemaan tiedon outputs-kohdassa annettuihin luokkiin. Regression avulla annetuista tiedoista voidaan oppia ennustamaan haluttua tietoa. (Brenico, M., (2018) s. 27) Koska neuroverkkojen tarkoituksena oli ennustaa pelaajien tuleva kausi, task-asetus määriteltiin regressioksi. (Ohjelmakoodi 2)

Debug-asetuksen päälle laittaminen antaa mahdollisuuden käyttää Ml5.js'n sisäänrakennettua debug-näkymää, joka on erityisen hyödyllinen, kun tarkastellaan neuroverkon toimintaa (kuva 2). Siksi debug määriteltiin todeksi (Ohjelmakoodi 2).

Ohjelmakoodi 2 Neuroverkon asetusten luominen.

```
const Asetukset = {
  inputs:18,
  outputs:3,
  task: 'regression',
  debug: true
};
```

Ml5.js antaa muutaman eri mahdollisuuden syöttää opetettavaa tietoa neuroverkolle. Opetettavan materiaalin voi antaa suoraan yhtenä tiedostona, joko määrittelemällä tiedoston

osoitteen suoraan neuroverkon asetuksiin (Ohjelmakoodi 2) tai käyttämällä loadData-funktiota. Kolmas mahdollisuus on käyttää Ml5.js-kirjaston mukana tulevaa addData-funktiota. (Ohjelmakoodi 3) Funktioon annetaan opetettavat tiedot rivi kerrallaan. Lisäksi pitää erotella input- ja output-tasoja vastaavat sarakkeen erikseen. Funktion ensimmäinen muuttuja vastaa input-tasolle annettavia tietoja ja toinen muuttuja vastaa output-tasolle annettavia tietoja. Työssä käytettiin tietojen syöttämiseen addData-funktiota, sillä tietojen määrittely sovellukselle neuroverkkoasetusten kautta tai loadData-funktiota käyttäen ei toiminut. Vian syytä ei saatu selville, mutta vian saattaa aiheuttaa yhteensopivuusongelmat react.js kanssa.

Kun tieto oli syötetty, se piti normalisoida normalizeData-funktiolla (ohjelmakoodi 3). Tiedon normalisoinnilla tarkoitetaan tiedon muuttamista esimerkiksi siten, että arvot vastaavat alkuperäisiä arvojaan, mutta ovat jotain 0 ja 1 välillä. Tämä auttaa nopeuttamaan oppimista. (Brenico, M., (2018) s. 100)

Ohjelmakoodi 3 Neuroverkkon luominen ja tiedon lataaminen neuroverkkoon.

```
const Neuroverkko = ml.neuralNetwork(Asetukset)

function loadplayers(){

  for (let index = 0; index < playerInput.length; index++) {
     Neuroverkko.addData(playerInput[index],playerOutput[index]);
  }

  Neuroverkko.normalizeData();
  Opettelu();
}
```

Kun tiedot oli syötetty ja normalisoitu, voitiin neuroverkkoa ruveta opettamaan. Oppimista varten piti luoda objekti, jossa oppimisen asetukset sijaitsivat, kuten aikaisemmin oli tehty neuroverkon asetusten kanssa. (Ohjelmakoodi 4) Asetuksiin määriteltiin vain epocs-asetus, joka yksinkertaisuudessaan kertoo vain, kuinka monta kertaa opetusmateriaali syötetään neuroverkolle opittavaksi. Neuroverkolle voisi myös määritellä batchSize-asetuksella, kuinka monen rivin perusteella neuroverkon painoja säädetään. (Brenico, M., (2018) s. 34)

Ohjelmakoodi 4 Oppimisen asetusten luonti ja oppimisen käynnistäminen.

```
function Opettelu(){
   const opetteluAsetukset = {
     epochs: 100,
     batchSize: 100
   };
```

Neuroverkon asetukset olivat valmiit, ja neuroverkolle voitiin antaa komento aloittaa oppiminen. Ml5.js sisältää funktion nimeltä train, joka antaa neuroverkolle oppimiseen määritetyt asetukset ja jonka avulla voidaan suorittaa funktioita kesken oppimisen ja sen jälkeen. Train-funktion ensimmäiseksi muuttujaksi annettiin oppimisen asetukset. Toiseksi muuttujaksi määriteltiin funktio, joka halutaan suorittaa jokaisen epochin jälkeen. Train-funktio lähettää toiseen muuttujaansa kaksi tietoa: epocin, jonka se on suorittanut; ja loss-funktion tuloksen. Viimeiseksi train-funktion muuttujaksi annettiin opettamisen jälkeen suoritettava funktio.

Ohjelmakoodi 5 Oppimisen kulku.

```
  function Opettelu(){
    const opetteluAsetukset = {
      epochs: 100,
      batchSize: 100
    };
    Neuroverkko.train(opetteluAsetukset,KeskenHarjoittelun,ValmisHarjoittel
u);
  }

  function KeskenHarjoittelun(epoch, loss){
    console.log(epoch);
  }

  function ValmisHarjoittelu(){
    console.log("valmista");
    Ennusta();
    console.log(Neuroverkko);
  }
```

Train-funktion viimeiseksi muuttujaksi annettiin funktio, joka kirjoittaa konsoliin, kun oppimisprosessi on ohitse. (Ohjelmakoodi 5) Tämän jälkeen haluttiin suorittaa Ennusta-funktio (Ohjelmakoodi 6). Ennusta-funktion sisällä suoritetaan Ml5.js-kirjastoon kuuluva predict-funktio, joka antaa sille ensimmäiseksi muuttujaksi määritellyn sisällön neuroverkolle, josta neuroverkko sitten ennustaa oppimallaan tavalla halutut tiedot. Predict-funktio lähettää ennustuksesta saadun tuloksen sille toiseksi määriteltynä olevaan muuttujaan. Tässä tapauksessa muuttujaksi oli määritelty Tulokset-funktio, jonka avulla neuroverkosta saadut tulokset kirjattiin konsoliin. Työn

tässä vaiheessa, tuloksen testaamiseen käytettiin ensimmäistä riviä opetukseen annetuista tiedoista. Näin pystyttiin hyvin tarkastelemaan, kuinka lähelle ennustus osuu. Tällä tavalla myös nähtiin, oliko neuroverkkoa opetettu liikaa.

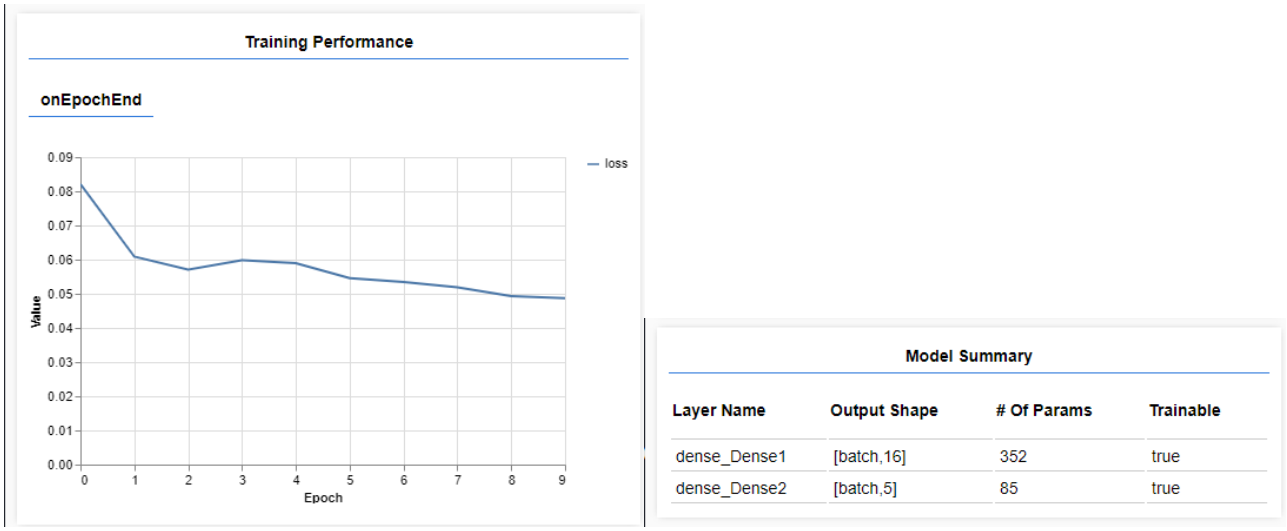
Ohjelmakoodi 6 Neuroverkon testaaminen ja käyttäminen.

```
function Ennusta(){
   Neuroverkko.predict(playerInput[0],Tulokset);
}

function Tulokset(error, result) {
   if(error){
     console.error(error);
     return;
   }
   console.log(result);
}
```

Oppimista suorittaessa Ml5.js tarjoaa mahdollisuuden avata ikkunan, josta voi hyvin seurata oppimista. (kuva 2) Ikkunan saa näkyviin, kun määrittelee neuroverkon asetuksissa debug-asetuksen todeksi. (Ohjelmakoodi 1) Graafi näyttää loss-funktion tuloksen. Yksinkertaistettuna loss-funktiolla lasketaan neuroverkon tuloksen virhe eli tuloksen ero oikeaan vastaukseen. (Brenico, M., (2018) s. 31). Mitä pienempi loss-arvo siis on, sitä lähempänä oikeaa vastausta ollaan. Debug-ikkunasta nähdään myös neuroverkon rakenne. (Kuva 2)

Kuva 2 Ml5.js debug-ikkunassa oleva taulukko, jossa näkyy loss-määrä ja lista neuronitasoista.



5.2 Sovelluksen käyttäminen

Sovelluksen käyttämistä varten rakennettiin yksinkertaiset verkkosivut. Verkkosivujen kautta neuroverkolle annettiin opetettava ja ennustettava materiaali. Verkkosivulle luotiin mahdollisuus antaa tiedostot, joita käytettään oppimisiin ja tiedosto, joka sisältää ennustukseen käytettävän tiedon.

6 Neuroverkon luominen käyttäen Knime-analytiikkasovellusta

Työssä käytetään Knimen ilmaista työpöytäsovellusta. Knime oli aluksi kokemattomalle Knimen käyttäjälle monimutkaiselta vaikuttava sovellus, jonka kuitenkin oppi hyvin nopeasti. Oppimista nopeutti se, kuinka paljon Knimessä työskentely muistuttaa koodaamista, vaikka itse koodia ei tarvitsekaan kirjoittaa.

Tässä osassa työtä tavoitteena oli luoda neuroverkko, jota voidaan opettaa ja käyttää Knimen kautta. Haluttiin myös tutkia mitä erilaisia mahdollisuuksia olisi jatkokehittää neuroverkon toimintaa, ja mihin sitä voitaisiin tulevaisuudessa hyödyntää.

Rakentaminen oli aluksi vaikea hahmottaa, koska vaikka oli saatu Ml5.js avulla luotu neuroverkko jo valmiiksi ja toimivaksi, oli Knimen kanssa toimiminen hyvin erilaista. Tavoitteeseen pääseminen kuitenkin helpottui, kun tarkasteltiin ml5.js-neuroverkon asetuksia. Aikaisempien tietojen, asetusten, lähteiden ja Knimessä olevien nodejen avulla saatiin luotua toimiva neuroverkko Knimeen.

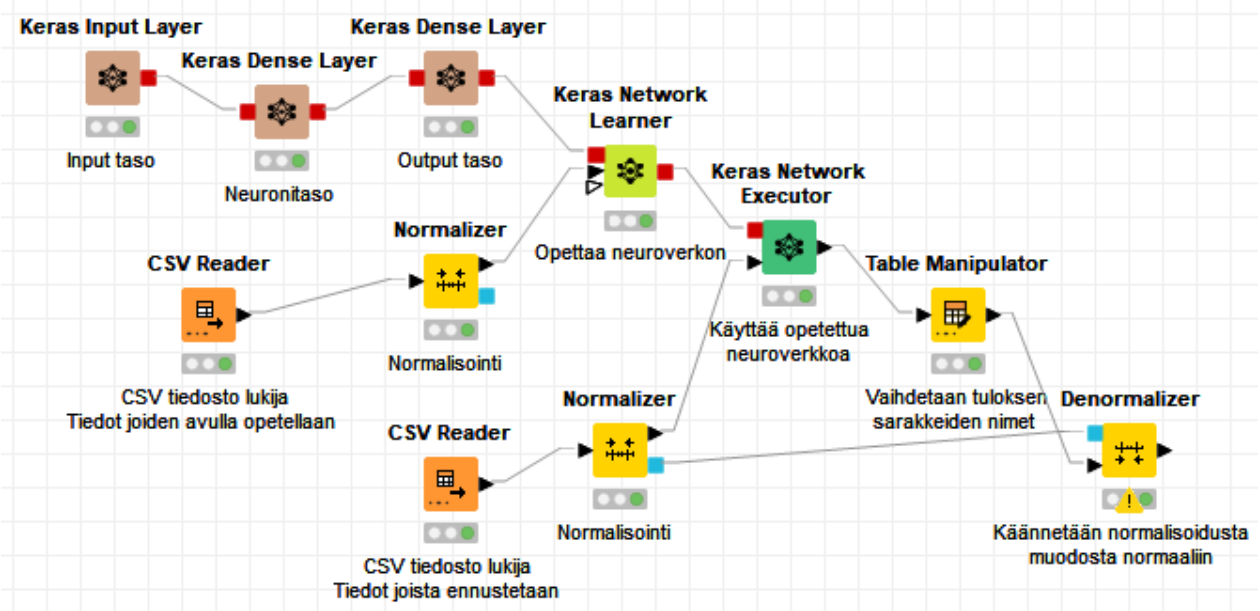
6.1 Neuroverkon rakentaminen

Neuroverkon luominen aloitettiin avaamalla Knimeen uusi workflow, jonka sisälle neuroverkkoa alettiin luomaan. Knime toimii nodejen avulla, joten aloitettiin selaamalla Knimen neuroverkkoihin liittyviä nodeja node-listasta. Kaikkia tarvittavia nodeja ei kuitenkaan löytynyt Knimestä valmiina. Knimeen täytyi siis ladata lisäosa, joka sisältää neuroverkkojen luomiseen tarkoitetun Keras-node-kirjaston. Kerasin asentamisen jälkeen saatiin käyttöön iso kirjasto, jonka avulla pystyttiin luomaan neuroverko. Ennen rakentamista yritettiin löytää tietoa siitä, mitä nodeja tullaan tarvitsemaan ja miksi. Knimen sivuilta löytyi jonkinlainen esimerkki Kerasilla rakennetun neuroverkon rakenteesta, mutta pelkän kuvan perusteella oli vaikea sanoa, mikä oli minkäkin noden merkitys. Siitä saatiin kuitenkin pieni käsitys siitä, mitkä voisivat olla keskeisimmät nodet.

Tämän jälkeen alettiin testailemaan erilaisia node-yhdistelmiä. Testailu tuotti tulosta, ja saatiin aikaiseksi muutama jollakin tavalla toimivan neuroverkko. Tavoitteena oli kuitenkin luoda selkeä

rakenteinen neuroverkko, jotta olisi helpommin ymmärrettävissä, mitä mikäkin node oikeasti tekee. Tähän käytettiin avuksi jo aiemmin Ml5.js:n avulla luotua neuroverkkoa. Tutkittiin, minkälaisia asetuksia ml5.js loi automaattisesti neuroverkon taustalle, ja selvitettiin tiettyjen asetusten merkityksiä käyttämällä hyödyksi lähteiksi etsittyjä kirjoja. Vertailtavuuden ja selkeyden vuoksi, oli myös tärkeää luoda lähestulkoon vastaavanlainen rakenne Knimessä, kuin millaisen Ml5.js loi automaattisesti.

Kuva 3 Knimellä luotu neuroverkko.



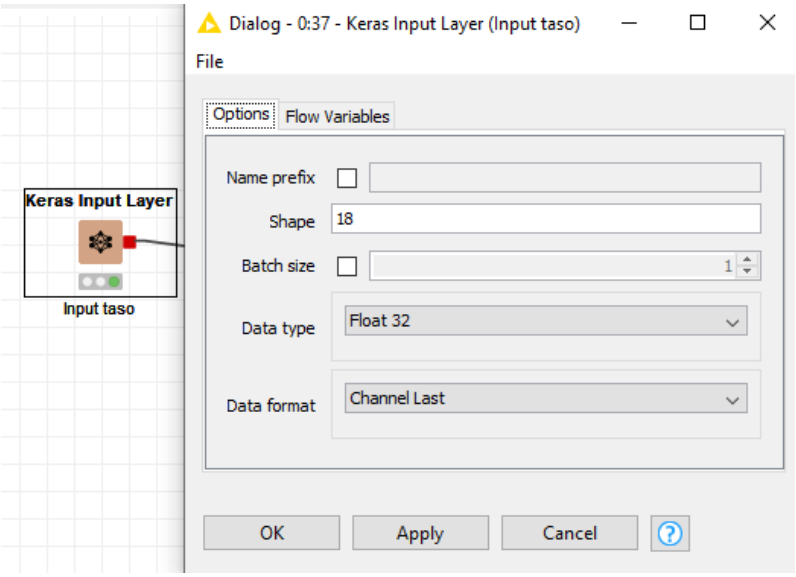
Testailun ja lisätutkimisen jälkeen saatiin yksinkertainen ja toimiva neuroverkko. Knimellä luotu workflow (kuva 3) saattaa vaikuttaa ensisilmäyksellä haastavalta ymmärtää. Workflow koostuu kuitenkin samoista osista kuin Ml5.js-neuroverkko ja samoista toiminnoista, joita on käsitelty jo teoriaosuudessa. Workflow'ta lukiessa kannattaa muistaa, että nodejen välinen tieto kulkee vasemmalta oikealle, eli vasemmalle puolelle nodea liittyvät viivat tuovat tietoa ja oikealle puolelle yhdistyvät viivat vievät tietoa. Toiminnallisuuden ymmärtämisen vuoksi ei tulla käyttämään Keras-etuliitettä, kun avaan toimintoja tarkemmin, ellei siitä erikseen ole hyötyä. Toiminnallisuus avataan kolmessa eri osassa.

6.1.1 Neuroverkon rakenne

Kuva 4 Neuroverkon tasoja vastaavat nodet

Neuroverkon rakenne koostuu kolmesta eri nodesta, input layerista ja kahdesta Dense layer -nodesta (Kuva 4), joista jokainen vastaa yhtä tasoa neuroverkossa. Input layer -taso nimensä mukaisesti on input-taso. Neuroni-tasona toimii vasemmalta päin katsottuna ensimmäinen Dense layer. Output-tasoa vastaa viimeinen Dense Layer. Vaikka rakenteessa on kaksi samanlaista nodea, ovat niiden tarkoitus ja asetukset erilaisia.

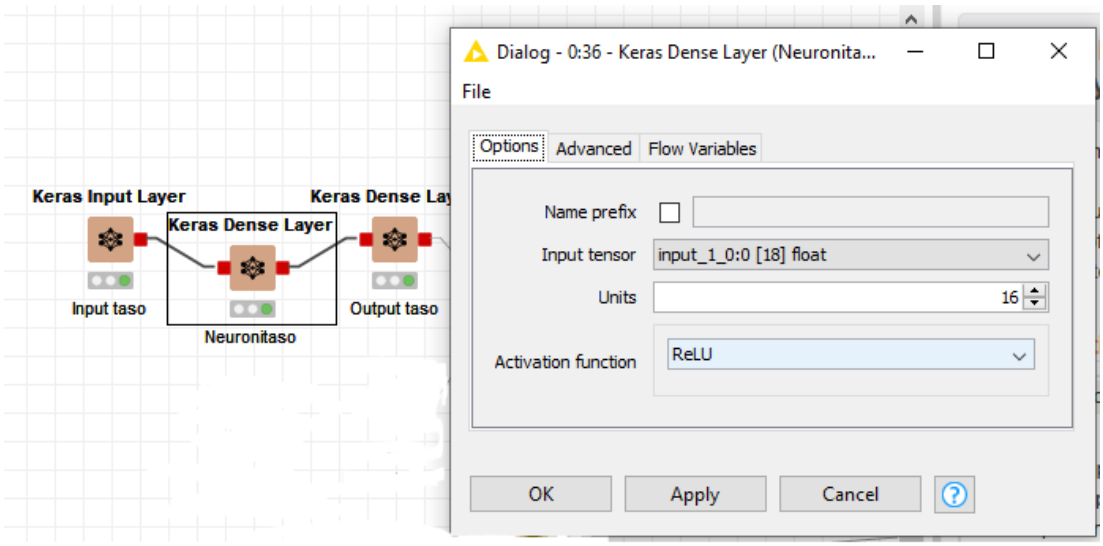
Kuva 5 Keras input layer node ja asetukset



Input Layer -node (kuva 5) vastaa rakenteessa siitä, kuinka monta neuronia on input-tasolla, ja näin ollen siitä, kuinka monta palasta tietoa neuroverkko vaatii. Noden asetukset sisältävät useita kohtia, joista työn kannalta tärkeimmät ovat Shape- ja Data type -asetukset. Shape-kohdassa määritetään muoto, eli kuinka monta neuronia input-tasolla tarvitaan. Tämä määräytyy sen

mukaan, kuinka monta eri muuttujaa neuroverkolle halutaan syöttää. Data Type -asetuksessa määritetään saatavan datan muoto. Datan muoto voi olla joko Boolean, Float tai Integer. Työssä luodun neuroverkon workflow'ssa Shape oli 18 ja Data type oli Float.

Kuva 6 Neuroni- ja output-tasojen asetukset-ikkuna



Vasemmalta katsottuna ensimmäinen Dense layer vastaa neuronitasoa. (Kuva 4) Keras-lisäosan kanssa tulee paljon nodeja, jotka vastaavat erilaisia tasotyylejä. Dense layer -node vastaa tasoa, jossa kaikki neuronit yhdistyvät seuraavan tason kaikkiin neuroneihin (kuva 1). Dense layer -nodejen asetuksissa on paljon mahdollisuuksia säätää tason toimintaa, mutta tärkeimmät ovat units- ja activation-function-asetukset. Units-asetus määrittelee tason neuronien määrän. Units-arvoksi määriteltiin 16 vastaamaan samaan kuin minkä Ml5.js loi automaattisesti. Activation function määrittelee, mitä laskukaavaa neuroni käyttää signaalin arvon laskemiseen. Activation functioniksi valittiin ReLU-funktio. (Kuva 6) ReLU-funktio valikoitui, koska haluttiin neuroverkkojen vastaavan toisiaan.

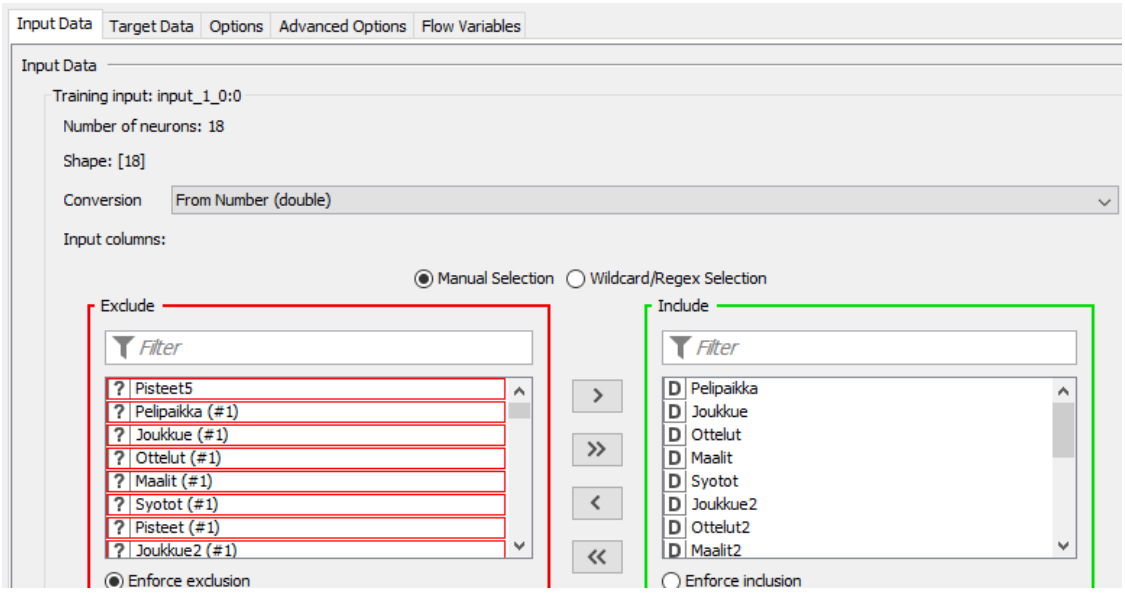
Viimeinen node, joka muodostaa neuroverkon, on kaikista oikeanpuoleisin node workflow'ssa. (Kuva 5) Tämä node vastaa neuroverkon output-tasoa. Koska kyseessä on sama node kuin neuronitasossa, ovat käytettävät asetuksetkin samat. Output-tason koko eli noden units -luku määrittyy sen mukaan, kuinka monta vastausta neuroverkosta halutaan. Units-arvoksi määriteltiin

3. Activation funtioniksi valikoitui sigmoid funktio, koska sen avulla saadaan suora vertailukohde aiemmin luotuun neuroverkkoon. (Kuva 6)

6.1.2 Oppiminen

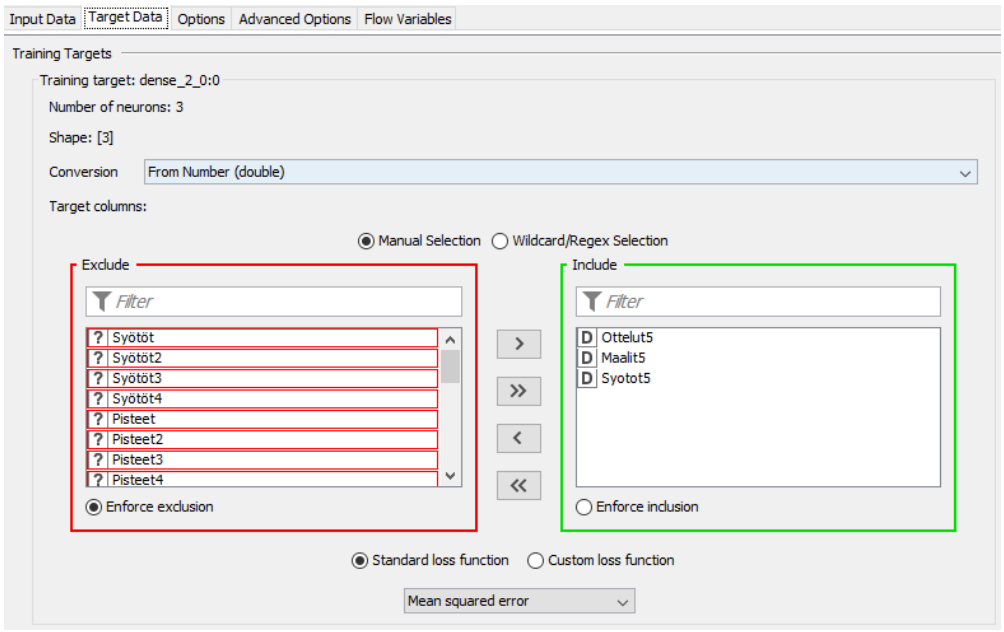
Neuroverkon oppimiseen Knimessä tarvitsee käyttää vain yhtä nodea, jonka sisällä oppiminen tapahtuu ja, joka sisältää oppimiseen tarvittavat asetukset. Noden nimi on Network Learner (kuva 4). Luettavuuden vuoksi tullaan viittaamaan Network Learner -nodeen nimellä **NL-node**. Oppimisen tapahtuessa yhdessä nodessa se tarkoitti, että määriteltäviä asetuksia oli useampi. Onneksi Knimessä nodejen määritteleminen on selkeää. Keras-lisäosassa tulleissa nodeissa on hyvä dokumentaatio siitä, miten mikäkin asetus vaikuttaa oppimiseen tai noden toimintaan. Jotta NL-nodea pystyi käyttämään, piti siihen ensin yhdistää tarvittavat nodet, joiden avulla saatiin data, jolla neuroverkkoa opetetaan, ja itse neuroverkon rakenne. Rakenne NL-nodeen saatiin, kun NL-nodeen yhdistetään rakenneosuuden viimeinen node, eli output-tasoa vastaava Dense layer -node. NL-nodeen voi myös yhdistää validation-datan. Tätä ei kuitenkaan koettu tarpeelliseksi käytettävän tiedon vähäisen määrän ja vertailtavuuden vuoksi. Validation-datan avulla pystyisi esimerkiksi lopettamaan oppimisen, kun validiointiin käytettävistä tiedoista saatu ennustustulos ei enää parane.

Kuva 7 Network learner -noden Input Data -asetukset



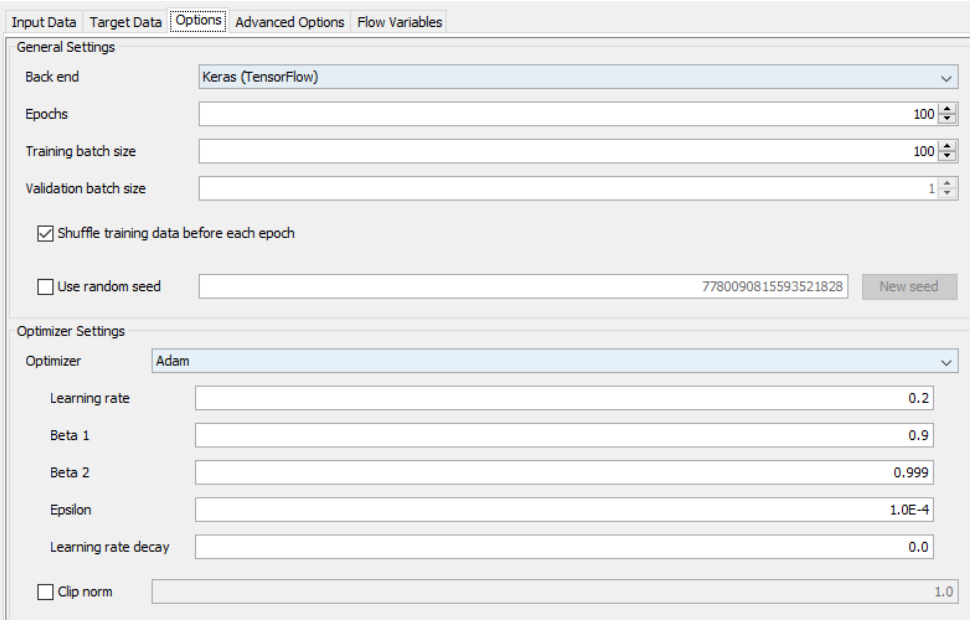
Yhdistämisen jälkeen oli aika siirtyä NL-noden asetusten määrittelyyn. NL-noden asetuksiin piti ensin määritellä, mitä dataa käytetään opettamiseen. NL-node tunnisti yhdistetyn datan sarakkeet, joista valittiin käytettävät sarakkeet. (kuva 8)

Kuva 8 Network learner -noden Target(output) Data -asetukset



Tämän jälkeen sama prossessi toistui, mutta tällä kertaa määriteltiin, mitkä sarakkeet vastaavat opeteltavaa lopputulosta. Samassa välilehdessä myös määriteltiin loss-funktio. Loss-funktioksi valikoitui Mean squared error -funktio jälleen kerran, jotta vertaillessa toiseen neuroverkkoon muuttujia olisi mahdollisimman vähän. (Kuva 9)

Kuva 9 Network learner -noden options-näkymä



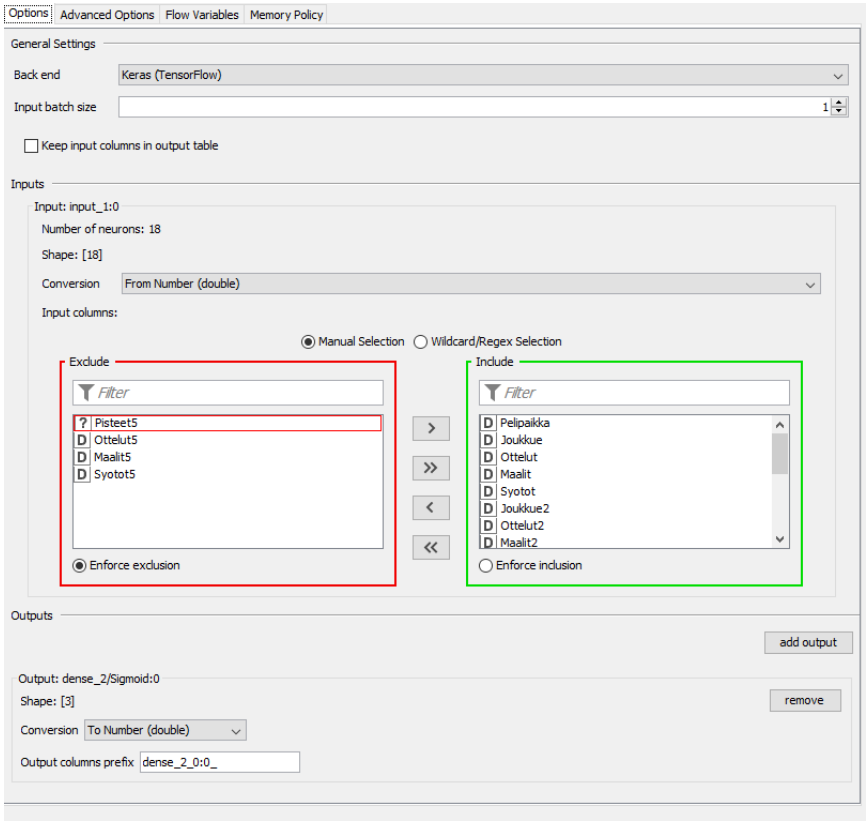
Viimeiset määriteltävät asetukset löytyivät Options-välilehdeltä (Kuva 10), jossa säädetään oppimiseen liittyvät asetukset. Oppimisen asetuksista ensimmäiset olivat jo aiemmasta neuroverkostakin tutut Epochs ja Batch size. Epochs-määräksi määriteltiin 100, ja Batch sizeksi valittiin 100, kuten aikaisemmankin neuroverkon kanssa. Tämän jälkeen määrittelin sivulta löytyvät optimisointi-asetukset. Pyrittiin saamaan optimisointi-asetukset vastaamaan aiemmin Ml5.js:llä luodun neuroverkon asetuksia. Optimizeriksi valikoitoitu Adam, jolle määritteltiin seuraavat asetukset: learning rate 0.2, beta1 0.9, beta2 0.999, Epsilon 0.0001 ja Learning rate decay 0.0. Adam ja sen asetukset määriteltiin vastaamaan Ml5.js neuroverkon asetuksia.

6.2 Sovelluksen käyttäminen

Viimeinen keskeinen node, jota neuroverkko tarvitsi, oli Network executor (Kuva 3), joka käyttää opetettua neuroverkkomallia. Network executor -node tarvitsee kaksi syötettä: neuroverkkomallin ja tiedon, jota se syöttää neuroverkkoon. Node tuottaa prosessointinsa jälkeen neuroverkosta saadun vastauksen, jonka se lähettää eteenpäin. Node sisältää useita asetuksia, joista tärkeimmät ovat Inputs- ja Outputs-asetukset. Inputs-kohdassa määritettiin, mihin saadun tiedon sarakkeita vastaavat tiedot valitaan samalla tavalla kuin aikaisemmin Network learner -noden kohdalla.

Outputs-asetuksessa täytyy valita, mikä neuroverkon rakenteeseen kuuluvista nodeista on Output-tasoa vastaava. (Kuva 11)

Kuva 10 Network executor -noden asetusikkuna



Kaikki tämä tarvitsi kuitenkin tietoa, jolla voidaan opettaa tai jonka pohjalta ennustus tehtiin. Ensin tieto piti tuoda Knimeen. Tähän käytettiin nodea nimeltä CSV-reader (kuva 3), joka nimensä mukaan lukee csv-tiedoston ja lähettää sen pohjalta tehdyn taulukon eteenpäin. CSV-readerin asetuksissa riittää, että valitsee käytettävän tiedoston sijainnin. Taulukko pitää kuitenkin normalisoida, ja tähän Knimestä löytyy node nimeltä Normalizer. Normalizerin asetuksissa voi määritellä, mille arvovälille tieto normalisoidaan. Koska haluttiin tiedon olevan 0 ja 1 välillä, ei asetuksia tarvinnut määritellä, sillä se arvoväli on Normalizer-noden perusasetuksena. Normalizer-node lähettää kahta tietoa: taulukon, jonka arvot on muutettu, ja mallin, jolla se on muutettu. Normalisoitua taulukkoa käytetään Network learner- ja Network executor -nodessa. Kun on tehty ennustus Network executor -nodella, täytyy ennustustaulukon sarakkeille antaa oikeat nimet. Tämä onnistuu käyttämällä Table manipulator -nodea. Table manipulator -nodesta saadaan siis taulukko, jonka sarakkeet vastaavat normalisointiin annetun taulukon sarakkeita, mutta tiedot

ovat silti 0 ja 1 välillä. Tiedot saadaan muutettua takaisin käyttämällä Denormalizer-nodea, joka nimensä mukaisesti muuttaa normalisoinnin takaisin. Denormalizer -nodelle tarvitsi antaa kaksi tietoa: table manipulatorista saatu taulukko, jossa on ennustus ja oikeat sarakkeen nimet. Toinen syötettävä tieto piti olla normalisointiprosessista saatu muutosmalli, jonka avulla Denormalizer muuttaa ennustustaulukon tiedot vastaamaan oikeaa muotoa.

Tämä rakennettu neuroverkko toimii siis Knime-sovelluksessa. Maksullisella Knime-serverillä on mahdollista ajaa Knime-workflow'ta. Workflow'ta voi käyttää myös Pythonin avulla (Knime, (2021, d). Knime itsessään on hyvä analytiikkatyökalu, joten jos neuroverkon tarkoituksena on toimia osana tutkimuksia, toimii pelkkä Knime-sovellus todella hyvin. Pythonin kautta tämän käyttäminen on hyvä mahdollisuus, jos Workflow'ta halutaan käyttää muualla kuin missä workflow-tiedosto sijaitsee. Knime-sovelluksessa nodet voi suorittaa muutamalla tavalla: Klikkaamalla hiiren oikealla napilla ja valitsemalla Execute, tai Knimen yläpalkista voi myös klikata haluaako suorittaa koko workflow'n kerralla vaiko vain valitun noden.

7 Neuroverkkojen tulokset

Opinnäytetyön tavoitteena oli vertailla kahden erilaisen työkalun käyttöä neuroverkkojen luomisessa. Vertailua varten luotiin molemmilla työkaluilla neuroverkko. Neuroverkoista luotiin samanlaiset, jotta työkalujen vertailu olisi helpompaa. Luotujen neuroverkkojen testaaminen antoi mahdollisuuden verrata myös neuroverkkojen työn tulosta, eli tässä tapauksessa neuroverkoista saatuja ennusteita. Tässä kappaleessa avataan ensin käytettyjen työkalujen eroja ja vertaillaan niitä. Tämän jälkeen tarkastellaan tarkemmin, millaisia tuloksia neuroverkoista saatiin.

7.1 Järjestelmien erot ja yhtäläisyydet

Knime ja Ml5.js ovat hyvin erilaisia järjestelmiä, mutta samanlaisuuksiakin löytyy. Suurimman eron huomaa heti, kun työkaluja alkaa käyttämään, sillä Knime on oma sovelluksensa, jonka sisällä neuroverkko luodaan, kun taas Ml5.js vaatii ympärilleen järjestelmän, jossa neuroverkko luodaan ja jossa sitä käytetään. Luotujen neuroverkkojen käyttämisessä huomataan myös suuria eroja. Knimellä luodun neuroverkon käyttäminen on rajallista, sillä se on oma sovelluksensa. Knime tarjoaa vain vähän mahdollisuuksia ulkopuolisille komennoille eikä sitä voida upottaa toisiin sovelluksiin. Ml5.js sen sijaan antaa mahdollisuuden lisätä neuroverkon lähestulkoon mihin vain järjestelmään, joka käyttää JavaScriptiä. Ml5.js:llä on toki vielä jotain yhteensopivuusongelmia muiden kirjastojen kanssa. Esimerkiksi testaillessa Ml5.js:n toimivuutta opinnäytetyön alkupuolella, aiheuttivat React.js kirjaston omat komponentit ongelmia. Ongelmat ratkesivat, kun käyttöön otettiin React.js komponenttien tilalle perinteiset HTML-tagit.

Toinen käyttämisen eroista ilmeni opettavan ja ennustettavan tiedon käytössä. Molemmat neuroverkot tarvitsivat tietyn malliset datatiedostot, mutta isoin ero oli tietojen muokattavuus neuroverkolle rakennetussa järjestelmässä. Knimessä pystyi paljon hyödyntämään sen data-analytiikkapuolta. Data-analytiikkanodeja käyttäen annettava tieto oli helposti muokattavissa neuroverkkoa käyttäessä. Ml5.js kanssa on toki mahdollista käyttää JavaScript-kieleen kuuluvia funktioita tai muita ulkopuolisia kirjastoja tietojen muokkaamiseen. Knime kuitenkin on tässä paljon sujuvampi ja, koska kyse on taulukoista, Knimen visuaalisuus on suuri etu. Tämä antaa mahdollisuuden tutkia, kuinka taulukko muuttuu vaiheiden välillä.

Vaikka järjestelmissä ilmenee paljon eroja, on niissä myös jotain samaa. Järjestelmien tärkein yhtäläisyys on niiden helppokäyttöisyys. Knimessä käytetty lisäosa Keras kuin myös ml5.js, ovat samalla tavalla luotu helpottamaan Tenserflow'n käyttöä. Molemmat kirjastot olivat ymmärrettävissä lähestulkoon yhtä helposti. Näistä Keras kuitenkin vaati hieman suurempaa tietämystä neuroverkkojen toiminnasta, mutta käytettävyydeltään järjestelmät olivat tasavertaisia. Toinen merkittävä mahdollisuus, joka löytyi molemmista kirjastoista, oli luotujen neuroverkkojen mallien tallentamien ja mahdollisuus käyttää valmiita malleja.

7.2 Ennustustulokset

Neuroverkkojen tuloksia on avattu taulukoihin. Taulukoissa maalit- ja syötöt-sarakkeista nähdään oikeiden vastausten ja neuroverkosta saatujen tulosten ero. **Yht**-sarakkeessa ovat yhteenlaskettuna **maalit-** ja **syötöt**-sarakkeiden arvot. **Yht**-sarakkeen avulla saa nopeasti kuvan siitä, kuinka lähelle ennuste osui. **Pelipaikka-**sarake kertoo, millä pelipaikalla pelaaja pelaa. **Kaikki kaudet saatavilla** -sarakkeesta voidaan katsoa, oliko ennustettavalta pelaajalta saatavilla kaikki viisi kautta ennustamiseen.

Jokaista tulosta varten neuroverkko opetettiin kymmenen kertaa, ja jokaisen opettamisen jälkeen tehtiin ennustus. Tämän jälkeen ennustukset yhdistetiin yhdeksi ennustukseksi laskemalla ennustusten keskiarvo. Tällä tavoin pystyttiin välttämään todennäköisyys sille, että toisesta järjestelmästä saatu tulos olisi paras mahdollinen ja toisesta saatava huonoin mahdollinen.

Taulukko 1 Ml5-neuroverkon tuloksien ero oikeisiin vastauksiin.

ML5 Tulokset	Kauden 2018/2019 Ennustusteen ero oikeisiin vastauksiin				
ID	Maalit	Syötöt	Yht	Pelipaikka	Kaikki Kaudet saatavilla
1	0,34 yli	4,04 yli	4,38	Hyökkääjä	Kyllä
2	2,69 alle	9,94 yli	12,63	Hyökkääjä	Kyllä
3	3,14 yli	5,53 yli	8,67	Puolustaja	Kyllä
4	6,12 yli	4,78 yli	10,90	Puolustaja	Ei
5	5,31 alle	13,77 alle	19,09	Hyökkääjä	Ei
6	4,38 alle	13,55 alle	17,93	Hyökkääjä	Ei
7	2,83 yli	12,9 alle	15,72	Hyökkääjä	Ei
8	2,25 alle	15,39 alle	17,64	Hyökkääjä	Ei
9	10,63 alle	2,71 alle	13,34	Hyökkääjä	Ei
10	1,8 yli	13,36 yli	15,15	Puolustaja	Kyllä

Taulukosta 1 nähdään, että Ml5.js:llä luodun neuroverkon ennusteen lähin arvaus oli 4,38 pisteen päässä ja huonoin arvaus erosi oikeasta vastauksesta 19,09 pistettä. Maalien tarkkuus oli keskiarvoltaan 71,80 %, ja syöttöjen puolestaan 64,40 %. Kaikkien ennustettavien pelaajien kokonaistarkkuus eli maali- ja syöttötarkkuuden keskiavo oli 68,10 %.

Taulukko 2 Knime-tulosten ero oikeisiin vastauksiin.

Knime Tulokset	Kauden 2018/2019 Ennustusteen ero oikeisiin vastauksiin				
ID	Maalit	Syötöt	Yht	Pelipaikka	Kaikki kaudet saatavilla
1	0,41 yli	3,56 yli	3,97	Hyökkääjä	Kyllä
2	2,27 alle	7,28 yli	9,55	Hyökkääjä	Kyllä
3	4,6 yli	4,9 yli	9,50	Puolustaja	Kyllä
4	2,37 yli	2,34 yli	4,71	Puolustaja	Ei
5	3,04 alle	12,15 alle	15,19	Hyökkääjä	Ei
6	3,75 alle	12,65 alle	16,40	Hyökkääjä	Ei
7	1,48 yli	15,18 alle	16,66	Hyökkääjä	Ei
8	4,06 alle	16,46 alle	20,52	Hyökkääjä	Ei
9	13,52 alle	3,45 alle	16,97	Hyökkääjä	Ei
10	1,66 yli	13,44 yli	15,10	Puolustaja	Kyllä

Knimellä tehdyn ennusteen (Taulukko 2) maalien tarkkuus oli keskiarvoltaan 73,37 % ja syöttöjen puolestaan 65,93 %. Huonoin ennuste poikkesi oikeasta vastauksesta 20,52 pistettä, mutta lähin ennuste poikkesi vain 3,97 pistettä. Kaikkien Ennustettavien pelaajien kokonaistarkkuus eli maali- ja syöttötarkkuuden keskiavo oli 69,65 %.

Vaikka neuroverkot pyrittiin rakentamaan samanlaisiksi, saatiin Knimellä luodusta neuroverkosta tarkempi ennustus. Ennustuksen tarkkuuteen vaikuttanut suurin ero saattaa olla Ml5.js:n syövereissä. Ml5.js käyttäessä neuroverkolle määrittyy paljon asetuksia, joita käyttäjä ei itse edes tule näkemään tai välttämättä pysty edes niihin vaikuttamaan. Onhan Ml5.js luotu olemaan nimenomaan helppokäyttöinen, jolloin vaikeimmat asiat tehdään automaattisesti. Asetusten automaattinen määrittyminen ei ole siis välttämättä huono asia, mutta sen vuoksi neuroverkkojen rakentaminen samanlaisiksi on hyvin vaikeaa. Ottaen huomioon opetettavan materiaalin vähäisyys, ovat kaikki ennustukset yllättävän tarkkoja. Vaikka minkään pelaajan kohdalla ei ennustus mennyt täysin oikein, oltiin monen pelaajan kohdalla hyvin lähellä oikeaa tulosta.

7.3 Ennustuksiin vaikuttavat tekijät

Tuloksiin vaikuttavia tekijöitä on paljon. Suurimpana vaikuttajana kuitenkin ovat tiedot, joita opettamiseen käytetään. Jääkiekossa, kuten muissakin palloilulajeissa, ennustamiseen vaikuttavia tekijöitä on paljon, oli kyse sitten yksittäisestä urheilijasta tai koko joukkueesta. Tässä työssä käytetyt tilastot olivat hyvin niukat, mutta toimivat demonstrointitarkoituksissa hyvin. Tilastoja voisi tulevaisuudessa kuitenkin täydentää hyvin paljon, sillä jääkiekkopelaajista saa hyvinkin syvällistä tietoa hyödyntämällä esimerkiksi Corsi % -tilasto, joka kertoo joukkueen kiekonhallinnasta pelaajan ollessa kentällä (SM-liiga, 2021). Yksinkertaisin lisäys olisi pelaajien iän lisääminen. Hyvä lisä olisi myös jonkin näköinen joukkueen menestyksen mittarin lisääminen, koska joukkueen vaikutus pelaajan tilastoihin saattaa olla suuri. Joukkueen menestyksen mittarina voisi hyvin toimia esimerkiksi edeltävän kauden runkosarjasijoitus ja merkintä siitä, voittiko joukkue mestaruuden vai ei.

Toiseksi suurin vaikuttaja oli neuroverkkojen rakenne. Koska työn tarkoituksena oli vertailla kahta avoimen lähdekoodin neuroverkkotyökalua, ei neuroverkkojen optimointiin tiettyä työtä varten syvennytty enempää kuin mikä oli toiminnallisuuden kannalta oleellista. Helpoimmat ja nopeimmat tavat kehittää rakenteita olisi esimerkiksi lisäämällä neuronitasoja ja säätää neuronitasojen kokoa. Erilaisten neuronien aktivoinnin ja ennustuksen tarkkuuden laskentaan käytettävien funktioiden toimivuuden ja vaikutuksen tutkiminen olisi hyvä seuraava askel neuroverkkojen optimoinnissa.

8 Johtopäätökset ja pohdinta

Tavoitteena oli vertailla neuroverkkojen luomiseen tarkoitettuja järjestelmiä ja antaa tarpeeksi tietoa neuroverkkojen toiminnasta, jotta mahdollisimman monelle aiheeseen tutustuminen olisi helpompaa. Teoriaosuudessa opetettavaa tietoa piti rajata paljon, jottei oppiminen muuttuisi haastavaksi ja erilasten matemaattisten funktioiden opettelemiseksi. Teoriaosuudesta syntyi kuitenkin helposti pureskeltava ja hyvin ymmärrettävä kokonaisuus. Vaikka teoriaisuudessa asiaa rajattiin paljon, syntyy neuroverkkojen toiminnasta riittävä kokonaisuus, jotta lukijakin pystyy ymmärtämään, mikä on minkäkin koodin pätkän tai Knime-noden merkitys neuroverkossa. Opinnäytetyön toiminnallinen osuus myös tukee toiminnallisuuden oppimista, koska luotavat neuroverkot ovat hyvin yksinkertaisia ja niiden luomiseen käytettävät järjestelmät helppoja ymmärtää.

Molemmilla järjestelmillä saatiin luotua toimiva neuroverkko. Vaikka vertailu olisi ollut mahdollista ilman toimivia neuroverkkoja, ei järjestelmistä olisi saatu niin kattavaa kokemusta kuin mitä lopulta saatiin. Toimivat neuroverkot loivat mahdollisuuden tarkastella koko neuroverkon hyödyntämistä, luomisesta aina käyttämiseen ja tulosten tarkasteluun saakka. Molemmissa järjestelmissä näkyi tavoite helppokäyttöisyydestä, minkä vuoksi oli helppo lähteä rakentamaan neuroverkkoja ilman suurta aikaisempaa tietämystä aiheesta. Käytettävyyden helppoudessa toki täytyy ottaa huomioon, että Knimellä neuroverkko luotiin vasta, kun Ml5.js:llä luotu neuroverkko oli jo valmis. Tämä tietysti opetti paljon neuroverkon luomisesta käytännössä jo ennen Knimen käyttämistä. Knimellä neuroverkkoa luodessa oli useita kohtia, joita osasi säätää ja yhdistää oikealla tavalla, koska oli opittu teorian lisäksi toimivuutta käytännössä Ml5.js:n kanssa. Knimen suurin heikkous, joka hieman jäi järjestelmien käyttöjärjestyksen vuoksi varjoon, oli vähäinen dokumentaatio. Knimellä yksinkertaisen neuroverkon luomisesta ei löytynyt yhtä paljon tietoa kuin Ml5.js:n käyttämisestä. Toivottavasti tämä opinnäytetyö voi tarjota muille ohjeet yksinkertaisen neuroverkon luomiseen Knimellä.

Vaikka järjestelmillä on paljon eroa, saattaa suurin silti olla lähestyttävyys. Käyttäjät, joilla on kokemusta koodaamisesta tai edes pienestä verkkosivujen muokkaamisesta, voivat kokea Ml5.js:n helpommaksi tavaksi oppia, koska Ml5.js ei vaadi uusien työkalujen opettelua. Knime voi taas

puolestaan olla lähestyttävämpi niille, jotka tulevat data-analytiikkan puolelta tai kokevat koodaamisen opettelemisen haastavaksi.

Opetukseen käytettyjen taulukoiden manipulointi Exceliä käyttämällä osoittautui haastavaksi. Tähän kuitenkin löytyi apua Knimestä. Knimen avulla pystyttiin luomaan toinen workflow, jolla pystyttiin luomaan molempiin järjestelmiin tarvittavat JSON- ja CSV-tiedostot. Tämä auttoi paljon Ml5.js käyttämisessä, koska tiedot saatiin jo valmiiksi oikeaan muotoon.

8.1 Knime-neuroverkon tulevaisuuden mahdollisuudet

Knimellä rakennettu neuroverkko toimii hyvin, mutta sitä voisi vielä kehittää edelleen. Oppimista parantaviin asetuksiin voisi paneutua lisää ja tutkia, mitä rakenteellisia muutoksia voi vielä tehdä parantaakseen tuloksia. Olisi myös kannattavaa tutkia, mitä muita neuroverkoihin liittyviä lisäosia Knimeen on saatavilla, ja tutkia, mitä mahdollisesti uusia tapoja ne voisivat tarjota.

Keras tuo myös oman monipuolisen mahdollisuutensa neuroverkon jatkokehittämiseen. Neuroverkon rakentaminen Keraksella loi mahdollisuuden tallentaa neuroverkon opetetun mallin .h5-muotoon, jota Keras pystyy lukemaan ja käyttämään. Tämä siis tarkoittaa, että Knimellä voi luoda neuroverkon ilman, että tarvitsee koodauskokemusta. Rakentamisen jälkeen mallin voi tallentaa ja esimerkiksi pyytää koodaamista taitavaa ihmistä luomaan Kerasta käyttävän sovelluksen, jolle tarvitsee vain syöttää neuroverkon malli. Näin ollen koodarin ei tarvitse tarkkaan tietää millaista dataa varten neuroverkko luodaan tai millaiset asetukset siihen tarvitaan.

8.2 Ml5.js-sovelluksen jatkokehittämismahdollisuudet

Ml5.js on hyvin monipuolinen ja joustava tapa luoda neuroverkko. Koska Ml5.js on JavaScript-kirjasto, voidaan sen toimivuus upottaa lähestulkoon mihin verkkosivuun vain. Toistaiseksi Ml5.js:llä on yhteensopivuusongelmia, mutta tähän saattaa olla tulossa helpotusta, sillä Tenserflow’ta, johon Ml5.js-kirjasto pohjautuu, tuetaan jo esimerkiksi node.js-kirjaston kanssa (Ml5.js, 2021, c).

Työssä tehtyä verkkosivua voisi kehittää vielä paljon. Verkkosivusta voisi vaikka rakentaa sellaisen, jossa neuroverkko olisi yksinkertaistettuna siten, että sen asetuksia voisi säätää ilman koodaamistaitoja. Tällöin käyttäjä voisi luoda tarvitsemansa neuroverkon ja ladata sen mallin verkkosivuilta, jolloin neuroverkon rakentaminen muualle olisi helpompaa, eikä vaatisi koodaamista niin paljon. Ml5.js-kirjastoon kuuluu funktio, jolla opetetun neuroverkon mallin voisi tallentaa JSON-tiedostoksi. Ml5.js-kirjastoa käyttäen voi myös kasata verkon valmiista mallista. Tällaisen verkkosivun rakentaminen olisi hyvinkin mahdollista.

Verkkosivun kanssa voisi myös harkita sen käyttämisen laajentamista opinnäytetyössä käytettyjen jääkiekkotilastojen ennustamiseen. Verkkosivun avulla saattaisi saada luotua lisäkiinnostusta tilastoihin ja niiden merkitykseen lajin seuraamisen parissa ja samalla luoda lähestyttävän tavan tutkia tekoälyn toimimista.

9 Yhteenveto

Opinnäytetyön teoriaosuus opetti minulle paljon neuroverkkojen toiminnasta ja niiden käyttömahdollisuuksista, mikä puolestaan auttoi toiminnallista osaa tehdessä. Ml5.js:n avulla kehitin sekä React.js-osaamistani että JavaScript-osaamistani. Vaikka Knimeä oli hieman käytetty jo ammattikorkeakoulussa, opin työn aikana paljon sen käyttämisestä ja mahdollisuuksista neuroverkkojen luomisessa ja taulukoiden muokkaamisessa. Tutkimuskysymyksien vastaamiseen vaadittavista neuroverkoista saatiin toimivat, ja kysymyksiin saatiin vastattua hyvin. Neuroverkkojen toiminnasta saatiin luotua yksinkertainen ja helposti ymmärrettävä tiivistys, jossa myös hieman käsiteltiin vaatimuksia. Knimen ja Ml5.js:n erot yhtäläisyydet niin toiminnallisuuksiltaan kuin myös järjestelminä saatiin käsiteltyä hyvin, sillä onnistuin rakentamaan toimivat neuroverkot. Onnistuneet neuroverkot myös auttoivat paremmin tarkastelemaan sitä mihin näitä järjestelmiä voidaan hyödyntää.

Neuroverkkoja olisi mielenkiintoista jatkokehittää ja tutkia, kuinka tarkkoja ennustuksia pystyttäisiin jääkiekkotilastoista saamaan. Tämä saattaisi tarkoittaa uusien järjestelmien opettelua. Olisi myös mielenkiintoista tutkia, saisiko esimerkiksi Knimellä luodun neuroverkon mallia käytettyä Ml5.js:n avulla verkkosivuilla. Tällä tavoin voisi luoda verkkosivun, jossa käyttäjät voisivat syöttää tietoja, joita haluaa ennustaa ilman, että verkkosivulle täytyisi rakentaa neuroverkon opettamiseen liittyvää ympäristöä.

Lähteet

Bernico, M., (2018) Deep Learning Quick Reference: Useful Hacks for Training and Optimizing Deep Neural Networks with TensorFlow and Keras
https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=5322203&ppg=1

Graupe, D. (2013). Principles of artificial neural network. World Scientific Publishing Company.
https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=1336559

Kananen, H., Puolitaival, H. (2019). TEKOÄLY – Bisneksen uudet työkalut.
https://bisneskirjasto-almatalent-fi.ezproxy.hamk.fi/teos/BAXBBXATCBIED#/kohta:Lukijalle/piste:tU

Kattan, A., Abdullah, R., Geem, Z.G., (2011). Artificial Neural Network Training and Software Implementation Techniques
https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=3021387&query=

Keller, J., Liu, D. & Fogel, D. (2016). Fundamentals of Computional Intelligence: Neural Networks, Fuzzy Systems, and Evolutional Computation.
https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=4592115

Keras. (n.d.) About keras. Haettu 17.3.2021
https://keras.io/about/

Knime. (n.d. -a) Dokumentaatio. Haettu 18.1.2021
https://docs.knime.com/

Knime. (n.d. -b) Getting started guide. Haettu 18.1.2021
https://www.knime.com/getting-started-guide

Knime. (n.d. -c) Community. Haettu 28.1.2021
https://www.knime.com/knime-community

Knime. (n.d. -c) Knime pythonilla. Haettu 4.2.2021
https://www.knime.com/blog/knime-and-jupyter

Marr, B., (2019). Artificial Intelligence in Practise: How 50 Successful Companies Used AI and Machine Learning to Solve Problems
https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=5751846

ml5.js. (n.d. -a) Dokumentaatio. Haettu 18.1.2021
https://learn.ml5js.org/#/

ml5.js. (n.d. -b) About. Haettu 29.1.2021
https://ml5js.org/about

ml5.js. (n.d. -c) About. Haettu 31.3.2021
https://learn.ml5js.org/#/faq

Morris, S., (n.d.). JavaScript frameworks vs libraries. Haettu 09.02.2021
https://skillcrush.com/blog/javascript-frameworks-vs-libraries/

NPM. (n.d.) About. Haettu 2.3.2021
https://www.npmjs.com/about

SM-Liiga. (n.d.) Edistyneet pelaaja tilastot. Haettu 15.3.2021
https://liiga.fi/fi/tilastot/2020-

2021/runkosarja/pelaajat/?team=&position=all&home_away=&player_stats=enhanced&sort=P#stats-wrapper

SparkAr. (n.d) Spark articles. Haettu 28.1.2021
https://sparkar.facebook.com/ar-studio/learn/articles/fundamentals/features-and-processes

TenserFlow (18.09.2020 -a) Tenserflow documentaatio. Haettu 17.3.2021
https://www.tensorflow.org/api_docs

TenserFlow (18.09.2020 -b) Case Studies. Haettu 17.3.2021
https://www.tensorflow.org/about/case-studies

Warwick, K., (2011). Artificial Intelligence: The Basics
https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=958762

Liite 1: Aineistonhallintasuunnitelma

Opinnäytetyötä varten ei kerätty luottamuksellista tietoa tai muuta arkaluonteista tietoa. Neuroverkkojen opettamiseen käytetty materiaali kerättiin SM-Liiga Oy:n sivuilta (liiga.fi). Sivuilta löytyvän tiedon käyttämistä varten varmistettiin SM-Liiga Oy:ltä, että tietoa saa käyttää opinnäytetyössä. Sivuilta saatu tieto kerättiin Excel-tiedostoon, josta se muutettiin erilaisiksi CSV- ja JSON-tiedostoiksi. Opinnäytetyön aikana syntyneet tulokset myös kerättiin Exceliin. Opinnäytetyön aikana syntyneet taulukot ovat tallessa omalla pöytäkoneellani.

Opinnäytetyöllä ei ollut toimeksiantajaa, eikä tästä syystä työn aikana ollut kokouksia, joista olisi syntynyt pöytäkirjoja.

Liite 2: Ml5.js sovelluksen koodi

```
import logo from './logo.svg';
import './App.css';
import * as ml from 'ml5';

import {useState} from 'react';

function App() {
  const [tulos, setTulos] = useState([]); //Tähän tallennetaan ennustamises
ta saadut tulokset
  const [playerPred, setPlayerspred] = useState([]); //Tähän tallennetaan e
nnustukseen käytettävät tiedot
  const [playerStats, setPlayerstats] = useState(false); // käytetty vain s
ivun päivittämiseen ennustamisen jälkeen
  const [playerInput, setPlayerinput] = useState(); //Tähän tallennetaan op
ettamissen käytettevät inputtia vastaavat tiedot taulukkona
  const [playerOutput, setPlayeroutput] = useState([]); //Tähän tallennetaa
n opettamissen käytettevät outputtia vastaavat tiedot taulukkona

  const Asetukset = {
    inputs:18,
    outputs:3,
    task: 'regression',
    debug: true
  };

  const Neuroverkko = ml.neuralNetwork(Asetukset)

  function loadplayers(){

    for (let index = 0; index < playerInput.length; index++) {
       Neuroverkko.addData(playerInput[index],playerOutput[index]);
    }

    Neuroverkko.normalizeData();
    Opettelu();
  }

  function Opettelu(){
    const opetteluAsetukset = {
      epochs: 100,
      batchSize: 100
    };
    Neuroverkko.train(opetteluAsetukset,KeskenHarjoittelun,ValmisHarjoittel
u);
  }

  function KeskenHarjoittelun(epoch, loss){
    console.log(epoch);
    console.log(loss);
  }

  function ValmisHarjoittelu(){
    console.log("valmista");
    TestiEnnustus();
    console.log(Neuroverkko);
  }

  function TestiEnnustus(){
    Neuroverkko.predict(playerInput[0],Tulokset);
```

```
  }

  function Tulokset(error, result) {
    if(error){
      console.error(error);
      return;
    }
    console.log(result);
}

  const fteamUpload = (event) => {
    var lukija = new FileReader();
    switch(event.target.name){
      case "Input":
        lukija.readAsText(event.target.files[0], 'UTF-8');
        lukija.onload = event => {
          console.log(event.target.result);
          setPlayerinput(JSON.parse(event.target.result));
        }
        break;
      case "Output":lukija.readAsText(event.target.files[0], 'UTF-8');
        lukija.onload = event => {
          console.log(event.target.result);
          setPlayeroutput(JSON.parse(event.target.result));
        }
        break;
    }
  }

  const EPelaaja = (event) => {
    var lukija = new FileReader();
    lukija.readAsText(event.target.files[0], 'UTF-8');
        lukija.onload = event => {
          console.log(event.target.result);
          setPlayerspred(JSON.parse(event.target.result));
        }
  }

  function Ennusta(){
    console.log(playerPred);
    for (let index = 0; index < playerPred.length; index++) {
      Neuroverkko.predict(playerPred[index],listaus);
    }
  }

  function listaus(error, result){
    console.log(result);
    tulos.push({Ottelut:result[0].Ottelut5, Maalit:result[1].Maalit5, Syoto
t:result[2].Syotot5});
    console.log(tulos);
    if (tulos.length == playerPred.length){
      setPlayerstats(true);
    }
    console.log(Neuroverkko);
  }

  return (
    <div className="App">
      <header className="App-header">
        <label>Pelaaja tilastot</label><br></br>
        <label>Pelaaja input tiedosto:</label>
        <input type="file" name="Input" onInput={fteamUpload}/><br></br>
        <label>Pelaaja Output tiedosto:</label>
```

```
          <input type="file" name="Output" onInput={fteamUpload}/><br></br>
          <button onClick={loadplayers}>Aloita oppiminen</button><br></br>
          <br></br>
          <label>Pelaaja Output tiedosto:</label>
          <input type="file" name="input" onInput={EPelaaja}/><br></br>
          <button onClick={Ennusta}>Klikkaa saadaksesi ennustus</button>
          <div>
          {tulos.map(function(res, Ottelut){
            return (<li key={Ottelut}>{res.Ottelut} {res.Maalit} {res.Syotot}
</li>)
          })}
        </div>
        </header>
      </div>
    );
}

export default App;
```

Liite 3: Katkelma opettamiseen käytettävästä CSV-tiedostosta

```
"Pelipaikka","Joukkue","Ottelut","Maalit","Syotot","Joukkue2","Ottelut2","M
aalit2","Syotot2","Joukkue3","Ottelut3","Maalit3","Syotot3","Joukkue4","Ott
elut4","Maalit4","Syotot4","Joukkue5","Ottelut5","Maalit5","Syotot5"
1,13,25,2,5,13,34,5,10,8,54,16,14,8,49,23,22,8,58,29,28
1,14,60,14,32,14,56,20,39,14,10,1,3,14,60,12,32,14,60,19,35
1,7,2,0,0,7,15,0,1,7,60,6,9,9,58,15,18,9,60,16,32
1,8,0,0,0,16,31,3,3,8,58,9,12,13,58,9,22,13,56,12,34
1,7,46,4,17,14,50,15,18,16,54,8,20,10,49,13,24,10,60,13,32
1,10,55,17,11,10,11,1,2,13,45,14,14,13,56,21,14,13,56,17,27
1,11,47,7,15,11,54,8,17,11,60,14,29,2,54,13,25,2,56,16,26
2,3,51,6,9,3,50,4,14,3,57,3,13,3,54,3,14,3,59,4,38
1,11,13,2,1,16,49,5,10,9,59,14,10,9,29,9,10,9,52,17,24
1,0,0,0,0,0,0,0,0,15,56,24,20,2,60,17,29,2,47,24,25
```

Liite 4: Katkelma ennustamiseen käytetystä CSV-tiedostosta

```
Pelipaikka;Joukkue;Ottelut;Maalit;Syotot;Joukkue2;Ottelut2;Maalit2;Syotot2;
Joukkue3;Ottelut3;Maalit3;Syotot3;Joukkue4;Ottelut4;Maalit4;Syotot4;Joukkue
5;Ottelut5;Maalit5;Syotot5
1;10;57;4;5;10;60;9;11;5;50;4;13;5;59;10;20;5;59;12;16
1;17;48;5;3;17;30;0;0;6;20;3;2;6;59;11;5;6;55;17;8
2;17;56;11;18;17;37;5;15;14;56;9;21;14;49;8;17;14;46;5;19
2;0;0;0;0;0;0;0;0;5;1;0;0;5;12;0;0;5;56;3;16
1;0;0;0;0;0;0;0;0;3;58;11;19;3;59;17;15;3;57;20;34
1;0;0;0;0;0;0;0;0;3;53;5;17;16;55;14;36;15;53;16;35
```