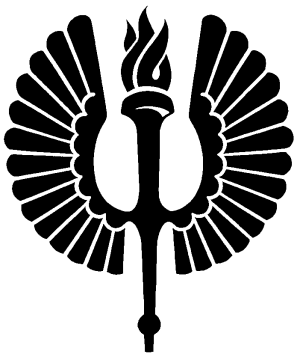


Syväoppivat neuroverkot ja niiden sovellukset

Sara Heininen

Pro gradu -tutkielma
Joulukuu 2020

MATEMATIIKAN JA TILASTOTIETEEN LAITOS
TURUN YLIOPISTO

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

HEININEN, SARA: Syväoppivat neuroverkot ja niiden sovellukset
Pro gradu -tutkielma, 54s.
Sovellettu matematiikka
Joulukuu 2020

Tämä tutkielma käsittelee syväoppivia neuroverkkoja ja niiden sovelluksia. Tutkielmassa käydään läpi neuroverkkojen perusrakenne ja erilaisia neuroverkkorakenteita, neuroverkon kouluttaminen, syväoppivien neuroverkkojen rakenne ja sovellukset sekä tekoälyn ja koneoppimisen perusteita. Lisäksi esitetään esimerkkiongelma, jossa luokitellaan kuvia neuroverkkojen avulla.

Tutkielman johdannossa annetaan lyhyt yleinen kuvaus neuroverkoista, sekä käydään läpi neuroverkkojen historiaa alkaen McCullochin ja Pittsin neuronimallista ja päättyen 1980-luvulla alkaneen neuroverkkojen modernin kehityksen saavutuksiin. Johdannon lopussa annetaan lyhyt kuvaus alan nykypäivän tilanteesta.

Tutkielman alussa käydään läpi tekoälyn ja koneoppimisen perusteita. Tekoälyä käsittelevässä osiossa luodaan yleinen katsaus tekoälyn määritelmään ja sitä tutkivaan tieteenalaan. Koneoppimisen osalta yleisen katsauksen lisäksi käsitellään valvottua ja valvomatonta oppimista sekä kolmea keskeistä koneoppimisongelmaa: luokittelua, regressiota ja klusterointia.

Seuraavaksi tutkielmassa siirrytään käsittelemään neuroverkkoja. Tässä tutkielmassa keinotekoisiin neuroverkkoihin viitataan yleensä yksinkertaisesti termillä "neuroverkot". Tutkielmassa käydään läpi neuroverkkojen perusteita, ominaisuuksia, rakennetta sekä erilaisia aktivaatiofunktioita. Eri neuroverkkorakenteista käsitellään monikerroksinen perseptroni, takaisinkytketty neuroverkko ja itseorganisoituva kartta. Neuroverkkojen kouluttamiseen liittyen käydään perusperiaatteiden lisäksi läpi gradientin laskeutuminen, vastavirta-algoritmi, koulutus-, testaus- ja arviointiaineisto, sekä ali- ja ylisovittaminen.

Neuroverkkojen jälkeen tutkielmassa siirrytään käsittelemään syväoppivia neuroverkkoja yksityiskohtaisemmin. Yleisen kuvauksen lisäksi syväoppiviin neuroverkkoihin liittyen käsitellään logistista regressiota, syväoppivien neuroverkkojen sovelluksia sekä syväoppivia neuroverkkorakenteita, joista esitellään konvoluutioneuroverkko ja pinotut autoenkooderit.

Asiasanat: keinotekoiset neuroverkot, syväoppivat neuroverkot, syväoppiminen, koneoppiminen, tekoäly.

Sisältö

1 Johdanto

Älykkään koneen rakentaminen on ollut useiden eri tieteenalojen tutkijoiden yhteisenä unelmana jo pitkään. [2] Älykkyyden määritelmästä on kuitenkin paljon ristiriitaisia mielipiteitä ja sen perimmäisestä luonteesta ymmärretään vain vähän. Nykyisen ymmärryksen perusteella älykäs kone voidaan määritellä esimerkiksi seuraavasti: Konetta voidaan pitää älykkäänä, jos se kykenee ymmärtämään tietyn ympäristön tai prosessin, tunnistamaan erilaisia ympäristön tai prosessin muuttujia, niiden välisiä suhteita ja niiden vaikutuksia, tekemään kyseisten muuttujien, niiden suhteiden ja vaikutusten perusteella päätelmiä, ja oppimaan tällä tavalla asioita kyseisestä ympäristöstä tai prosessista sekä sen poikkeamista ja käyttöolosuhteista. Perinteisen tekoälyn tieteenalalla tämän tyyppistä älykkyyttä on yritetty simuloida systeemeillä, jotka vaativat täydellistä tietämyksen esitysmuotoa (engl. knowledge representation), eli joille tiedot analysoitavasta prosessista on tarjottava kokonaisena ja tarkkana matemaattisena esityksenä. Monia todellisen maailman prosesseja ei kuitenkaan voida esittää tällä tavalla, ja on osoitettu, että hyvin monimutkaisen matemaattisen muotoilun käyttö saattaa selvästi rajoittaa prosessien mallinnusta. Systeemin olisi pystyttävä paitsi käsittelemään epätäydellistä tietoa mallinnettavasta prosessista myös pärjäämään huomattavien odottamattomien prosessissa tai sen käyttöolosuhteissa tapahtuvien muutosten kanssa. Laskennallinen älykkyys (engl. computational intelligence) vastaa näihin haasteisiin; sen menetelmillä voidaan rakentaa niin sanottuja älykkäitä systeemejä, jotka tiettyjen algoritmien mukaan toimimisen sijaan pystyvät tekemään päätelmiä prosessista tai ympäristöstä vähäisillä ennakkotiedoilla ja toimimaan joustavasti ja sopeutumaan odottamattomiin muutoksiin. Tällaiset systeemit ovat hyviä ratkaisemaan ongelmia liittyen huonosti määriteltyihin, epälineaarisiin, kompleksisiin, stokastisiin tai aikaan sidoksissa oleviin (engl. time-variant) prosesseihin. [4] Keinotekoiset neuroverkot kuuluvat tällaisiin laskennallisesti älykkäisiin systeemeihin.

Keinotekoiset neuroverkot (engl. artificial neural networks, lyh. ANN) ovat koneoppimisen tieteenalaan kuuluvia laskennallisia malleja. [8] Niiden rakenne jäljittelee elävien olentojen hermoston rakennetta. [5] Keinotekoi-

set neuroverkot rakentuvat toisiinsa liitetyistä keinotekoisista neuroneista eli prosessointiyksiköistä, jotka toteutetaan matemaattisesti vektoreilla ja neuronien välisiin liitoksiin liitetyistä painoista koostuvilla matriiseilla. [2] "Neuroverkot"termin biologinen konnotaatio saattaakin olla harhaanjohtava: vaikka keinotekoiset neuroverkot ovat saaneet innoituksensa biologisista neuroverkoista, ne imitoivat elävien olentojen aivoissa esiintyviä neuroverkkoja hyvin karkeasti. Ne tulisikin nähdä ensisijaisesti matemaattisina objekteina. [1] [3] Niiden maine moninaisten ongelmien tehokkaina ratkaisijoina perustuu niiden suureen kapasiteettiin kuvata epälineaarisia systeemejä ja oppia asioita tällaisten systeemien taustalla olevasta käyttäytymisestä niistä saatavan datan perusteella. Tämä onkin yksi keinotekoisten neuroverkkojen viehättävimmistä ominaisuuksista. [2]

Neuroverkot vaikuttavat ilmestyneen tyhjästä 1980-luvun lopulla, mutta todellisuudessa niiden historia alkaa vuodesta 1943, kun McCulloch ja Pitts esittivät ensimmäisen biologiseen neuroniin perustuvan matemaattisen mallin artikkelissaan “A logical calculus of the ideas immanent in nervous activity”. Heidän mallinsa johti ensimmäisen keinotekoisen neuronin kehittämiseen, ja loi näin pohjan keinotekoisten neuroverkkojen alalle tekoälyn osa-alueena. [2] [8] [7] Ensimmäinen neuroverkkojen kouluttamiseen käytettävä menetelmä kehitettiin muutama vuosi myöhemmin vuonna 1949. Kyseisen menetelmän nimi on kehittäjänsä mukaan Hebbin sääntö (engl. Hebb's rule), ja se perustui neurofysiologisiin hypoteeseihin ja havaintoihin. [2]

Neuroverkkojen aikainen kehitys jatkui 1950-luvulla, kun neurobiologi Frank Rosenblatt kehitti neuronin kaltaisen, perseptroniksi kutsutun elementin. Se laskee syötteidensä painotetun summan, vähentää summasta kynnysarvon ja palauttaa tuloksena yhden kahdesta mahdollisesta arvosta. [4] Perseptronin kyky tunnistaa yksinkertaisia kuvioita herätti kiinnostusta, ja vuonna 1960 Widrow ja Hoff kehittivätkin kaksi uutta mallia: ADALINEn, joka on lyhennys sanoista "adaptive linear element", ja hieman myöhemmin MADALINEn, eli "multiple ADALINEn". Ensimmäinen todellisen elämän ongelman ratkaisemiseen käytetty neuroverkko oli muodostettu näistä malleista; kyseisen verkon tehtävä oli poistaa kaiut puhelinlinjoista. [2] [4]

Alkuvaiheen työ innosti tutkijoita tekemään lisää tutkimusta neuroverk-

kojen parissa, kunnes ala koki merkittävän takaiskun vuonna 1969, kun Minsky ja Papert julkaisivat kirjan “Perceptrons: An Introduction to Computation Geometry”. Kirjassaan he osoittivat ajan yksikerroksisten neuroverkkojen kuten perseptronin ja ADALINEn olleen kykenemättömiä luokittelemaan epälineaarisesti separoituviin luokkiin kuuluvia kuvioita. He osoittivat, etteivät neuroverkot pystyneet oppimaan hyvin yksinkertaisten loogisten funktioiden kuten "exclusive or-funktion syötteiden ja ulostulojen välistä suhdetta, ja tekivät kirjansa viimeisessä luvussa päätelmän: heidän mukaansa neuroverkkojen oppimisen teorian laajentaminen monimutkaisempiin monikerroksisiin algoritmeihin oli toivotonta. [2] [7] Neuroverkkojen tutkimus ja sen rahoitus vähenivät dramaattisesti, ja täyttymättömät lupaukset saivat pettymyksen tunteen laskeutumaan alan ylle. [2] [4] [7] Vaikka Minskyn ja Papertin artikkeli oli syynä neuroverkkojen tutkimuksen keskeytymiseen, on kuitenkin todennäköistä että kiinnostus alaa kohtaan olisi vähentynyt joka tapauksessa neuraalisten prosessien puutteellisen ymmärryksen ja riittämättömän laskentatehon takia. Tarvittiin aikaa uusien ideoiden, tietämyksen esitysmuotojen ja oppimisen teorian kehittämiseen. [7]

Takaiskusta huolimatta eräät lähinnä kognitiivisen psykologian alan tutkijat jatkoivat työtään ja edistivät alaa myös 1970-luvulla. Näihin tutkijoihin kuuluivat esimerkiksi James Anderson, Stephen Grossberg, John Hopfield sekä Teuvo Kohonen, jonka tutkimus johti kuuluisten itseorganisoituvien karttojen (engl. self-organizing maps, Kohonen networks) kehittämiseen. [7]

Neuroverkkojen tutkimus muuttui aktiivisemmaksi vasta 1980-luvulla. Vuonna 1982 John Hopfield julkaisi artikkelin, jossa hänen tavoitteensa ei ollut vain mallintaa aivojen toimintaa, vaan luoda hyödyllisiä laitteita. Hän osoitti selkeästi matemaattisella analyysillä, mitä tällaiset verkot voisivat tehdä ja miten ne voisivat toimia. [4] Seuraava askel eteenpäin tuli vuonna 1986, kun Rumelhart ja McClelland julkaisivat vastavirta-algoritmin (engl. backpropagation), epälineaarisen neruroverkkojen koulutusalgortimin, joka voitti monet perseptronin ja ADALINEn rajoituksista. Tämä algoritmi ei ollut uusi: sekä Paul Werbos että David Parker olivat kehittäneet hyvin samankaltaiset algoritmit, Werbos vuonna 1974 ja Parker vuonna 1985. Rumelhart ja McClelland kuitenkin julkaisivat algoritminsa yhteydessä tutkimuskokonai-

suuden, joka tuki vastavirta-algoritmin soveltuvuutta kiinnostavien ja moninaisten ongelmien ratkaisemiseen, ja oli laajan tutkijoiden joukon saatavilla. Vuoden 1986 jälkeen neuroverkkoihin liittyvän tutkimuksen ja sen rahoituksen määrät alkoivat kasvaa eksponentiaalisesti, ja neuroverkoista tuli neljänkymmenen vuoden tekeillä olon jälkeen nopeasti sensaatio. [7] Suureen suosioon vaikuttivat myös tietokoneiden ja niiden muisti- ja prosessointikykyjen kehitys, tehokkaampien ja paremmin virheitä kestävien optimointialgoritmien kehittyminen sekä biologisiin hermostoihin liittyvät uudet löydökset. [2]

Neuroverkkojen alan modernin kehityksen 1980-luvulla aloittaneet tutkijat pitivät työtään varmaankin onnistuneena, mutta heidän työnsä menestyksen syy ei ole se mitä he odottivat. Alkuperäinen motivaatio neuroverkkojen kehittämiselle oli ajatus hermoston kaltaisen systeemin rakentamisesta; Aivoja mukailevan koneen rakentamista pidettiin lupaavana tutkimussuuntana, koska hyvinkin yksinkertaiset hermostot ovat monissa tehtävissä huomattavasti tietokoneita kyvykkäämpiä. Neuroverkkotutkimuksen ensimmäinen aalto 1960-luvulla oli saanut alkunsa samoista lähtökohdista, mutta sen epäonnistumisen syinä olleet matemaattisten ja laskennallisten työkalujen puute eivät ole ongelma enää nykypäivänä. Tämä onkin neuroverkkojen historiaan liittyvä paradoksi; vaikka tietokoneet ja koneoppimisen matemaattiset ja tilastotieteelliset keinot ovat edistyneet valtavasti, hermoston toiminnan jäljittely on vaikeaa, sillä aivojen toiminta hermoston tietojenkäsittelymekanismeista lähtien on edelleen pitkälti tuntematonta. [1]

Aivojen perimmäinen rakennuspalikka on neuroni, tietyntyyppinen solu, joita on aivoissa noin miljardi kappaletta. Biologinen neuroni vastaanottaa syötteitä, yhdistää ne jollakin tavalla, suorittaa yleensä epälineaarisen operaation syötteiden yhdistelmälle ja antaa ulostulona lopullisen tuloksen. Kuitenkin aivojen täsmällinen toiminta on edelleen mysteeri. [4] Onkin siis epätodennäköistä, että hyödyllisiä koneita voitaisiin rakentaa imitoimalla mekanismeja, joiden toiminta on pohjimmiltaan tuntematonta. Neuroverkkojen menestys koneoppimisen menetelmänä ei siis johdu niiden tavasta jäljitellä aivoja; Ne kannattaa pitkälti erottaa biologisesta taustastaan ja nähdä matemaattisten ja tilastotieteellisten keinojen kautta, sillä kehitys koneoppimisen

saralla on tapahtunut ja odotettavasti myös tulee tapahtumaan tällä tavalla. [1]

Viime vuosina lukuisat käytännön sovellukset ja suuri määrä uutta tutkimusta ovat mahdollistaneet neuroverkkojen teorian kehityksen. [2] Kehityksen uuden aallon 1980-luvulla tapahtuneen alun jälkeen on kehitetty valtavasti tapoja parannella vastavirta-algoritmia, yletön määrä uusia algoritmeja sekä monia olennaisesti uusia ja erilaisia neuroverkkorakenteita. Neuroverkkoja on sovellettu lähes jokaisella tieteen ja insinööritaidon alalla, neuroverkkotekniikkaa sisältävät kaupalliset tuotteet ovat menestyneet markkinoilla, ja neuroverkkojen teolliset sovellukset ovat parantaneet tuottavuutta ja laatua sekä vähentäneet kuluja. [7] Neuroverkkoja on kuitenkin alettu tutkia toden teolla vasta 1990-luvun alussa, ja niillä on kaiken jo tapahtuneen kehityksen jälkeenkin edelleen valtavaa potentiaalia tutkimuksen aiheena. [2]

2 Tekoäly

Älykkyydestä tiedetään vain vähän. Sen määritelmästä kiistellään edelleen, ja suuri osa siihen liittyvästä ymmärryksestä tulee todennäköisesti pysymään ihmisten käsityskyvyn saavuttamattomissa. Varteenotettava määrä tutkimusta keskittyykin nykyisin älykkyyden ymmärtämiseen ja esittämiseen. [4]

Tekoälyn (engl. artificial intelligence, lyh. AI) tieteenalan tavoitteena on älykkyyden ymmärtäminen ja älykkäiden järjestelmien rakentaminen. Yhden tietyn lähestymistavan tai menetelmän käyttämisen sijaan tekoälyn tieteenala pyrkii kehittämään älykkäitä järjestelmiä mahdollisimman moniin eri tarkoituksiin. Koska tehtävät, joita varten älykkäitä toimijoita kehitetään, saattavat poiketa toisistaan hyvinkin paljon, tieteenalalla käytetyt menetelmät ovat moninaisia, ja tekoäly onkin jakautunut useisiin eri aloihin joilla yhteistä tavoitetta lähestytään eri tavoin. Tekoäly tarjoaa laajan valikoiman tehokkaita työkaluja monenlaisiin eri alojen sovelluskohteisiin, ja myös hyödyntää mielenkiintoisia tuloksia useilta muilta tieteenaloilta, joihin kuuluvat esimerkiksi logiikka, operaatioanalyysi, tilastotiede, säätötekniikka, kuvankäsittely, kielitiede, filosofia, psykologia ja neurobiologia. Tekoäly onkin poikkeuksellisen poikkitieteellinen ala, ja monimuotoisuutensa vuoksi siihen liittyvien

projektien kehittäminen saattaa joskus olla monimutkaista, mutta lähes aina myös erittäin jännittävää. [8]

Vaikka tekoälyn tavoite tieteenalana on selkeä, itse tekoälyä on vaikea määritellä. Ensimmäisenä tätä yritti John McCarthy, joka vuonna 1955 määritteli tekoälyn jotakuinkin seuraavasti: Tekoälyn päämäärä on kehittää koneita, jotka käyttäytyvät kuin ne olisivat älykkäitä. Tämä määritelmä on kuitenkin puutteellinen; jo hyvin yksinkertaisilla virtapiireillä toimivat pienet liikkuvat robotit, kuten niin kutsutut Braitenbergin ajoneuvot (engl. Braitenberg vehicles), pystyvät tuottamaan älykkäältä vaikuttavaa käytöstä, kuten ryhmien muodostamista, yhteentörmäyksien välttämistä, johtajan seuraamista ja aggressiivisuutta. Kuitenkin on selvää, että sellaiset monimutkaiset käytännön ongelmat, joiden ratkaiseminen on tekoälyn tavoitteena, ovat aivan liian vaativia tällaisille pienille roboteille. Tekoälylle onkin siis täytynyt keksiä uusia määritelmiä. Erään uusista yrityksistä teki Elaine Rich, joka määritteli tekoälyn seuraavasti: Tekoäly on tutkimusta siitä, miten saada tietokoneet tekemään asioita joissa ihmiset ovat tällä hetkellä parempia. Tämä ytimekäs määritelmä sekä kuvailee sitä miten tekoälyn tutkimus on kehittynyt viimeisten viidenkymmenen vuoden aikana, että säilyttää ajantasaisuutensa myös pitkälle tulevaisuuteen. [8]

Laskennallinen älykkyys ja sen piiriin kuuluvat neuroverkot ovat osa tekoälyä, ja niiden menestys on edesauttanut koko tekoälyn tieteenalan kehitystä: Noin vuonna 2010, kun neuroverkkoja oli tutkittu aktiivisesti noin kahdenkymmenenviiden vuoden ajan, oltiin onnistuttu rakentamaan syviä neuroverkkoja, jotka oppivat luokittelemaan kuvia hyvin suurella tarkkuudella. Koska kuvien luokittelu on elintärkeä ominaisuus älykkäille roboteille, tämä kehitys käynnisti tekoälyn vallankumouksen, joka tulee puolestaan johtamaan sellaisiin elämää muuttaviin sovelluksiin kuten itseohjautuvat autot, palvelurobotit sekä älykodit. Käytännön tekoäly onkin jo kehittynyt insinööritaidon alaksi, jossa ohjelmistoja kehitetään teollisuudessa suurissa eri erikoistumisalojen asiantuntijoista koostuvissa tiimeissä. Teknologia kehittyy nopeasti ja siksi onkin tärkeää muistaa, että kasvun rajat on ylitetty jo pitkän aikaa sitten, ja kehityksellä on myös varjopuolensa. Uutta teknologiaa kehitettäessä ja uusia keksintöjä toimeen pantaessa täytyy siis ajatella myös

vastuullisuutta ja kestävyyttä. [8]

3 Koneoppiminen

Yksi ihmisälyn erityisitä vahvuuksista on sen sopeutumiskyky: ihmiset pystyvät mukautumaan erilaisiin ympäristöihin ja olosuhteisiin, ja muuttamaan käytöstään olosuhteita vastaavaksi oppimalla. [8] Koneelle sopeutumiskyky puolestaan tarkoittaa kykyä muuttaa tai kehittää sen omia parametreja tai rakennetta saavuttaakseen tavoitteensa paremmin. [4] Tämänkaltaisten sopeutumiskykyisten koneiden tutkimusta kutsutaan koneoppimiseksi.

Koneoppiminen (engl. machine learning) tutkii tietokonealgoritmeja, jotka kehittyvät automaattisesti kokemuksen kautta. [8]. Koska ihmisten kyky oppia kokemuksesta on huomattavasti parempi kuin tietokoneiden ja koska tekoälyn tieteenalalla koneet pyritään saamaan tekemään juuri niitä asioita joissa ihmiset ovat tällä hetkellä parempia, koneoppiminen ja siihen liittyvien algoritmien ja oppimismekanismien kehittäminen ja tutkiminen on keskeinen tekoälyn osa-alue. Neuroverkot ovat koneoppimisen tieteenalan osa-alue. [8]

Koneen älykkään käytöksen rakenne saattaa olla niin monimutkainen, että sen ohjelmoiminen optimaalisesti on hyvin vaikeaa tai jopa mahdotonta. Tällöin tarvitaan koneoppimisalgoritmeja: koneen käytös voidaan toteuttaa ohjelmoidun ja opitun käytöksen yhdistelmänä, jossa koneoppimisalgoritmi on vastuussa koneen oppimisesta. [8] Koneoppiminen onkin uusin useista yrityksistä muuntaa ihmisten tietämys ja päättelykyky sellaiseen muotoon, että sen avulla voitaisiin suunnitella ja rakentaa koneita ja automaattisia systeemejä. Ilmiselviltä vaikuttavien, mutta kuitenkin vaikeasti formalisoitavien konseptien ilmaisemiseen käytetään koneoppimisessa matematiikkaa, mikä auttaa kulloinkin ratkaistavana olevan tehtävän ymmärtämisessä. [11] Kaikkia konsepteja ei kuitenkaan voida määritellä riittävän tarkasti. Tällaiset konseptit on esitettävä koneelle esimerkkien muodossa. Koneen on jollakin tavalla muunnettava annetut esimerkit tiedoksi; jälleen siis tarvitaan koneoppimisalgoritmeja. [9]

Koneoppimisen ydin muodostuu kolmesta perustavanlaatuisesta konseptista: datasta, mallista ja oppimisesta. Keskeistä on suunnitella algoritmeja,

jotka saavat arvokasta tietoa irti datasta automaattisesti. Tämän prosessin halutaan nimenomaan olevan automaattinen, sillä tavoitteena on kehittää yleiskäyttöisiä menetelmiä, joilla saadaan merkityksellisiä tuloksia monista eri dataseteistä, mieluiten ilman kyseisen sovellusalan täyttä asiantuntijuutta. Tämän tavoitteen saavuttamiseksi luodaan yleensä analysoitavaa datasettiä muistuttavaa dataa tuottavan prosessin kaltainen malli. Mallin sanotaan oppivan, jos se löytää automaattisesti kuvioita ja rakenteita datasta optimoimalla parametrejaan, ja sen sanotaan oppivan datasta, jos sen suoritus paranee, kun se on saanut kokemusta datasta. [11]

Malli on kuvaus jostakin näkyvissä tai havaittavissa olevan maailman osasta, joka on tyypillisesti jonkinlainen dataa tuottava prosessi. Jos malli kuvataan matemaattisesti, se koostuu algebrallisista yhtälöistä tai differentiaaliyhtälöistä, jotka kuvaavat syyt, kuten muuttujat tai mallin syötteet, vaikutuksiksi, kuten mallinnettaviksi suureiksi tai mallin ulostuloiksi. Matemaattisessa mallissa sekä syiden, eli muuttujien, että vaikutusten, eli mallinnettavien suureiden, saamat arvot ovat numeroita. [1] [11] Malli on siis mallinnuksen kannalta oleelliset asiat sisältävä yksinkertaistettu versio sen kuvaamasta todellisesta tuntemattomasta prosessista, ja hyvää mallia voidaan käyttää todellisen maailman tapahtumien ennustamiseen. [11]

Kun tutkittava datasetti ja siihen sopiva malli on muodostettu, malli voidaan kouluttaa. Malli koulutetaan optimoimalla sen tietyt parametrit käytettävissä olevan datan avulla. Mallin kykyä ennustaa käytettävää dataa mitataan hyöty- tai virhefunktion avulla. Mallin toivotaan tietysti olevan hyvä, mutta tämä tavoite ei ole täysin selkeä; yksi koneoppimisen pääkysymyksistä on, mitä hyvillä malleilla tarkoitetaan. Sanan "hyvä"objektiivinen määritteleminen ei ole täysin itsestäänselvää, mutta ohjenuorana voidaan käyttää periaatetta, että hyvän mallin pitäisi yleistää oppimaansa hyvin eli suoriutua hyvin uuden ja ennennäkemättömän datan analysoimisesta. Mallin hyvä suoriutuminen tutun datan ennustamisesta ei välttämättä tarkoita että malli on oppinut, sillä malli on saattanut opetella käytetyn datan ulkoa. Tällainen malli ei välttämättä yleisty hyvin uuden datan ennustamiseen, ja käytännössä onkin usein tärkeää altistaa koneoppimisjärjestelmä tilanteille joita se ei ole vielä kohdannut. [11]

Koneiden kyky oppia oli pitkään vain unelma. Rosenblattin perseptroni herätti toki kiinnostusta, mutta sen aiheuttama innostus jäi lyhytikäiseksi, ja sitä seuranneet yritykset pärjäsivät vielä heikommin. Läpimurtoa ei tapahtunut ja rahoitus oli vähäistä, ja niinpä koneoppiminen jäi muiden menestyneempien tieteenalojen varjoon. Kaikki kuitenkin alkoi muuttua 1970-luvulla, kun silloin suosituissa tietopohjaisissa tietämysjärjestelmissä (engl. knowledge-based systems) huomattiin heikkous: mistä järjestelmien tarvitsema tieto tulisi? Vallitsevan mielipiteen mukaan sen kuuluisi muodostua insinöörien ja alan asiantuntijoiden yhdessä kokoamista jos-niin-säännöistä. Käytännön yritykset eivät kuitenkaan olleet vakuuttavia, kommunikointi insinöörien ja asiantuntijoiden välillä oli haastavaa, ja suurien, kymmenistä tuhansista säännöistä koostuvien tietokantojen kokoaminen osoittautui turhauttavaksi. Ryhmä visionäärejä teki ongelman korjaamiseksi yksinkertaisen mutta uskaliaan ehdotuksen: sen sijaan että koneelle annettaisiin ohjeet tietyn ongelman ratkaisemiseksi, ohjeet voitaisiin antaa epäsuorasti välittämällä tarpeelliset taidot antamalla esimerkkejä, joista kone voisi oppia. Jotta tämä onnistuisi, tarvittiin oppimisen toteuttamiseen kuitenkin algoritmeja, joiden puute olikin tähän uuteen ehdotukseen liittyvistä vaikeuksista suurin. Tarvittavien koneoppimisen tekniikoiden puute ei kuitenkaan ollut este vaan innoittava haaste. Ajatus koneiden varustamisesta oppimistaidoilla herätti suurta innostusta, joka räjähti vuonna 1983, kun "Machine Learning: The AI Approach"julkaistiin. Kyseinen kirja koostui suuresta määrästä tutkimuksia, joissa ehdotettiin mitä moninaisempia tapoja käsitellä koneiden oppimista, ja uusi tieteenala syntyi sen vaikutuksen alaisena lähes yhdessä yössä. Näinä alkuaikoina tärkeitä kysymyksiä olivat paitsi miten oppia, myös mitä oppia ja miten. Tämä aika oli luovaa, mutta monet hyvät ajatukset valitettavasti hylättiin myöhemmin. Realististen sovellusten käytännön tarpeet näyttivät lupaavimman suunnan kehitykselle, ja tutkimusmenetelmät muuttuivat järjestelmällisemmiksi. Oppimisalgoritmien suoritusten vertailu mahdollistui, tilastotieteelliset arviointimenetelmät levisivät laajalle, vapaasti yleisessä käytössä olevat versiot monista suosituista ohjelmistoista tulivat saataville, ja koneoppimisen parissa toimivien asiantuntijoiden määrä kasvoi tuhansiin. Nykyisin hupputeknologiaan keskittyvät yritykset perustavat

omia koneoppimisryhmiä, ja useat yliopistot opettavat koneoppimista jo kandidaattivaiheen opiskelijoille. [9]

3.1 Koneoppimisongelmat

Kolme tyypillistä koneoppimisen avulla ratkaistavaa ongelmaa ovat luokittelu, regressio ja klusterointi. Tässä osiossa esitellään nämä kolme ongelmatyyppiä.

3.1.1 Luokittelu

Luokittelussa koneoppimisalgoritmi määrää automaattisesti annetulle kuviolle luokan. Tarkemmin sanottuna, jos algoritmille syötteenä annetun kuvion piirrevektori sisältää n piirrettä, algoritmin tehtävänä on löytää n-ulotteiseen piirreavaruuteen kuuluva $n-1$ -ulotteinen hypertaso, joka jakaa kuvioiden luokat mahdollisimman hyvin eli siten, että väärin luokiteltujen kuvioiden prosentuaalinen osuus kaikista kuvioista on mahdollisimman pieni. Tällaista algoritmia, joka pystyy jakamaan piirrevektoreita äärelliseen määrään luokkia, kutsutaan luokittelijaksi. Ratkaistakseen luokitteluongelman luokittelija tarvitsee siis joukon kuvioita, kuvioita vastaavat piirrevektorit jotka sisältävät luokittelutehtävälle oleelliset tiedot kuvioista, ja joukon luokkia, joihin kuviot jaetaan, ja sen tehtävänä on kuvata syötteenä saadut piirrevektorit oikeisiin luokkien arvoihin. [1] [8]

3.1.2 Regressio

Regressiossa tietylle syötteelle odotettu ulostulo on tietyn luokan sijaan jatkuvalle välille kuuluva lukuarvo. [9] Regressiota suorittavan algoritmin tehtävänä on siis löytää funktio, joka kuvaa algoritmin saamat syötteet niitä vastaaviksi reaaliarvoisiksi ulostuloiksi. Tämän regressiofunktion löytämiseksi on tehtävä oletus liittyen funktion lineaarisuuteen: Helpointa on olettaa funktion olevan lineaarinen. Jos tämän oletuksen pohjalta tehty malli ei kuitenkaan ole tyydyttävä, on oletettava että kuvattava funktio on epälineaarinen ja tehtävä uusi epälineaarinen malli, kuten polynomi tai neuroverkko. Regressio on perustavanlaatuinen koneoppimisongelma ja tärkeä osa myös

luokitteluun tarkoitettuja algoritmeja. Monet luokitteluongelmat voidaankin nähdä epälineaarisina regressio-ongelmina. [1] [11]

3.1.3 Klusterointi

Toisin kuin luokittelussa ja regressiossa, klusteroinnissa koulutusaineistoon kuuluvia syötteitä vastaavia ulostuloja ei anneta koneoppimisalgoritmille. Algoritmille annettava data ei siis ole etukäteen strukturoitua, ja klusteroinnin ideana onkin rakenteiden löytäminen datasta: Algoritmin tehtävänä on jakaa samankaltaiset syötteet klustereiksi kutsuttuihin ryhmiin. Syötteenä algoritmille annetaan joukko piirrevektoreiden kuvaamia esimerkkejä ilman odotettuja luokkia, ja ulostulona se antaa vähintään kaksi esimerkeistä koostuvaa klusteria. Klusterointi kuuluu olennaisena osana valvomattomaan oppimiseen. [8] [9]

3.2 Valvottu ja valvomaton oppiminen

Suurin osa koneoppimisalgoritmeista voidaan jakaa kahteen luokkaan, joita kutsutaan valvotuksi ja valvomattomaksi oppimiseksi. [8] Valvotussa oppimisessa (engl. supervised learning) algoritmin käytettävissä on syötteiden lisäksi niitä vastaavat toivotut ulostulot, eli jokainen käytettävään koulutusaineistoon kuuluva esimerkki koostuu piirrevektorista ja halutusta ulostulosta. Kun tällainen mallinnettavaa prosessia ja sen käytöstä kuvaava koulutusaineisto on saatavilla, voidaan käyttää valvottua oppimisalgoritmia, joka opettajan tapaan kertoo koulutettavalle mallille, mikä on oikea reaktio kuhunkin mallin syötteenä saamaan esimerkkiin. [2] Käytettävä koulutusaineisto osoittaa siis miten mallin kuuluisi toimia: mallin saamilleen syötteille antamia ulostuloja verrataan koulutusaineistoon sisältyviin tavoitteena oleviin ulostuloihin, minkä jälkeen mallin parametreja muutetaan käytettävän oppimisalgoritmin avulla jotta mallin ulostulot saataisiin lähemmäs tavoiteulostuloja. [4] Valvottu oppiminen on nykyisin vakiintunut ala, ja sillä on monia menestyneitä sovelluskohteita. [8] Valvottuun oppimiseen kuuluvia oppimisalgoritmeja ovat muun muassa gradientin laskeutuminen ja vastavirta-algoritmi. [4]

Toisin kuin valvotussa oppimisessa, valvomattomassa oppimisessa (engl. unsupervised learning) malli ei tarvitse tietoa syötteitä vastaavista toivotuista ulostuloista. [2] Tavoiteulostuloja ei siis tiedetä, eikä mallin parametrien muuttamiseen käytetä mitään ulkoisia signaaleja. Koska ei voida tietää onko mallin syötteelle antama ulostulo oikea, oppimisessa etsitään syötteistä tiettyjä piirteitä, säännönmukaisuuksia tai trendejä, ja malliin tehdään muutoksia sen toiminnan perusteella. Malli siis oppii reagoimaan toistuvasti esiintyviin kuvioihin vaikka sille ei erikseen kerrota onko se oikeassa vai väärässä. Valvomaton oppiminen sopii yleensä klusterointiin, piirteiden erottamiseen (engl. feature extraction), luokitteluun ja samankaltaisuuden mittaamiseen (engl. similarity measure). Tällä hetkellä valvomatonta oppimista ei kuitenkaan ymmärretä hyvin, ja se on edelleen tutkimuksen alaisena. Neuroverkkojen yhteydessä valvomatonta oppimista käytetään nykyisin vain itseorganisoituvien karttojen kouluttamisessa. [4]

Valvotun ja valvomattoman oppimisen lisäksi on mahdollista käyttää puolivalvottua oppimista (engl. semi-supervised learning) tai vahvistusoppimista (engl. reinforcement learning). Puolivalvotussa oppimisessa mallin syötteenä saamaan koulutusaineistoon kuuluvista esimerkeistä vain osalle on ilmoitettu niitä vastaavat toivotut ulostulot. Vahvistusoppimisessa taas malli saa vain silloin tällöin ympäristöltään positiivista tai negatiivista palautetta toiminnastaan. Mallin toiminta vaikuttaa sen ympäristöön, joka ohjaa oppimista antamalla mallille palautetta, ja näin malli oppii miten tietyn syötteen kohdalla tulee toimia. [8] [4]

4 Neuroverkot

Keinotekoiset neuroverkot (engl. artificial neural networks, lyh. ANN) ovat datan käsittelyyn käytettäviä koneoppimisen työkaluja, joille on ominaista hyvä laskentateho, vikasietoisuus (engl. fault tolerance), oppiminen kokemusperäisestä datasta ja kyky yleistää. Ne ovat pohjimmiltaan matalan tason laskennallisia algoritmeja, jotka suoriutuvat yleensä hyvin numeerisen datan käsittelystä. Ne pystyvät sovittamaan epälineaarisia funktioita kokeelliseen dataan, ja niiden luontainen etu onkin niiden kyky käsitellä ja mallintaa

epälineaarisia systeemejä. Kuten mitä tahansa muutakin tilastollista mentelmää käytettäessä, asianmukaisen ja riittävän datan käytettävissä oleminen on neuroverkkoja käytettäessä välttämätöntä. [1] [4]

Neuroverkko koostuu neuroneista, joita kutsutaan myös prosessointielementeiksi (engl. processing element, lyh. PE). Yhden neuroverkon neuronit ovat yleensä kaikki samanlaisia. Neuroverkossa neuronit on yhdistetty jollakin useista mahdollisista tavoista verkoksi, ja jokaisella neuronien välisellä yhteydellä on paino, jota kutsutaan myös synaptiseksi painoksi; nämä painot antavat neuroverkolle sen kyvyn oppia ja muistaa oppimansa. Neuroverkon arkkitehtuurilla tarkoitetaan sen neuronien topologista järjestystä, johon kuuluu neuronien lukumäärä, neuroneista muodostettujen kerrosten määrä ja rakenne sekä verkossa kulkevien signaalien suunta, ja se voi olla hyvinkin erilainen riippuen neuroverkon tyypistä. Arkkitehtuurit vaihtelevat erilaisista eteenpäin syöttävistä eli myötäkytketyistä verkoista takaisinkytkettyihin rakenteisiin. [6]

Neuroverkot ovat osa ohjelmoimattomia, ilman algoritmeja toimivia sopeutuvia tiedonkäsittelyjärjestelmiä tutkivaa insinööritaidon alaa. Tällaiset järjestelmät reagoivat ympäristöönsä kehittämällä yhteyksiä eri objektien välille, eli toisin sanoen ne oppivat esimerkeistä. Neuroverkot ovat juuri tällaisia järjestelmiä; ne ovat aivojen toiminnan kaltaiseen tiedon muistiin painamiseen ja prosessointiin perustuvia, rinnakkaislaskentaa (engl. parallel computing) voimakkaasti hyödyntäviä tietokonearkkitehtuureja, jotka vahvistavat sisäisiä laskentarakenteitaan etukäteen suunnitellun ja ohjelmoidun algoritmin seuraamisen sijaan kokemuksen perusteella. Niiden kykyihin kuuluu paljon aivojen toiminnan kaltaista käytöstä, kuten oppimista, assosiaatiota, luokittelua, yleistämistä, piirteiden erottamista, optimointia ja sopeutumista. [6]

Neuroverkkojen tärkeimmät ominaisuudet ovat kokemuksen perusteella sopeutuminen, kyky oppia, kyky yleistää, tiedon jäsentely, vikasietoisuus, tiedon hajautettu varastointi (engl. distributed storage) ja helpotettu prototyypin kehittäminen. Kokemuksen perusteella sopeutumisella tarkoitetaan neuroverkon kykyä muuttaa sen sisäisiä parametreja, eli yleensä sen synaptisia painoja, syötteenä saatujen esimerkkien pohjalta. Neuroverkon syötteenään saamat esimerkit kuvaavat mallinnettavan prosessin käytöstä, ja jotta

neuroverkko tunnistaisi ne, niiden on oltava numeerisia. Esimerkit esitetään numeroina vektorimuodossa, johon kaikki todellisesta maailmasta saatu tieto, kuten kuvat, äänet, teksti tai aikasarjat, on muokattava jotta neuroverkko voisi käsitellä sitä. [10] [2] Näin neuroverkko pystyy saamaan tietoa kokemuksen kautta. Oppimiskyvyllä tarkoitetaan neuroverkon kykyä löytää mallinnettavan prosessin muuttujien väliset yhteydet jonkin oppimisalgoritmin avulla. Kun neuroverkko on oppinut sille annettujen syötteiden ja ulostulojen väliset suhteet, se voi yleistää oppimaansa. Tämä tarkoittaa, että neuroverkko pystyy antamaan mille tahansa annetulle syötteelle sitä vastaavan, lähellä toivottua tai odotettua ulostuloa olevan ulostulon. Näin neuroverkon avulla voidaan arvioida ulostuloja myös sellaisille syötteille, joiden ulostuloja ei vielä tiedetä. Tiedon jäsentelyllä taas tarkoitetaan neuroverkon kykyä järjestellä tiettyyn prosessiin liittyvää tietoa ja näin klusteroida samankaltaisia syötteitä. Vikasietoisuus ja tiedon hajautettu varastointi liittyvät neuroverkon neuronien välisien yhteyksien ominaisuuksiin: Neuroverkot ovat vikasietoisia (engl. fault-tolerant), koska neuronien välisten yhteyksien suuren määrän ansiosta ne pystyvät jatkamaan toimintaansa vaikka osa niiden rakenteesta olisi vioittunut, ja neuroverkon tietyn prosessin toimintaan liittyvän tiedon hajautettu varastointi jokaiseen suuresta määrästä neuronien välisiä yhteyksiä parantaa neuroverkon häiriönsietokykyä, mikäli joitakin neuroneita menetetään. Helpotetulla prototyypin kehittämisellä taas tarkoitetaan juuri neuroverkkojen prototyyppien kehittämisen helppoutta: riippuen niiden nimenomaisen sovelluskohteen erityispiirteistä, useimmista neuroverkkorakenteista voidaan luoda prototyyppejä helposti ohjelmistojen (engl. software) tai laitteistojen (engl. hardware) avulla, sillä niiden ulostulot saadaan koulutusprosessin jälkeen yleensä käyttäen joitakin perustavanlaatuisia matemaattisia operaatioita. [2] Näiden ominaisuuksien lisäksi neuroverkkojen on osoitettu pystyvän approksimoimaan mitä tahansa kompaktissa joukossa määriteltyä epälineaarista funktiota halutulla tarkkuudella. [4]

Neuroverkkomallin suunnittelu ja toteutus sisältää viisi pääasiallista vaihetta, jotka ovat datan kerääminen, piirteiden erottaminen, neuroverkon arkkitehtuurin valitseminen, neuroverkon kouluttaminen ja neuroverkon testaus ja arviointi. Neuroverkkoa varten on kerättävä asianmukaista kouluttamiseen

ja testaamiseen tarvittavaa dataa, ja raaka data on käsiteltävä ennen käyttöä. Tämän jälkeen neuroverkon suunnittelija suorittaa piirteiden erottamisen eli valitsee tietämyksensä ja kokemuksensa avulla ne piirteet, joilla on tai joilla uskotaan olevan jokin vaikutus mallinnettaviin arvoihin. Kun piirteiden erottaminen ja neuroverkon arkkitehtuurin valitseminen on tehty, neuroverkko voidaan kouluttaa, koulutettua neuroverkkoa voidaan testata ja sen kykyä yleistää oppimaansa voidaan arvioida kerätyn datan avulla. Riippuen saaduista tuloksista, on usein tarpeen toistaa koko prosessi tai osia siitä useita kertoja ja valita koulutetuista neuroverkoista se, joka suoriutuu parhaiten. [1] [6]

Neuroverkoilla on paljon eri käyttötarkoituksia. Luokittelun, kuvioiden tunnistamisen ja regression lisäksi niihin kuuluvat muun muassa todennäköisyysjakauman tiheysfunktion estimointi, yhteen sopivien kuvioiden tunnistaminen, useamman muuttujan funktioiden approksimoiminen, aikasarja-analyysi, epälineaarinen signaalinkäsittely, epälineaaristen systeemien mallintaminen, käänteinen mallintaminen (engl. inverse modelling), optimointi ja älykäs signaalinkäsittely. [6] Lisäksi neuroverkkopohjaiset ohjausjärjestelmät ovat herättäneet viime vuosina kiinnostusta, sillä moniin ohjausjärjestelmiin kuuluu tiettyä tuntematonta epälineaarisuutta, jonka käsittelyyn neuroverkot soveltuvat hyvin. [4] Ongelmat joiden ratkaisemiseen neuroverkot soveltuvat ovatkin luonteeltaan yleensä epälineaarisia tai dimensioltaan suuria, niihin sisältyy meluisaa (engl. noisy), monimutkaista, epätarkkaa, epätäydellistä tai virheille altista dataa tai huonosti ymmärrettyjä fysikaalisia tai tilastollisia malleja, tai niille ei tunneta selkeää matemaattista ratkaisua tai algoritmia. Parhaiten neuroverkoille sopivat ongelmat, joihin liittyvä tieto on epätarkkaa tai joihin liittyville prosesseille ei ole olemassa asianmukaista mallia, mutta joiden toiminnasta on saatavilla riittävästi edustavaa dataa. [6]

Suurin osa neuroverkkojen sovelluksista on musta laatikko -malleja (engl. black-box models). Tällaisten mallien tavoitteena on löytää syötteiden ja niitä vastaavien ulostulojen välille matemaattinen lauseke, jolla voidaan ennustaa annetun syötteen ulostulon arvo. [1] Monien informaatiota eteenpäin syöttävien neuroverkkojen voidaankin nähdä olevan musta laatikko -malleja:

ne vastaanottavat ja käsittelevät niille annettuja syötteitä ja tuottavat syötteille ulostuloja jonkin epälineaarisen funktion mukaisesti. [6] Musta laatikko -mallit ovat hyvin alkeellisia matemaattisia malleja: niiden toiminta perustuu vain havaintoihin, ja vaikka ne saattavat pystyä ennustamaan arvoja syötteille, ne eivät pysty selittämään syötteiden ja ulostulojen suhdetta. [1] Kaikki neuroverkkomallit eivät kuitenkaan ole musta laatikko -malleja. [6]

Ainakin laskennallisesti, tavanomainen digitaalinen tietokone pystyy tekemään kaiken sen mitä neuroverkotkin. Tämän takia onkin tärkeää kysyä, mitä hyötyä neuroverkoista on verrattuna tavallisiin tietokoneisiin. Vastaus tähän kysymykseen löytyy kahdesta neuroverkkojen merkittävästä ominaisuudesta, joista ensimmäinen on niiden laskennallinen yksinkertaisuus ja itseorganisoituminen (engl. self-organization). Neuroverkot ja niiden biologisia neuroverkkoja jäljittelevä rakenne ovat itse asiassa ennennäkemätön tietokonearkkitehtuuri, jonka avulla voidaan ratkaista monimutkaisia ja matemaattisesti huonosti määriteltyjä, epälineaarisia, epästationaarisia tai stokastisia ongelmia käyttämällä hyvin yksinkertaisia matemaattisia operaatioita. Neuroverkkojen suunnittelussa tavoitellaan äärimmäistä yksinkertaisuutta ja itseorganisoituvuutta; Niiden taustalla oleva algoritminen rakenne on hyvin yksinkertainen, mutta se pystyy mukautumaan useiden erityyppisten ongelmien vaatimuksiin. Tavanomaiset algoritmit ja menetelmät käyttävät ratkaisuissaan monimutkaisia yhtälöryhmiä ja soveltuvat vain yhden tietyn ongelman ratkaisemiseen, kun taas neuroverkot ovat laskennallisesti ja algoritmisesti yksinkertaisia ja lisäksi ne soveltuvat itseorganisoituvuutensa ansiosta monenlaisten ongelmien ratkaisemiseen. [3]

Toinen neuroverkkojen tavallisiin tietokoneisiin verrattuna edullinen ominaisuus on niiden erittäin rinnakkainen rakenne. Tavanomaiset digitaaliset tietokoneet toimivat peräkkäisesti, eli ne tekevät yhden asian loppuun ennen kuin aloittavat seuraavaa, ja jos yksi tietokoneen miljoonista transistoreista menee rikki, koko kone pysähtyy. Toisin kuin tietokoneet, biologiset neuroverkot ovat rakenteeltaan rinnakkaisia: aivojen toiminta ei häiriinny, vaikka aikuisen ihmisen keskushermostossa neuroneita kuolee vuosittain tuhansia. Myös keinotekoisten neuroverkkojen rakenne on tällä tavalla päällekkäinen. Tällä hetkellä suurinta osaa neuroverkoista simuloidaan kuiten-

kin tavanomaisilla digitaalisilla tietokoneilla, jolloin niiden kyky toimia komponenttien hajoamisesta huolimatta ei ole voimassa. Tämän sijaan neuroverkkoja voidaan kuitenkin simuloida yhä useammin saatavilla olevilla (neuroverkko)laitteistoilla joissa yksi mikropiirin siru voi sisältää jopa tuhansia neuroneita, jolloin niiden vikasietoisuus onkin voimassa. [3]

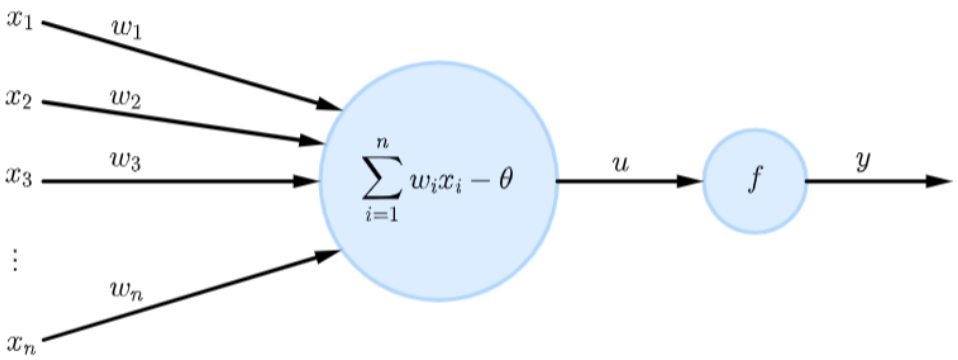
Neuroverkkoihin suuntautuvan kiinnostuksen ei pidä johtua vain niiden karkeasta tavasta jäljitellä aivojen toimintaa; jopa niiden itseorganisoituminen voidaan toteuttaa tavanomaisiin tietokoneisiin mutkikkaiden tekoälyalgoritmien avulla. Kuten aiemmin on mainittu, neuroverkkojen suurin etu on se, että ne mahdollistavat erilaisten monitahoisten ongelmien ratkaisemisen matalan tason ohjelmoinnilla ilman uudelleenohjelmointia tai muulla tavalla järjestelmään puuttumista. [3]

4.1 Rakenne

Neuroni on epälineaarinen parametrisoitu rajoitettu funktio. Sen muuttujia kutsutaan yleensä syötteiksi ja sen arvoa ulostuloksi. [1] Jokainen neuronimalli koostuu prosessointielementistä, syötteiden synaptisista yhteyksistä ja yhdestä ulostulosta. [4] Nimestään huolimatta neuroverkoissa käytetyt keinotekoiset neuronit eivät ole lähelläkään biologisia neuroneita. [6]

Neuroni yhdistää datasta saadut syötteet ja joukon kertoimia eli painoja, joista jokainen joko vahvistaa tai vaimentaa itseään vastaavaa syötettä. Näin painot määräävät kunkin syötteen merkittävyyden liittyen neuroverkon suoritettavana olevaan tehtävään; jos kyseessä on esimerkiksi luokittelu, painot vahvistavat niitä syötteitä, joista on eniten apua datan virheettömässä luokittelussa. Neuroni laskee syötteiden ja niitä vastaavien painojen tulojen summan ja antaa lasketun summan syötteenä neuronin aktivaatiofunktiolle. Aktivaatiofunktion laskeman arvon mukaan tehdään päätös siitä, pitäisikö kyseisen syötteen antaman signaalin edetä neuroverkossa ja vaikuttaa neuroverkon lopulliseen tulokseen. Jos signaalin annetaan edetä, sanotaan että kyseinen neuroni on aktivoitu. [10]

Neuroni koostuu seitsemästä perusosasta, jotka ovat syötteet, painot, lineaarinen summausfunktio, kynnysarvo, aktivaatiopotentiaali, aktivaatio-



Kuva 1: Neuronin rakenne.

funktio ja ulostulo. Syötteet $(x_1, x_2, \ldots, x_n)$ kuvaavat tietyn sovelluskohteen muuttujien arvoja. Neuronille syötteet ovat sen ulkoisesta ympäristöstä tulevia signaaleja tai esimerkkejä. Painot tai synaptiset painot $(w_1, w_2, \ldots, w_n)$ mahdollistavat syötteiden tärkeyden kvantifioimisen neuronin käyttötarkoituksen suhteen määräämällä tietyn painoarvon jokaiselle syötteelle, ja lineaarinen summausfunktio Σ kokoaa nämä painot ja niitä vastaavat syötteet. Kynnysarvo θ määrittää sopivan kynnyksen, jonka lineaarisen summausfunktion arvon täytyy ylittää, jotta neuroni aktivoituisi eli jotta syötteen antama signaali etenisi neuronin ulostuloon. Aktivaatiopotentiaali u on lineaarisen summausfunktion Σ ja kynnysarvon θ erotuksen arvo, eli neuroni aktivoituu, jos $u \geq 0$. Aktivaatiopotentiaali u annetaan syötteenä aktivaatiofunktiolle f, jonka tarkoitus on pitää neuronin ulostulon mahdolliset arvot kohtuullisten rajojen välillä. Neuronin ulostulo y on neuronin tuottama lopullinen, tiettyjä syötteitä vastaava arvo, jota voidaan halutessa käyttää myös neuroniin yhdistettyjen muiden neuronien syötteenä. Neuronin toiminta voidaan kuvata matemaattisesti seuraavien yhtälöiden avulla [2]:

$$\begin{cases} u = \sum_{i=1}^{n} w_i x_i - \theta \\ y = f(u) \end{cases}$$

Neuronin rakenne on näkyvillä kuvassa 1 [12].

Kaksi tai useampia neuroneita voidaan yhdistää kerrokseksi, ja yhdestä tai useammasta tällaisesta kerroksesta voidaan muodostaa neuroverkko. [4]

Neuroverkko on siis kompositio siihen kuuluvien neuroneiden epälineaarisista funktioista. [1] Neuroverkossa neuronit on yhdistetty toisiinsa, ja jokaisella neuronien välisellä yhteydellä on paino. [9] On kuitenkin huomattava, että vaikka neuroneiden sanotaan olevan yhdistettyjä toisiinsa, useimmissa sovelluksissa neuronit eivät ole fyysisiä objekteja eivätkä niiden väliset yhteydet ole todellisuudessa olemassa; elektronisen toteutuksen sijaan neuronien tekemät laskelmat toteutetaan ohjelmistoina. [1]

Neuroverkon rakenne voidaan jakaa kolmeen osaan: syötekerrokseen (engl. input layer), piilokerroksiin (engl. hidden layers) ja tuloskerrokseen (engl. output layer). Syötekerroksen tehtävä on vastaanottaa dataa, signaaleja tai piirteitä neuroverkon ulkopuoliselta ympäristöltä, ja usein myös normalisoida tai skaalata vastaanotetut syötteet ennen kuin ne siirtyvät neuroverkossa eteenpäin. [2] Normalisointia tai skaalaamista lukuunottamatta syötekerroksen neuronit eivät suorita laskelmia. [7] Piilokerroksien tehtävä on löytää rakenteita analysoitavasta datasta. Ne suorittavat suurimman osan neuroverkon sisäisistä laskelmista. [2] Näitä kerroksia kutsutaan piilokerroksiksi, koska niiden syötteet ja ulostulot ovat ikään kuin piilossa: niitä käytetään vain neuroverkon sisäisiin yhteyksiin eivätkä ne ole ulkomaailman saatavilla. [6] Neuroverkon piilokerrosten ja niihin kuuluvien neuronien määrä riippuu ratkaistavan ongelman tyypistä ja monimutkaisuudesta sekä siihen liittyvän saatavilla olevan datan määrästä ja laadusta. [2] Kuitenkin neuroverkon piilokerrosten ja niihin kuuluvien neuronien optimaalisen määrän löytämistä korostetaan usein liikaa: vaikka piilokerrosten neuronien sopivan määrän löytäminen on neuroverkon tarkkuuden kannalta tärkeää, neuroverkon syötteenä saaman datan esitysmuoto ja koulutusaineston muotoilu ovat neuroverkon tulosten tarkkuuden varmistamisessa usein huomattavasti tärkeämpiä. [7] Tuloskerroksen tehtävä on tuottaa ja esittää neuroverkon lopulliset ulostulot. Tämän kerroksen neuronit suorittavat neuroverkon viimeiset laskelmat: ne muodostavat neuroverkon tulokset aikaisempien kerrosten neuronien suorittamien laskelmien pohjalta, ja niiden ulostulot ovat koko neuroverkon ulostulot. [2] [1]

Neuroverkkoa, joka koostuu syötekerroksesta ja tuloskerroksesta, kutsutaan yksikerroksiseksi neuroverkoksi. Tällaisten neuroverkkojen pääasiallinen

heikkous on niiden kyvyttömyys luokitella muita kuin lineaarisesti separoituvia luokkia epälineaarisen aktivaatiofunktion käytöstä huolimatta. Tämä voidaan korjata lisäämällä neuroverkkoon piilokerroksia; neuroverkkoa, joka koostuu syötekerroksesta, tuloskerroksesta ja yhdestä tai useammasta syöte- ja tuloskerrosten välissä olevasta piilokerroksesta, kutsutaan monikerroksiseksi neuroverkoksi. [6] [5]

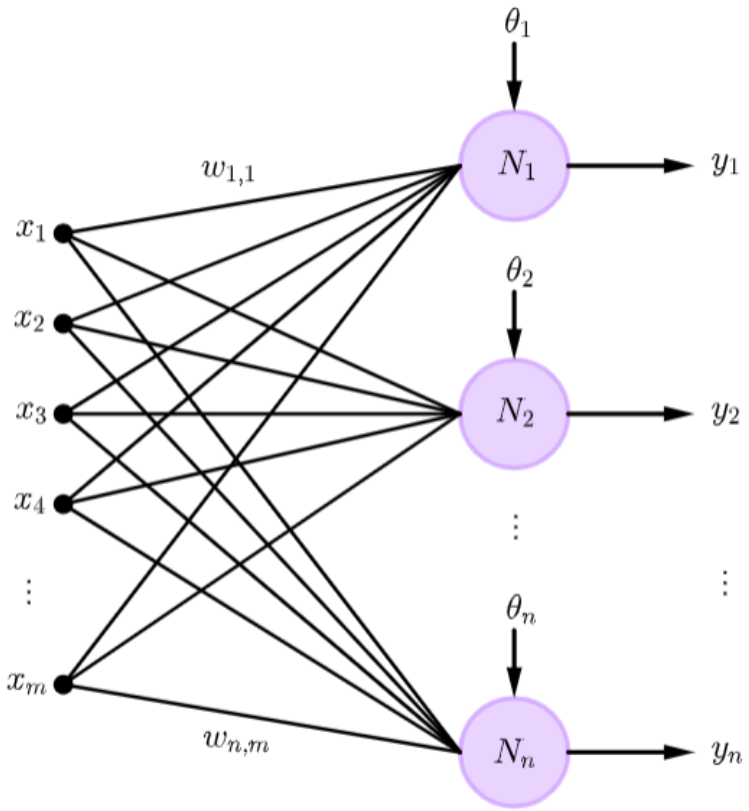
Neuroverkon arkkitehtuuri määrittelee millä tavalla neuroverkkoon kuuluvat neuronit on toisiinsa nähden järjestetty ja mihin suuntaan tieto liikkuu neuronien välisissä yhetyksissä, kun taas neuroverkon topologia määrittelee tietyn arkkitehtuurin omaavan neuroverkon rakenteellisen kokoonpanon. Saman arkkitehtuurin alle voi siis kuulua useampia eri topologioita, jotka koostuvat eri määristä neuroneita. Lisäksi samaan arkkitehtuuriin kuuluvissa topologioissa voidaan käyttää eri aktivaatiofunktioita. [2] Teoriassa eri aktivaatiofunktioiden käyttäminen on mahdollista myös saman neuroverkon eri kerroksissa tai jopa eri neuroneissa, mutta tavallisesti yhden neuroverkon kaikissa piilokerroksissa käytetään samaa aktivaatiofunktiota. [5]

Jokaisen neuroverkon perustana on myötäkytketty neuroverkko (engl. feedforward neural network). Myötäkytketyssä neuroverkossa informaatio kulkee neuroneiden välillä vain yhteen suuntaan eli eteenpäin, syötteistä kohti ulostuloja. Kuten muutkin neuroverkot, se on neuroniensa funktioiden yhdistelmänä saatu epälineaarinen funktio. Myötäkytketyt neuroverkot ovat staattisia malleja, eli niiden syötteiden ollessa vakioita myös niiden ulostulot ovat vakioita. [1]

Tarkastellaan yksikerroksisen myötäkytketyn neuroverkon rakennetta. Yhden piilokerroksen sisältävän myötäkytketyn neuroverkon rakennetta käsitellään monikerroksisten perseptronien yhteydessä, ja kaksi tai useampia piilokerroksia sisältäviä neuroverkkoja käsitellään syväoppivat neuroverkot -osiossa. Yksikerroksisen myötäkytketyn neuroverkon syötteet ovat vektorimuodossa

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_m)^\top.$$

Syötteiden vektori on pystyvektori, joka on tässä kirjoitettu vaakavektorin transpoosina. Neuroverkon neuronien välisten yhteyksien painot määrittelee



Kuva 2: Yksikerroksisen myötäkytketyn neuroverkon rakenne.

painomatriisi

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{bmatrix},$$

jossa paino $w_{i,j}$ tarkoittaa tuloskerroksen neuronin N_i ja syötteen x_j välisen yhteyden painoa. Neuroverkon kynnysarvot ovat vektorimuodossa

$$\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)^\top.$$

Kuten syötteiden yhteydessä, kynnysarvojen vektori on pystyvektori, joka on tässä kirjoitettu vaakavektorin transpoosina. Neuroverkon ulostulo vektori-

muodossa on

$$\boldsymbol{y} = f(W\boldsymbol{x} + \boldsymbol{\theta}) = (y_1, y_2, \ldots, y_n)^\top,$$

jossa W on edellä esitetty painojen matriisi, $\boldsymbol{x}$ on edellä esitetty syötteiden vektori, $\boldsymbol{\theta}$ on edellä esitetty kynnysarvojen vektori ja f on neuroverkon neuroneissa käytetty aktivaatiofunktio. [4] Tällainen neuroverkko, jossa on m syötekerroksen neuronia ja n tuloskerroksen neuronia, laskee n epälineaarisen funktion arvoa m:lle syötteelleen. [1] Yksikerroksisen myötäkytketyn neuroverkon rakenne on näkyvillä kuvassa 2 [12].

4.1.1 Aktivaatiofunktio

Neuroverkoissa voidaan käyttää useita erilaisia aktivaatiofunktioita, jotka rajoittavat neuroverkon neuroneiden ulostulot tietylle välille. Valinta erilaisten aktivaatiofunktioiden välillä tehdään sovelluskohteen mukaan. [4] [6] Tässä osiossa esitellään seitsemän erilaista aktivaatiofunktiota.

Lineaarinen funktio on muotoa

$$f(u) = u.$$

Sen arvo on siis sama kuin aktivaatiopotentiaalin arvo. Lineaarista funktiota voidaan käyttää aktivaatiofunktiona esimerkiksi neuroverkoissa, joiden tehtävänä on funktion approksimointi eli käyrän sovitus.

Porrasfunktio (engl. step function) on muotoa

$$f(u) = \begin{cases} 1, & \text{jos } \ u \geq 0 \\ 0, & \text{jos } \ u < 0. \end{cases}$$

Jos aktivaatiopotentiaali u on suurempi tai yhtä suuri kuin nolla, porrasfunktio saa arvon yksi, ja muussa tapauksessa se saa arvon nolla.

Kaksisuuntainen porrasfunktio (engl. bipolar step function) on muotoa

$$f(u) = \begin{cases} 1, & \text{jos } \ u > 0 \\ 0, & \text{jos } \ u = 0 \\ -1, & \text{jos } \ u < 0. \end{cases}$$

Se saa siis yhden arvoista 1, 0 tai -1 riippuen siitä, onko aktivaatiopotentiaali u nolla, suurempi kuin nolla, vai pienempi kuin nolla.

Symmetrinen ramppifunktio (engl. symmetric ramp function) on muotoa

$$f(u) = \begin{cases} a, & \text{jos } u > a \\ u, & \text{jos } -a \leq u \leq a \\ -a, & \text{jos } u < a. \end{cases}$$

Kun aktivaatiopotentiaali u kuuluu määritellylle välille $[-a, a]$ tämän funktion arvo on aktivaatiopotentiaalin arvo. Muussa tapauksessa funktion arvot rajoittuvat määritellyn välin päätearvoihin.

Logistinen funktio (engl. logistic function) on muotoa

$$f(u) = \frac{1}{1 + e^{-\beta u}},$$

missä β on reaaliarvoinen vakio. Tämän funktion arvot ovat aina reaalilukuja väliltä $(0, 1)$.

Hyperbolinen tangenttifunktio (engl. hyperbolic tangent function) on muotoa

$$f(u) = \frac{1 - e^{-\beta u}}{1 + e^{-\beta u}},$$

missä β on reaaliarvoinen vakio. Tämän funktion arvot ovat aina reaalilukuja väliltä $(-1, 1)$.

Gaussin funktio (engl. Gaussian function) on muotoa

$$f(u) = e^{-\frac{(u-c)^2}{2\sigma^2}},$$

missä parametri c määrittelee funktion keskustan eli keskiarvon ja σ määrittelee keskihajonnan. Tämän funktion arvot ovat samat aktivaatiopotentiaaleille, jotka ovat yhtä kaukana funktion keskiarvosta. [2]

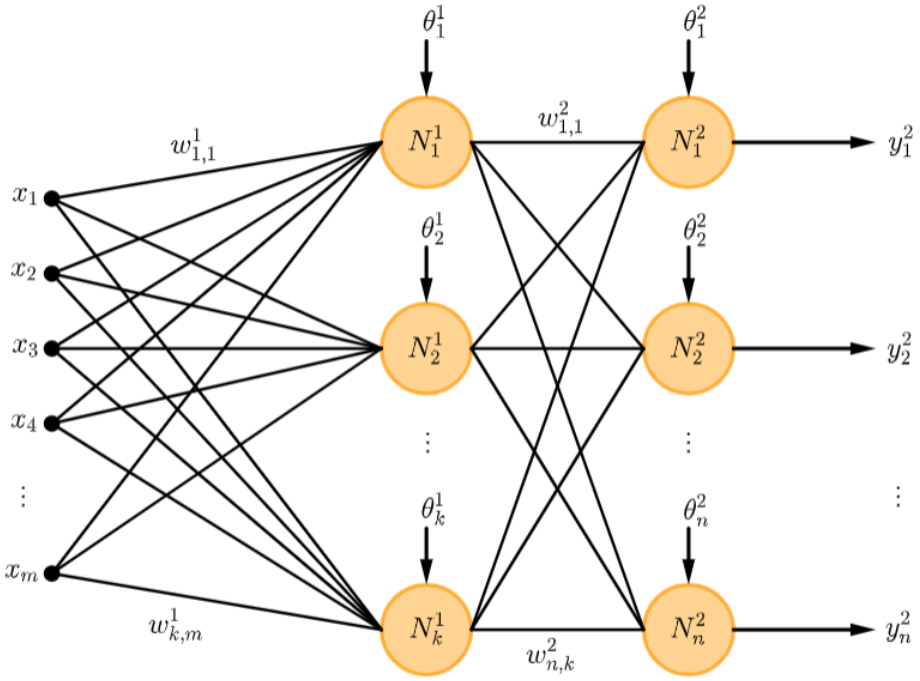
4.2 Erilaisia neuroverkkorakenteita

Tässä osiossa käsitellään kolmea erilaista neuroverkkorakennetta.

4.2.1 Monikerroksinen perseptroni

Monikerroksinen perseptroni (engl. multilayer Perceptron, lyh. MLP) on yksi pääasiallisista monikerroksista myötäkytkettyä neuroverkkoarkkitehtuuria käyttävistä neuroverkkorakenteista. Monikerroksinen perseptroni on aina

kaksi- tai useampikerroksinen: se koostuu syötekerroksesta, tuloskerroksesta ja vähintään yhdestä piilokerroksesta. Signaalit kulkevat monikerroksisessa perseptronissa sen arkkitehtuurin mukaisesti aina samaan suuntaan, syötekerroksesta tuloskerrokseen. [2] Sen toimintaa on helppo ymmärtää, ja se suoriutuu käytännön ongelmien ratkaisemisesta yleensä hyvin. Näistä syistä monikerroksinen perseptroni onkin yleisin ja suosituin nykyisin käytössä oleva neuroverkkorakenne. [6]



Kuva 3: Yhden piilokerroksen sisältävän monikerroksisen perseptronin rakenne. Piilokerroksen ulostuloja ei ole selkeyden vuoksi merkitty kuvaan.

Tarkastellaan yhden piilokerroksen sisältävän monikerroksisen perseptronin rakennetta. Oletetaan tarkasteltavassa neuroverkossa olevan$m$ syötettä, k piilokerroksen neuronia ja n tuloskerroksen neuronia. Jokaisella neuroverkon kerroksella lukuunottamatta syötekerrosta on oma painomatriisi W, kynnysarvojen vektori $\boldsymbol{\theta}$ ja ulostulojen vektori $\boldsymbol{y}$, ja lisäksi neuroverkolla on syöt-

teiden vektori $\boldsymbol{x}$. Neuroverkon syötteet ovat vektorimuodossa

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_m)^\top.$$

Syötekerroksen ja piilokerroksen välisten yhteyksien painot määrää painomatriisi

$$W^1 = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \cdots & w^1_{1,m} \\ w^1_{2,1} & w^1_{2,2} & \cdots & w^1_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ w^1_{k,1} & w^1_{k,2} & \cdots & w^1_{k,m} \end{bmatrix},$$

jossa paino $w^1_{i,j}$ tarkoittaa piilokerroksen neuronin N^1_i ja syötteen x_j välisen yhteyden painoa. Piilokerroksen ja tuloskerroksen välisten yhteyksien painot taas määrää painomatriisi

$$W^2 = \begin{bmatrix} w^2_{1,1} & w^2_{1,2} & \cdots & w^2_{1,k} \\ w^2_{2,1} & w^2_{2,2} & \cdots & w^2_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ w^2_{n,1} & w^2_{n,2} & \cdots & w^2_{n,k} \end{bmatrix},$$

jossa paino $w^2_{i,j}$ tarkoittaa tuloskerroksen neuronin N^2_i ja piilokerroksen neuronin N^1_j välisen yhteyden painoa. Piilokerroksen kynnysarvot ovat vektorimuodossa

$$\boldsymbol{\theta}^1 = (\theta^1_1, \theta^1_2, \ldots, \theta^1_k)^\top,$$

ja tuloskerroksen kynnysarvot ovat vektorimuodossa

$$\boldsymbol{\theta}^2 = (\theta^2_1, \theta^2_2, \ldots, \theta^2_n)^\top.$$

Piilokerroksen ulostulo vektorimuodossa on

$$\boldsymbol{y}^1 = f(W^1\boldsymbol{x} + \boldsymbol{\theta}^1) = (y^1_1, y^1_2, \ldots, y^1_k)^\top,$$

jossa W^1 on edellä esitetty painojen matriisi, $\boldsymbol{x}$ on edellä esitetty syötteiden vektori, $\boldsymbol{\theta}^1$ on edellä esitetty kynnysarvojen vektori ja f on neuroverkon neuroneissa käytetty aktivaatiofunktio. Tuloskerroksen ulostulo eli koko neuroverkon ulostulo vektorimuodossa on

$$\boldsymbol{y}^2 = f(W^2\boldsymbol{y}^1 + \boldsymbol{\theta}^2) = (y^2_1, y^2_2, \ldots, y^2_n)^\top,$$

jossa W^2 on edellä esitetty painojen matriisi, $\boldsymbol{y}^1$ on piilokerroksen ulostulojen vektori, $\boldsymbol{\theta}^2$ on edellä esitetty kynnysarvojen vektori ja f on neuroverkon neuroneissa käytetty aktivaatiofunktio. [4] Yhden piilokerroksen sisältävän monikerroksisen perseptronin rakenne on näkyvillä kuvassa 3 [12].

Monikerroksisilla perseptroneilla on laaja joukko sovelluskohteita, ja niitä pidetäänkin soveltamisen kannalta yhtenä monipuolisimmista neuroverkkorakenteista. Niiden sovelluksia käytetään useilla eri tieteenaloilla, joihin kuuluvat esimerkiksi lääketiede, biologia, kemia, fysiikka, taloustiede, geologia, ympäristötiede, psykologia ja useat eri insinööritaidon alat, ja näistä sovelluksista tärkeimpiin kuuluvat funktion approksimointi eli käyrän sovitus, hahmontunnistus (engl. pattern recognition), prosessinohjaus (engl. process control), aikasarjan ennustaminen ja systeemin optimointi. [2] Monipuolisuutensa lisäksi monikerroksisilla perseptroneilla on myös haittapuolia: niitä voidaan kouluttaa vain valvotun oppimisen avulla, kouluttamiseen tarvitaan runsaasti esimerkkejä, ja se voi olla hidasta ja huonosti käyttäytyvää. Kouluttamisen nopeutta on kuitenkin onnistuttu parantamaan uusien oppimisalgoritmien avulla ja rajoittamalla joidenkin painojen arvoja koulutuksen aikana. [6]

Yhden piilokerroksen sisältävän monikerroksisen perseptronin on osoitettu pystyvän approksimoimaan mitä tahansa realistista funktiota mielivaltaisella tarkkuudella, jos sen painot ja piilokerrosten neuronien määrä on valittu oikein. Tämä tarkoittaa, että kaksikerroksista perseptronia voidaan periaatteessa käyttää lähes minkä tahansa luokitteluongelman ratkaisemiseen. Kuitenkaan oikeaa piilokerrosten neuronien määrää ja oikeita painoja ei tiedetä; ratkaisu on siis olemassa, mutta siitä että se koskaan löydettäisiin ei ole takuuta. [9]

4.2.2 Takaisinkytketty neuroverkko

Takaisinkytketyissä neuroverkoissa (engl. recurrent neural networks) neuroneiden ulostuloja käytetään aikaisempien kerrosten neuronien syötteinä. Tähän arkkitehtuuriin kuuluvat neuroverkot ottavat tämänhetkisten ulostulojen tuottamisessa huomioon myös edelliset ulostulot; toisin kuin staattiset myö-

täkytketyt neuroverkot, takaisinkytketyt neuroverkot ovat dynaamisia malleja. Tästä syystä niille sopivia sovelluskohteita ovat ajan mukaan muuttuvat systeemit, ja niitä voidaankin siis käyttää esimerkiksi aikasarjojen ennustamiseen. [2] [1]

Takaisinkytketty neuroverkko voidaan muodostaa myötäkytketystä neuroverkosta yhdistämällä myötäkytketyn neuroverkon tuloskerroksen ulostulot sen syötekerrokseen siten, että tuloskerroksen ulostuloja käytetään neuroverkon syötteinä seuraavia ulostuloja muodostettaessa. [4] Jokainen takaisinkytketty neuroverkko voidaankin esittää kanonisessa muodossa, joka koostuu myötäkytketystä neuroverkosta, jonka ulostulot annetaan sille takaisin syötteinä. [1] On myös mahdollista muodostaa takaisinkytketty neuroverkko, jossa jokaisen neuroverkon kerroksen ulostulot annetaan syötteinä edelliselle kerrokselle; tällaista neuroverkkoa kutsutaan kokonaan takaisinkytketyksi neuroverkoksi (engl. fully recurrent neural network). [3] Takaisinkytkettyjen neuroverkkojen palautesilmukan muodostamisen tavoitteena on mahdollistaa neuroverkon ulostulojen säätely ulostulojen avulla, mikä on erityisen mielekästä jos tämänhetkinen ulostulo $O(t)$ vaikuttaa seuraavaan ulostuloon $O(t + \Delta)$. Tällä hetkellä tutkimusyhteisössä on laajasti käytössä kolme erilaista takaisinkytkettyä neuroverkkorakennetta, jotka ovat Elmanin verkko (engl. Elman network), Jordanin verkko (engl. Jordan network) ja Hopfieldin verkko (engl. Hopfield network). [4]

4.2.3 Itseorganisoituva kartta

Itseorganisoituvat kartat (engl. self-organizing maps), joita kutsutaan kehittäjänsä mukaan myös Kohosen kartoiksi (engl. Kohonen maps), ovat aivokuoren toiminnasta innoituksensa saanut neuroverkkorakenne. Niissä käytetään valvomattomaan oppimiseen kuuluvaa kilpailullista kouluttamisprosessia, jossa neuronit kilpailevat keskenään oikeudesta muuttaa painojaan. Itseorganisoituvat kartat ovat itseorganisoitumisen suhteen yksi laajimmalle levinneistä neuroverkkorakenteista. Vaikka niillä on moninaisia sovelluksia eri tieteenaloilla, niiden tutkituimmat sovelluskohteet ovat kuvioiden luokittelu ja klusterointi, joiden suorittamiseen ne soveltuvat yksinkertaisen rakenteen-

sa ja hienostuneen kouluttamisprosessinsa ansiosta hyvin. [2]

Itseorganisoituva kartta koostuu syötekerroksesta ja kilpailullisesta kerroksesta. Se kuvaa syötteenä saamansa avaruudellisesti toisiaan lähellä olevat kuviot neuroneihin, jotka ovat myös avaruudellisesti lähellä. Näin saadaan syötteiden topografinen kartta, joka näyttää neuroverkon syötteinään saamien kuvioiden väliset luonnolliset suhteet. Neuroverkolle syötteenä annettava kuvio on vektorimuodossa $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$. Tämän syötteen ja yhden kilpailullisen kerroksen neuronin väliset yhteydet ovat vektorimuodossa $\boldsymbol{w}_{ji} = (w_{j1}, w_{j2}, \ldots, w_{jn})$, jossa j on kilpailullisen kerroksen neuronin numero ja i on syötekerroksen neuronin numero. Syötteenä saadun kuvion etäisyys jokaisesta kilpailullisen kerroksen neuronin painojen vektorista lasketaan jonkin metriikan mukaan. Yleensä käytetään euklidinista etäisyyttä. Voittaja on se kilpailullisen kerroksen neuroni, jolle laskettu arvo on pienin; sen painot ovat lähimpänä syötteenä saadun kuvion vastaavia arvoja. Kun kilpailun voittava neuroni on löydetty, on löydettävä kyseisen neuronin naapurusto, jonka koko ja muoto vaihtelevat. Naapuruston löydyttyä voittavan neuronin ja sen naapurustoon kuuluvien neuronien painot päivitetään. Vain nämä voittavat neuronit saavat muuttaa painojaan; muiden kilpailullisen kerroksen neuronien painot eivät muutu. Mikäli sama tai samankaltainen kuvio annetaan syötteenä neuroverkolle myöhemmin, painonsa päivittäneet neuronit voittavat kyseisen kuvion kohdalla muita neuroneja todennäköisemmin. [7]

4.3 Kouluttaminen

Kun halutaan rakentaa käyttökelpoinen oppiva kone, pelkkä neuroverkkorakenne ja sen osat eivät yksin riitä; tarvitaan lisäksi automatisoitu oppimismenetelmä. Tämän oppimismenetelmän tarkoituksena on arvioimalla neuroverkon tuottamaa virhettä sekä oppimisen aikana että sen jälkeen määrittää neuroverkon painojen arvot siten, että neuroverkko pystyy lopulta tuottamaan asianmukaisen ratkaisun. [6] Tätä oppimisprosessia kutsutaan neuroverkon kouluttamiseksi.

Neuroverkon kouluttaminen on algoritminen menetelmä, jonka avulla

neuroverkon parametreja eli sen painoja ja kynnysarvoja muutetaan siten, että neuroverkko pystyy ratkaisemaan sille annetun ongelman mahdollisimman hyvin. [1] Kouluttamisessa neuroverkko siis pakotetaan tuottamaan tietty vastaus reaktiona tiettyyn syötteeseen. [4] Tätä proseduuria kutsutaan myös oppimiseksi, ja se voidaan määritellä prosessina, jossa neuroverkko muuttaa itseään ulkoisten ärsykkeiden perusteella siten, että se pystyy lopulta yleistämään oppimaansa ja tuottamaan haluttuja vastauksia. [6] Kouluttamisen tavoite on siis tuottaa rajoitetun koulutusaineiston avulla neuroverkko, joka osaa käsitellä myös sellaista dataa, jota sen kouluttamisessa ei ole käytetty. [7]

Neuroverkon kouluttaminen suoritetaan iteroimalla; kun verkon painoja on muutettu, syötteet esitetään neuroverkolle uudestaan ja koko painojen muuttamisen prosessi toistetaan. Yksi iteraatio muodostuu syklistä, johon kuuluu syötteen antaminen neuroverkolle, ulostulon laskeminen syötteelle, virheen laskeminen ja neuroverkon painojen muuttaminen. [7] Tyypillisesti neuroverkon kouluttamiseen vaaditaan kymmeniä tai satoja iteraatioita, joiden suorittaminen saattaa olla hidasta; järjestelmien prototyypeillä kouluttamiseen saattaa riittämättömän laskentatehon takia kulua päiviä tai jopa viikkoja. [1] [4] Kouluttamiseen käytettävät algoritmit ovat kuitenkin kehittyneet suunnattomasti: vielä 1990-luvun alussa kouluttamiseen tarvittiin usein kymmeniä tai satoja tuhansia iteraatioita, joiden suorittamiseen kului tehokkailtakin tietokoneilta päiviä. [1]

Neuroverkon kouluttamiseen käytetään algoritmia, joka määrittelee millä tavalla neuroverkon painoja muutetaan kouluttamisen aikana. Painojen lopulliset vakaat arvot määrittelevät neuroverkon "ohjelmoinnin". [6] Nämä painot eivät ole neuroverkolle annetun ongelman ratkaisun suhteen yksikäsitteiset: monet painojen arvojen eri yhdistelmät voivat johtaa kaiken kaikkiaan samaan ratkaisuun. [7] Joissakin neuroverkoissa painot eivät koskaan lähesty vakaita arvoja, vaan niiden arvot vaihtelevat käytön aikana jatkuvasti, jolloin neuroverkon "ohjelmoinnin"määrittelevät painojen arvot ja niiden dynamiikka. Neuroverkkoja voidaan opettaa ja niiden painoja muuttaa neuroverkon rakenteesta ja tyypistä riippuen monilla eri tavoilla. [6]

Online- ja offline-oppiminen ovat neuroverkkojen koulutuksessa käytettä-

viä menetelmiä, jotka määrittelevät kuinka monta esimerkkiä neuroverkolle syötetään ennen painojen päivittämistä. Online-oppimisessa neuroverkon painoja muutetaan jokaisen koulutusaineistoon kuuluvan esimerkin esittämisen jälkeen. Offline- eli eräoppimisessa taas neuroverkon painoja muutetaan vasta, kun kaikki koulutusaineistoon kuuluvat esimerkit on esitetty neuroverkolle kerran. Jokaisen koulutusaineiston esimerkin esittämistä kerran kutsutaan kierrokseksi; offline-oppimista käyttävissä verkoissa neuroverkon tekemät virheet summataan yhden kierroksen ajalta ja koko kierroksen aikana tehdyt virheet otetaan huomioon aina, kun verkon painoja säädetään. Kouluttamisessa vaaditaan siis aina vähintään yksi kierros, jotta koulutettavan neuroverkon painoja voidaan muuttaa. Tämän takia kaikkien koulutusaineistoon kuuluvien esimerkkien on oltava saatavilla koko koulutusprosessin ajan. Käytännössä offline-oppiminen ei yleensä ole online-oppimista edullisempaa. [2] [7]

Neuroverkkojen kouluttamisessa voidaan käyttää valvottua oppimista, valvomatonta oppimista tai vahvistusoppimista. Valvotussa oppimisessa neuroverkko oppii sille esitetyistä esimerkeistä ja niille halutuista ulostuloista. Jokaisen esimerkin tai kierroksen kohdalla lasketaan halutun ulostulon ja neuroverkon antaman ulostulon välinen virhe, ja laskettua virhettä käytetään neuroverkon painojen päivittämiseen. Kouluttamisprosessin edetessä virhe pienenee vähitellen, kunnes se saavuttaa joko minimiarvonsa tai ainakin hyväksyttävän pienen arvon.

Valvomattomassa oppimisessa koulutettavalle neuroverkolle annetaan syötteinä vain esimerkkejä ilman niiden toivottuja ulostuloja. Neuroverkko muodostaa saamistaan esimerkeistä ryhmiä tai klustereita siten, että samaan ryhmään tai klusteriin kuuluvat esimerkit ovat jollakin tavalla mitattuna samankaltaisia. Joskus muodostetut klusterit voidaan nähdä ratkaistavan ongelman ja siihen liittyvän datan kontekstissa kategorioina, jolloin koulutettua neuroverkkoa voidaan käyttää tuntemattomien syötteiden luokittelemiseen samaan tapaan kuin valvotun oppimisen avulla koulutettuja neuroverkkoja.

Vahvistusoppimisessa neuroverkon tekemän virheen laskemisen sijasta neuroverkolle kerrotaan jokaisen esimerkin kohdalla onko sen antama ulostulo hyväksytty vai hylätty. Jos ulostulo on hylätty, neuroverkko muuttaa paino-

jaan kunnes sen ulostulo hyväksytään tai kunnes ennalta määrätty yritysten määrä tulee täyteen, minkä jälkeen neuroverkko siirtyy käsittelemään seuraavaa esimerkkiä. Vahvistusoppiminen on hidasta ja tehotonta, mutta käyttökelpoista ja hyödyllistä sellaisten monimutkaisten neuroverkkojen kouluttamisessa, joiden valvotun oppimisen yhteydessä tarvittavien derivaattojen laskeminen on vaikeaa. [6]

4.3.1 Gradientin laskeutuminen

Neuroverkon kouluttamisessa tarkoituksena on minimoida neuroverkon tekemä virhe eli löytää neuroverkolle sellaiset painot, joilla neuroverkon antamien ulostulojen ja toivottujen ulostulojen välinen ero on mahdollisimman pieni. Kouluttaminen on näin ollen optimointiongelma. Tarkastellaan neuroverkkoa, jolla on vain yksi ulostulo $f(\boldsymbol{x}, \boldsymbol{w})$ ja jonka koulutusaineistoon kuuluu N esimerkkiä. Minimoitava virhefunktio voidaan valita neuroverkolle annetun ongelman mukaan: regression tapauksessa voidaan käyttää pienimmän neliösumman virhefunktiota (engl. least squares cost function)

$$J(\boldsymbol{w}) = \frac{1}{2} \sum_{k=1}^{N} [y_p^k - f(\boldsymbol{x}^k, \boldsymbol{w})]^2,$$

missä $\boldsymbol{x}^k$ on esimerkin k piirrevektori, y_p^k on esimerkin k haluttu ulostulo, $\boldsymbol{w}$ on neuroverkon painojen vektori ja $f(\boldsymbol{x}^k, \boldsymbol{w})$ on neuroverkon ulostulo piirrevektorille $\boldsymbol{x}^k$ painoilla $\boldsymbol{w}$, kun taas luokittelun tapauksessa on luontevampaa käyttää risti-entropia virhefunktiota

$$E = -\sum_{k} \sum_{i=1}^{C} \gamma_i^k \log\left(\frac{f_i(\boldsymbol{x}^k)}{\gamma_i^k}\right) + (1 - \gamma_i^k) \log\left(\frac{1 - f_i(\boldsymbol{x}^k)}{1 - \gamma_i^k}\right),$$

missä $\boldsymbol{x}^k$ on esimerkin k piirrevektori, γ_i^k on tuloskerroksen neuronin i haluttu ulostulo, nolla tai yksi, esimerkille k ja $f_i(\boldsymbol{x}^k)$ on tuloskerroksen neuronin i ulostulo. Käsittelyä jatketaan tästä eteenpäin käyttäen pienimmän neliösumman virhefunktiota $J(\boldsymbol{w})$, mutta käsiteltävät menetelmät soveltuisivat yhtä hyvin myös risti-entropia virhefunktiolle E. Neuroverkon kouluttamisen tavoitteena on nyt siis löytää painot $\boldsymbol{w}$ joilla virhefunktion $J(\boldsymbol{w})$ arvo

on mahdollisimman pieni. Kun käsiteltävänä on malli joka ei ole parametriensa suhteen lineaarinen, esimerkiksi myötäkytketty neuroverkko, ratkaistava minimointiongelma on usean muuttujan epälineaarinen optimointitehtävä. Tällaisessa tapauksessa optimointiongelman ratkaisemiseen käytetään gradienttimenetelmiä, joiden käyttö perustuu virhefunktion gradientin laskemiseen mallin parametrien suhteen jokaisen iteraation kohdalla. Laskettua gradienttia käytetään sitten edellisen iteraation painojen päivittämiseen. [1]

Gradientin laskeutumisen tarkoituksena on siis löytää kouluttamisessa käytetyn virhefunktion globaalia minimiä vastaavat painot. Tavoitteena on muuttaa painoja niin, että virhefunktion kuvaajalla "laskeudutaan"mahdollisimman jyrkästi, jolloin virhefunktion arvo pienenee; Tämän tavoitteen saavuttamiseen käytetään virhefunktion gradienttia. Vastavirta-algoritmi on tunnetuin tähän tarkoitukseen käytetyistä tekniikoista. [9]

4.3.2 Vastavirta-algoritmi

Vastavirta-algoritmi (engl. backpropagation algorithm) on suosittu ja laskennallisesti taloudellinen tapa laskea virhefunktion gradientti, jota tarvitaan usein neuroverkkojen kouluttamisessa. Se ei varsinaisesti ole oppimisalgoritmi, vaan tekniikka jota käytetään gradienttimenetelmiä käytettäessä. [1] Nimi "vastavirta"viittaa tapaan laskea virhefunktion gradientti rekursiivisesti tuloskerroksesta kohti syötekerrosta. [4] Tässä osiossa esitetään vastavirta-algoritmin käyttö myötäkytketyn neuroverkon kouluttamisessa.

Tämä matemaattinen tarkastelu on tehty lähteen [1] luvun 2.5.2.2 mukaisesti. Tarkastellaan myötäkytkettyä neuroverkkoa, jossa on piilokerroksen neuroneita, vain yksi tuloskerroksen neuroni ja ei kynnysarvoja. Yleistäminen tällaisesta neuroverkosta neuroverkkoon, jossa on useampia tuloskerroksen neuroneita, on suoraviivaista. Olkoon

$$y_i = g\left(\sum_{j=1}^{n_i} w_{ij}x_{ij}\right) = g(u_i)$$

neuronin i ulostulo, jossa g on neuronin aktivaatiofunktio, x_{ij} ovat neuronin i syötteet, w_{ij} ovat neuronin i painot ja u_i on neuronin i aktivaatiopotentiaali. Neuronin i n_i syötettä voivat olla joko neuroverkon syötteitä tai

muiden neuroneiden ulostuloja, joten merkintä x_j tarkoittaa tästä eteenpäin joko neuronin j ulostuloa y_j tai neuroverkon syötettä j. Virhefunktio jonka gradientti on laskettava on muotoa

$$J(\boldsymbol{w}) = \sum_{k=1}^{N} [y_p^k - f(\boldsymbol{x}^k, \boldsymbol{w})]^2 = \sum_{k=1}^{N} J^k(\boldsymbol{w}),$$

missä merkinnät ovat samat kuin aiemmin esitetyssä pienimmän neliösumman virhefunktiossa. Virhefunktion gradientin laskemiseksi voidaan laskea jokaiseen esimerkkiin k liittyvän funktion $J^k(\boldsymbol{w})$ gradientti ja summata lasketut gradientit.

Vastavirta-algoritmissa gradientin laskemiseen käytetään toistuvasti differentiaalilaskennan ketjusääntöä. Funktio $J^k(\boldsymbol{w}) = [y_p^k - f(\boldsymbol{x}^k, \boldsymbol{w})]^2$ on riippuvainen painoista w_{ij} vain tuloskerroksen neuronin i ulostulon kautta, ja kyseinen ulostulo taas on neuronin i aktivaatiopotentiaalin u_i funktio. Näin ollen ketjusäännön mukaan saadaan

$$\left(\frac{\partial J^k}{\partial w_{ij}}\right)_k = \left(\frac{\partial J^k}{\partial u_i}\right)_k \left(\frac{\partial u_i}{\partial w_{ij}}\right)_k = \delta_i^k x_j^k,$$

missä $\left(\frac{\partial J^k}{\partial u_i}\right)_k$ on funktion $J^k(\boldsymbol{w})$ gradientin arvo neuronin i aktivaatiopotentiaalin u_i suhteen kun neuroverkon syötteenä on esimerkki k, $\left(\frac{\partial u_i}{\partial w_{ij}}\right)_k$ on neuronin i aktivaatiopotentiaalin u_i osittaisderivaatta painon w_{ij} suhteen kun neuroverkon syötteenä on esimerkki k, ja x_j^k on neuronin i syötteen j arvo kun neuroverkon syötteenä on esimerkki k. Ongelmaksi jää siis laskea δ_i^k. Nämä arvot voidaan laskea verkon neuroneille rekursiivisesti tuloskerroksesta aloittaen.

Tuloskerroksen neuronille i pätee

$$\delta_i^k = \left(\frac{\partial J^k}{\partial u_i}\right)_k = \left(\frac{\partial}{\partial u_i}[(y_p^k - f(\boldsymbol{x}^k, \boldsymbol{w}))^2]\right)_k = -2f(\boldsymbol{x}^k, \boldsymbol{w}) \left(\frac{\partial f(\boldsymbol{x}^k, \boldsymbol{w})}{\partial u_i}\right)_k.$$

Koska neuroverkon ulostulo $f(\boldsymbol{x}^k, \boldsymbol{w})$ on tuloskerroksen neuronin ulostulo $y_i = g(u_i)$, lauseke voidaan kirjoittaa muodossa

$$\delta_i^k = -2f(\boldsymbol{x}^k, \boldsymbol{w}) \left(\frac{\partial g(u_i)}{\partial u_i}\right)_k = -2f(\boldsymbol{x}^k, \boldsymbol{w}) g'(u_i^k),$$

missä $g'(u_i^k)$ on tuloskerroksen neuronin aktivaatiofunktion derivaatta kun neuroverkon syötteenä on esimerkki k. Mallintamiseen suunnitelluissa myötäkytketyissä neuroverkoissa tuloskerroksessa käytetty aktivaatiofunktio on lineaarinen, jolloin lauseke sieventyy muotoon $\delta_i^k = -2f(\boldsymbol{x}^k, \boldsymbol{w})$.

Piilokerroksen neuronin i kohdalla virhefunktio on riippuvainen kyseisen neuronin aktivaatiopotentiaalista vain niiden neuronien m aktivaatiopotentiaalien kautta, jotka vastaanottavat neuronin i ulostulon. Näin ollen neuronille i pätee

$$\delta_i^k = \left(\frac{\partial J^k}{\partial u_i}\right)_k = \sum_m \left(\frac{\partial J^k}{\partial u_m}\right)_k \left(\frac{\partial u_m}{\partial u_i}\right)_k = \sum_m \delta_m^k \left(\frac{\partial u_m}{\partial u_i}\right)_k.$$

Lisäksi $u_m^k = \sum_i w_{mi} x_i^k = \sum_i w_{mi} g(u_i^k)$, joten

$$\left(\frac{\partial u_m}{\partial u_i}\right)_k = w_{mi} g'(u_i^k).$$

Näin ollen saadaan

$$\delta_i^k = \sum_m \delta_m^k w_{mi} g'(u_i^k) = g'(u_i^k) \sum_m \delta_m^k w_{mi}.$$

Nämä arvot voidaan siis laskea rekursiivisesti tuloskerroksesta lähtien. Näin ollen funktioiden $J^k(\boldsymbol{w})$ gradientit

$$\left(\frac{\partial J^k}{\partial w_{ij}}\right)_k = \delta_i^k x_j^k$$

voidaan laskea, ja virhefunktion gradientti on nyt

$$\left(\frac{\partial J}{\partial w_{ij}}\right) = \sum_k \left(\frac{\partial J^k}{\partial w_{ij}}\right)_k.$$

Neuroverkon painot voidaan kouluttamisen kierroksella i päivittää esimerkiksi seuraavasti:

$$\boldsymbol{w}(i) = \boldsymbol{w}(i-1) - \mu_i \nabla J(\boldsymbol{w}(i-1)),$$

missä $\mu_i > 0$ on oppimisnopeudeksi kutsuttu parametri. Tämä on vain yksi useista eri tavoista, joilla painot voidaan päivittää. [1]

Rumelhart, Hinton ja Williams kehittivät vastavirta-algoritmin vuonna 1986 monikerroksisten perseptronien kouluttamiseen, ja sen tarjoama täsmällinen tapa säätää neuroverkkojen piilokerroksien painoja kiihdytti neuroverkkojen kehitystä mahdollistamalla yksikerroksisten neuroverkkojen vaikeuksien voittamisen. [3] Vastavirta-algoritmista on kehitetty useita muunnoksia, kuten resilient propagation -algoritmi ja Levenberg-Marquardt -menetelmä. [2]

4.3.3 Koulutusaineisto, testausaineisto ja arviointiaineisto

Neuroverkon kouluttamista varten on kerättävä dataa. [1] Neuroverkot ratkaisevat ongelmia yleensä niille annetun datan perusteella, ja tästä syystä riittävän ja oleellisen datan keräämiseen ja käsittelyyn on neuroverkkosovelluksia käyettäessä kiinnitettävä erityistä huomiota. [6] Kerättyä dataa on oltava tarpeeksi, ja siihen kuuluvien esimerkkien on edustettava tyypillisiä neuroverkon käytössä kohtaamia tapauksia. [1] Kerätty data jaetaan yleensä kahteen osaan, koulutusaineistoon (engl. training data) ja testausaineistoon (engl. test data). [2] Usein käytetään lisäksi testausaineistosta erillistä arviointiaineistoa (engl. validation data). [7]

Koulutusaineistoa käytetään neuroverkon oppimisprosessissa. [2] Siihen kuuluvat esimerkit sisältävät neuroverkolle opetettavat tiedot, ja niiden on siis oltava edustava otos neuroverkolle ratkaistavaksi annetusta ongelmasta. [8] Kouluttamisessa yksi kierros koostuu koulutusaineiston jokaisen esimerkin esittämisestä kerran neuroverkolle. [2] Koulutusaineiston muodostamiseen käytetään 60-90 prosenttia kerätystä datasta, ja ihanteellisessa tilanteessa koulutusaineistossa on enemmän esimerkkejä kuin koulutettavassa neuroverkossa neuronien välisiin yhteyksiin liittyviä painoja. [2] [7] Vuonna 1989 Baum ja Haussler osoittivat, että myötäkytketyssä neuroverkossa, jossa käytetyt aktivaatiofunktiot ovat lineaarisia ja neuronien välisien yhteyksien painojen määrä on w, halutun tarkkuuden $(1 - e)$ saavuttamiseksi tarvitaan $\frac{w}{e}$ esimerkin suuruinen koulutusaineisto. Näin ollen jos neuroverkon tarkkuuden halutaan olevan 90%, $e = 0,1$ ja esimerkkejä tarvitaan siis kymmenen kertaa niin monta kuin neuroverkossa on painoja. Tätä esimerkkien määrää

voidaan pitää muissakin tapauksissa suosituksena. [7]

Testausaineiston avulla arvioidaan neuroverkon kykyä yleistää koulutusaineistosta oppimiansa asioita uutta dataa käsiteltäessä. On tärkeää, että tämä arviointi tehdään neuroverkolle ennestään tuntemattomalla datalla, sillä muuten koulutettava systeemi voisi tallentaa kouluttamisaineiston esimerkit ja vaikuttaa näin toimivan kouluttamisaineiston kohdalla optimaalisesti. [8] Testausaineiston muodostamiseen käytetään 10-40 prosenttia kerätystä datasta, ja sitä voidaan käyttää neuroverkon asianmukaisen yleistämiskyvyn tarkistamiseen joko kouluttamisen aikana tai sen jälkeen. [2] [6] Neuroverkon painot eivät muutu testauksen aikana, mutta testausaineistolla saatuja tuloksia voidaan käyttää neuroverkon suoritusten parantamiseen kouluttamisprosessin yhteydessä. [7]

Kun neuroverkko on koulutettu koulutusaineiston avulla ja sen toimintaa on testattu testausaineiston avulla, neuroverkon yleistämiskyky, tarkkuus ja soveltuvuus annetun ongelman ratkaisemiseen tarkistetaan käyttämällä arviointiaineistoa. [6] Kuten testausaineistonkin kohdalla, neuroverkon painot eivät muutu arvioinnin aikana. Arviointiaineistoa tulisi käyttää neuroverkon toiminnan lopulliseen tarkistamiseen, eikä sitä siis tule käyttää osana koulutusprosessia. Arvointiaineistolla saadut tulokset määrittelevät, kuinka hyvä käytettävä neuroverkkomalli on; niiden avulla todistetaan neuroverkon pystyvän ratkaisemaan tehtävä, jota varten se on koulutettu. [7]

Jos neuroverkon kouluttamisessa käytetään online-oppimista, kouluttamiseen tarvittava data kerätään jostakin prosessista kouluttamisen aikana. Suurimmassa osassa neuroverkkojen sovelluksista käytetään kuitenkin offline- eli eräoppimista, jota käytettäessä kaikki koulutus-, testaus- ja arviointiaineistoon kuuluva data kerätään etukäteen. [7] Tätä menetelmää käytettäessä neuroverkon on opittava kaikki data kerralla, joten jos koulutusaineistoon lisätään uusia esimerkkejä, neuroverkko ei voi oppia niitä erikseen vaan koko oppimisprosessi on käytävä uudelleen läpi käyttäen uutta koulutusaineistoa, joka sisältää sekä uudet että vanhat esimerkit. Tämän ongelman korjaamiseksi on olemassa oppimisalgoritmeja, joiden avulla koulutettavaa mallia voidaan muuttaa erikseen jokaisen uuden esimerkin pohjalta. [8]

Parhaan mahdollisen neuroverkkomallin suunnittelu on tärkeä osa onnis-

tunutta neuroverkkosovellusta, mutta jos käytetty koulutus-, testaus- ja arviointiaineisto eivät kuvaa ratkaistavaa ongelmaa riittävän hyvin, mallin optimaalinen suunnittelu on merkityksetöntä. Sovelluksen onnistumisen kannalta käytettävän datan käsitteleminen ja koulutusaineiston esimerkkien valitseminen ovat mallin suunnittelua huomattavasti tärkeämpiä, ja neuroverkkosovelluksen menestys riippuukin suurilta osin sen kehittäjän kyvystä ymmärtää käytettävää dataa ja esittää sitä neuroverkolle mahdollisimman kompaktissa mutta kuitenkin tietorikkaassa muodossa. [7]

4.3.4 Ali- ja ylisovittaminen

Alisovittaminen (engl. underfitting) ja ylisovittaminen (engl. overfitting) ovat neuroverkkojen tiloja, jotka voivat syntyä kun käytettävä neuroverkkomalli on liian pieni tai liian suuri. Alisovittaminen tapahtuu, kun käytettävä neuroverkko on liian pieni: jos neuroverkossa on liian vähän neuroneja ja parametreja, sen kyky oppia analysoitavasta prosessista koulutusaineiston pohjalta on riittämätön. Tällainen neuroverkko ei pysty oppimaan ongelmaan liittyvän prosessin käytöstä eikä siis tekemään prosessiin liittyviä hypoteeseja. [2] [1] Liian pieni neuroverkko ei ole tarpeeksi "joustava"oppimaan koulutusaineistoa, joten neuroverkon tekemä virhe on suuri sekä koulutus- että testausaineiston kohdalla. [2] [9] Tätä ongelmaa kutsutaan siis alisovittamiseksi, ja sen aiheuttamia hankaluuksia voidaan lieventää lisäämällä neuroverkon neuronien ja parametrien määrää. Suurempien neuroverkkojen tekemä virhe on yleensä pienempi, mutta neuroverkon kokoa kasvatettaessa neuroverkon testausaineiston kohdalla tekemä virhe alkaa jossakin vaiheessa jälleen kasvaa. Tämä johtuu siitä, että liian suuret neuroverkot ovat alttiita toisenlaiselle ongelmalle, ylisovittamiselle. [9]

Jos neuroverkossa on liian paljon neuroneja ja parametreja, se pystyy opettelemaan koulutusaineiston ulkoa. Tällainen neuroverkko on liian "joustava": Se pystyy tuottamaan täsmällisiä ulostuloja koulutusaineiston esimerkeille, koska se muistaa kyseisen aineiston toivotut ulostulot. Neuroverkon tekemä virhe on tällöin koulutusaineiston kohdalla yleensä hyvin pieni. Kuitenkin kun neuroverkolle esitetään esimerkkejä, jotka eivät kuulu koulutusai-

neistoon, sen antamat ulostulot ovat merkityksettömiä. Neuroverkon yleistämiskyky on siis huono, ja sen tekemä virhe on testausaineiston kohdalla hyvin suuri. Tätä ongelmaa kutsutaan ylisovittamiseksi. Neuroverkon koon kasvattaminen harkitsemattomasti ei siis takaa asianmukaista yleistämiskykyä, vaan johtaa lähes poikkeuksetta ylisovittamiseen. [1] [2]

5 Syväoppivat neuroverkot

Syväoppivat neuroverkot (engl. deep-learning networks) eli syvät neuroverkot (engl. deep neural networks, lyh. DNN) ovat neuroverkkoja, jotka sisältävät useampia neuroneista muodostettuja kerroksia kuin aiemmin käsitellyt neuroverkot. Ne ovat siis nimensä mukaan "syviä"verrattuna muihin neuroverkkorakenteisiin. [9] Ne ovat neuroverkkojen alalla johdonmukainen kehityssuunta, jossa yhdistyvät suuret ja monimutkaiset neuroverkkorakenteet ja datan valvomaton esikäsittely. Syväoppivat neuroverkkorakenteet ovat suhteellisen uusia, mutta nyt jo menestyneitä: ne ovat läpimurto piirteiden generoimisessa (engl. feature generation), kuvia käsittelevät järjestelmät pystyvät niiden avulla tunnistamaan lähes minkä tahansa kohteen pikselöidyistä kuvista, ja ne pystyvät ensimmäisinä neuroverkkoina luovasti tuottamaan tekstejä, musiikkia ja jopa maalauksia. Syväoppivia neuroverkkoja pidetäänkin yhtenä tekoälyn tieteenalan virstanpylväistä. [8] On kuitenkin muistettava, että tietyn menetelmän suosio ei tarkoita sen olevan ihmelääke, joka pystyisi ratkaisemaan kaikki koneoppimisongelmat; jos koulutusaineiston koko on rajoitettu ja siihen kuuluvien esimerkkien piirteiden määrä on kohtuullinen, perinteiset koneoppimisen menetelmät voivat toimia yhtä hyvin tai jopa paremmin kuin syväoppimisen menetelmät. [9]

Syväoppiviksi neuroverkoiksi lasketaan neuroverkot, joissa on syöte- ja tuloskerros mukaan lukien enemmän kuin kolme kerrosta. "Syvä"onkin neuroverkkojen yhteydessä tiukasti määritelty termi, jota käytetään vain useamman kuin yhden piilokerroksen sisältävistä neuroverkoista. [10] Kuten muissakin neuroverkoissa, tietyn kerroksen neuronien ulostuloja käytetään syväoppivissa neuroverkoissa seuraavan kerroksen neuronien syötteinä. Syötekerroksen ulostuloja käytetään siis ensimmäisen piilokerroksen syötteinä, viimei-

sen piilokerroksen ulostuloja käytetään tuloskerroksen syötteinä, ja muiden piilokerrosten ulostuloja käytetään niitä seuraavien piilokerrosten syötteinä. [2] Kerrosten suuren määrän ja syötekerroksen koon takia parhaatkin tämänhetkiset syväoppivien systeemien toteutukset ovat laskennallisesti hyvin raskaita. [8]

Syvien neuroverkkojen ylimmät kerrokset koulutetaan valvotun oppimisen menetelmillä, kuten vastavirta-algoritmilla. Verkon alempien kerrosten tehtävä taas on piirteiden erottaminen; korkeintaan yhden piilokerroksen sisältäviä neuroverkkoja käytettäessä piirteiden erottaminen on tehtävä erikseen, yleensä sovelluskohteen asiantuntijan toimesta, kun taas syväoppivat neuroverkot ovat hyviä tunnistamaan hyvät piirteet automaattisesti. [9] [5] Tämä tapahtuu hierarkkisesti: Jokainen syvän neuroverkon kerros käsittelee tiettyjä edellisen kerroksen ulostuloihin perustuvia piirteitä. Jokainen kerros kokoaa ja yhdistää edellisen kerroksen käsittelemiä piirteitä, ja mitä pidemmälle neuroverkossa edetään, sitä monimutkaisempia piirteitä verkon neuronit pystyvät tunnistamaan. [10]

Syväoppivien neuroverkkojen avulla monimutkaisia funktioita pystytään approksimoimaan halutulla tarkkuudella, ja approksimointiin tarvitaan muihin neuroverkkoihin verrattuna vähemmän neuroneita ja näin ollen vähemmän neuronien välisiin yhteyksiin liittyviä painoja. Tästä syystä syväoppivien neuroverkkojen kouluttamisessa voidaan käyttää suhteellisen vähän esimerkkejä sisältäviä koulutusaineistoja. [5] Toisaalta syväoppivat neuroverkot pystyvät löytämään piileviä rakenteita raa'asta datasta, jonka esimerkkien tavoiteulostuloja ei tiedetä, kuten kuvista, teksteistä, videoista ja äänitteistä. Ne ovat hyviä käsittelemään ja klusteroimaan sekä etsimään yhtäläisyyksiä ja poikkeavuuksia juuri tällaisesta datasta, jota kukaan ihminen ei ole järjestellyt, ja koska tällainen data muodostaa valtaosan maailman datasta, sitä on saatavilla paljon. Syvien neuroverkkojen kyky käsitellä valtavia määriä tällaista raakaa dataa parantaa niiden suorituksia ja antaa niille etulyöntiaseman verrattuna aikaisempiin neuroverkkorakenteisiin; mitä enemmän dataa neuroverkolla on käytettävissä kouluttamisen aikana sitä parempi sen suoritus todennäköisesti on. [10]

5.1 Logistinen regressio

Luokittelutehtävää suoritettaessa syväoppivan neuroverkon tuloskerroksen tehtävänä on luokitella jokainen neuroverkolle annetuista esimerkeistä sille todennäköisimpään luokkaan. Jokainen tuloskerroksen neuroni edustaa yhtä mahdollisista luokista, ja esimerkin luokittelemiseksi sen on edelliseltä kerrokselta saamiensa syötteiden ja parametrien perusteella tuotettava todennäköisyys sille, että kyseinen esimerkki kuuluu sen edustamaan luokkaan. Tuloskerroksen neuronien ulostulojen on siis kuuluttava välille $[0, 1]$. Niiden syötteet kuitenkin kuuluvat usein paljon suuremmalle välille; tuloskerroksen neuronien on siis pystyttävä muuttamaan jollekin jatkuvalle välille kuuluvat syötteet välille $[0, 1]$ kuuluvaksi, todennäköisyyttä kuvaavaksi arvoksi. Tähän tarkoitukseen käytetään logistiseksi regressioksi kutsuttua menetelmää. Tässä menetelmässä tuloskerroksen neuronien aktivaatiofunktiona käytetään funktiota

$$f(u) = \frac{1}{1 + e^{-u}}.$$

Tämä funktio sopii syötteiden muuttamiseen todennäköisyyksiksi, koska sen arvot kuuluvat välille $(0, 1)$, kun $u \to \infty$ niin $f(u) \to 1$, ja kun $u \to -\infty$ niin $f(u) \to 0$. Neuroverkolle annettu esimerkki luokitellaan siihen luokkaan, jota edustavan neuronin laskema todennäköisyys on kaikkein suurin. Jos mahdollisia luokkia on vain kaksi, voidaan tuloskerrokselle asettaa lisäksi kynnysarvo siten, että tuloskerroksen neuroni antaa ulostulona arvon 1, jos sen laskema todennäköisyys on kynnysarvoa suurempi, ja muussa tapauksessa arvon 0. Kynnysarvolla voidaan vaikuttaa virheellisiin tuloksiin: matala kynnysarvo kasvattaa väärien positiivisten tuloksien eli tuloksien joissa esimerkille annetaan luokaksi arvo 1 määrää, kun taas korkea kynnysarvo kasvattaa väärien negatiivisten tuloksien eli tuloksien joissa esimerkille annetaan luokaksi arvo 0 määrää. [10]

5.2 Syväoppivia neuroverkkorakenteita

Tässä osiossa esitellään kaksi syväoppivaa neuroverkkorakennetta.

5.2.1 Konvoluutioneuroverkko

Konvoluutioneuroverkot (engl. convolutional neural networks, lyh. CNN) ovat erityisesti kuvien analysointiin suunniteltuja syväoppivia neuroverkkoja. Tavallisten monikerroksisten neuroverkkojen syötteet ovat aina vektorimuodossa, mikä kuvia analysoitaessa tuhoaa tiettyä syötteisiin liittyvää tietoa: kuvien pikseleiden tai kolmiulotteisten kuvien vokseleiden keskinäinen rakenne sisältää tietoa, joka häviää kun kuvan pikselit tai vokselit muutetaan vektorimuotoon. Konvoluutioneuroverkot pystyvät hyödyntämään tätä rakenteen sisältämää tietoa, sillä ne pystyvät ottamaan kaksi- tai kolmiulotteisia kuvia vastaan syötteinä. Ne sisältävät yleensä tavallisten neuronikerrosten lisäksi konvoluutiokerroksia (engl. convolutional layers), joiden tehtävä on tunnistaa piirteitä syötteistä, ja alinäytteistyskerroksia (engl. sub-sampling layers), jotka auttavat laskemiseen kuluvan ajan lyhentämisessä. [5]

5.2.2 Pinotut autoenkooderit

Pinotut autoenkooderit (engl. stacked autoencoders, lyh. SAE) ovat syväoppivia neuroverkkoja, jotka koostuvat autoenkoodereista. Autoenkooderit ovat tietyntyyppisiä syötekerroksesta, tuloskerroksesta ja yhdestä piilokerroksesta koostuvia neuroverkkoja, joiden tehtävänä on muodostaa niille syötteenä annetulle datalle tiivistetty esitysmuoto minimoimalla syötteen ja tiivistetyn esitysmuodon välinen rekonstruktiovirhe. Vähän kerroksia sisältävän rakenteensa takia yhden autoenkooderin kyky tuottaa tällaisia esitysmuotoja on rajallinen. Useita autoenkoodereita pinoamalla voidaan kuitenkin rakentaa pinottu autoenkooderi, jonka kyky tuottaa tiivistettyjä esitysmuotoja on huomattavasti parempi. Pinottujen autoenkoodereiden rakenne on hierarkkinen: mitä matalammalla kerros on neuroverkossa, sitä yksinkertaisempia ovat sen tunnistamat kuviot, ja mitä korkeammalla kerros on, sitä monimutkaisempia tai abstraktimpia kuvoita se pystyy tunnistamaan. Kuten tavallistenkin monikerroksisten neuroverkkojen kohdalla, myös pinottujen autoenkoodereiden kouluttamisessa voidaan käyttää vastavirta-algoritmia, mutta lokaaleihin optimeihin juuttumisen estämiseksi kannattaa harkita ahnetta kerroksittaista oppimista (engl. greedy layer-wise learning), jossa neuroverkon kerrokset

koulutetaan yksitellen. [5]

5.3 Sovelluksia

Neuroverkkojen kehityksen alkuperäinen tavoite oli pääasiassa hahmontunnistukseen kykenevien systeemien kehittäminen, ja ensimmäiset neuroverkkojen epätriviaalit teolliset sovellukset 1980-luvulla liittyivätkin hahmontunnistukseen tai signaalien tunnistamiseen. [1] Nykyään neuroverkoilla ja erityisesti syväoppivilla neuroverkoilla on lukemattomia sovelluskohteita kaikilla teollisuudenaloilla. Hahmontunnistus on edelleen erittäin tärkeä sovelluskohde; neuroverkkojen sovelluskohteisiin kuuluvat esimerkiksi ihmisten tai kasvojen tunnistaminen valokuvista, sormenjälkien tunnistaminen, kalaparvien tunnistaminen kaikuluotaimen lukemista, sotilasajoneuvojen tunnistaminen ja luokittelu tutkakuvien perusteella, kaikuluotain- ja tutkasignaalien luokittelu, teollisuudessa käytettyjen komponenttien tunnistaminen ja lajittelu, puheentunnistus, ja käsinkirjoitetun tekstin tunnistamiseen liittyvät sovellukset, kuten esimerkiksi pakettien ja kirjeiden osoitteiden postinumeroiden tai sekkien automaattinen lukeminen. [8] [1] [6]

Neuroverkot oppivat hahmontunnistuksen ohella paljon muitakin tehtäviä; niitä voidaan käyttää esimerkiksi itseohjautuvien autojen ohjaamiseen, älykkäiden kulkuneuvojen optimaalisten reittien suunnitteluun, robottien liikkeiden ohjaamiseen, koneiden komponenttien keston ennustamiseen, poliittisissa vaaleissa valituksi tulemisen todennäköisyyden arviointiin, hakujen tekemiseen internetsivustoilla ja tiedonlouhintaan (engl. data mining). [8] [1] [6] Terveydenhoidossa niiden sovelluskohteisiin kuuluvat muun muassa verinäytteiden analysointi, syövän havaitseminen ja sydänkohtausten diagnosointi, kun taas pankkialalla ja pörssissä neuroverkkoja käytetään osakkeiden hintojen ennustamiseen, pankin asiakkaiden luottokelpoisuuden arviointiin, osakkeiden myynnistä ja ostamisesta päättämiseen, lainahakemusten arviointiin ja luottokorttipetosten havaitsemiseen. [8] [6] Neuroverkoista on hyötyä myös backgammonia ja shakkia pelaaville tietokoneille, ja syväoppimisen avulla pystyttiin rakentamaan ensimmäinen tietokoneohjelma joka kykeni voittamaan yhden maailman parhaista Go-pelin pelaajista. [8]

Yksi syväoppivien neuroverkkojen uusi ja kiehtova sovellusala on luovuus. Syväoppimisen avulla ohjelmistot pystyvät esimerkiksi säveltämään jazz-musiikkia, muodostamaan kahdesta kuvasta ensimmäisen kuvan tyyliä ja toisen kuvan sisältöä mukailevan uuden kuvan, tuottamaan lyhyitä tietokoneohjelmia ja oppimaan kielioppia. Shakespearen töiden avulla koulutettu ohjelma pystyy jopa tuottamaan seuraavanlaista uutta tekstiä:

> KING LEAR:
> O, if you were a feeble sight, the courtesy of your law,
> Your sight and several breath, will wear the gods
> With his heads, and my hands are wonder'd at the deeds,
> So drop upon your lordship's head, and your opinion
> Shall be against your honour.

Syväoppiminen on siis tehnyt tekstien, musiikin ja jopa maalausten luovan tuottamisen ensimmäistä kertaa mahdolliseksi neuroverkoille. [8]

6 Esimerkkiongelma: kuvien luokittelu

Tässä osiossa luokitellaan neuroverkon avulla Fashion-MNIST-datasettiin kuuluvia kuvia. Muodostetaan kaksi erilaista neuroverkkoa, joiden avulla pyritään luokittelemaan käytettävään testausaineistoon kuuluvat kuvat mahdollisimman suurella tarkkuudella. Tutkimusta varten asetetaan kaksi tutkimusongelmaa, joihin pyritään löytämään vastaus neuroverkkojen hyperparametreja muuttamalla ja vertailemalla eri neuroverkkojen tarkkuuksia.

6.1 Tutkimusongelmat

Tavoitteena on tutkimuksen puitteiden mahdollistamien keinojen avulla löytää neuroverkko, jolla saavutetaan testausaineistoa luokitellessa mahdollisimman suuri tarkkuus. Tutkimusongelmia on kaksi.

Ensimmäinen tutkimusongelma on seuraava: Millä hyperparametreilla myötäkytketyn neuroverkon tarkkuus on testausaineistoa luokitellessa suurin?

Tähän kysymykseen pyritään vastaamaan muuttamalla seuraavia hyperparametreja:

- **Piilokerrosten määrä.** Tutkimuksen aikana tämä hyperparametri käy läpi arvot 1, 2 ja 3.
- **Piilokerrosten neuronien määrät.** Tutkimuksen aikana tämä hyperparametri käy läpi arvot 400, 600, 800 ja 1200.
- **Piilokerrosten aktivaatiofunktio.** Tutkimuksen aikana tämä hyperparametri käy läpi vaihtoehdot "relu"(Rectified linear unit), "sigmoid"ja "tanh"(Hyperbolic tangent function).
- **Optimointialgoritmi.** Tutkimuksen aikana tämä hyperparametri käy läpi vaihtoehdot "SGD"(Stochastic gradient descent) ja "RMSprop"(Root mean square propagation).
- **Erän koko.** Tutkimuksen aikana tämä hyperparametri käy läpi arvot 10, 50, 100, 200 ja 500.
- **Kierrosten määrä.** Tutkimuksen aikana tämä hyperparametri käy läpi arvot 10, 15, 20, 25, 30, 35, 40 ja 45.

Toinen tutkimusongelma on seuraava: Saavutetaanko konvoluutioneuroverkkoa käyttämällä parempi tarkkuus testausaineistoa luokitellessa kuin parasta löydettyä myötäkytkettyä neuroverkkoa käyttämällä?

6.2 Tutkiminen

Tutkimuksessa käytettävänä alustana oli Google Colaboratory [13], ja neuroverkkojen toteutus tehtiin Python-ohjelmointikielellä [14]. Tutkimuksen ensimmäisessä osassa käytetty ohjelma perustuu tämän Pro gradu -tutkielman ohjaajan Turun yliopiston matematiikan ja tilastotieteen laitoksen yliopistonlehtori Yury Nikulinin kirjoittamaan ohjelmaan. Tutkimuksen toisessa osassa

käytetty ohjelma perustuu lähteessä [15] esitettyyn ohjelmaan. Tutkimuksen ensimmäisessä osassa käytetty ohjelma on esitetty liitteessä A, jossa näkyvät myös ohjelman tulosteet, kun sitä käytetään parhaan löydetyn myötäkytketyn neuroverkon luomiseen, kouluttamiseen ja arvioimiseen. Tiedot kaikista tutkimuksen ensimmäisessä osassa muodostetuista ja koulutetuista neuroverkoista ovat liitteessä B. Tutkimuksen toisessa osassa käytetty ohjelma on esitetty liitteessä C.

Tutkimuksessa käytetty Fashion-MNIST-datasetti sisältää 70000 Zalando-yrityksen tuotekuvaa, joista 60000 kuuluu koulutusaineistoon ja 10000 testausaineistoon. Jokainen datasettiin kuuluva esimerkki on 28 pikseliä leveä ja korkea harmaasävyinen kuva, joka kuuluu yhteen kymmenestä mahdollisesta luokasta. Luokat on numeroitu nollasta yhdeksään, ja niiden sisältämien tuotekuvien kuvaukset ovat "T-paita", "housut", "villapaita", "mekko", "takki", "sandaalit", "paita", "lenkkitossut", "laukku"ja "nilkkurit". Jokaiseen luokkaan kuuluu 7000 esimerkkiä. [16]. Tällä datasetillä parhaat saavutetut tarkkuudet laskettuna oikein luokiteltujen esimerkkien prosentuaalisena määränä testausaineiston kaikista esimerkeistä testausaineistoa luokiteltaessa vaikuttavat olevan noin 83 prosentin ja 97 prosentin välillä. Monet näistä tuloksista on saavutettu konvoluutioneuroverkkojen avulla. [17]

Tutkiminen aloitettiin pyrkimällä vastaamaan ensimmäiseen tutkimusongelmaan eli selvittämällä, millä hyperparametreilla myötäkytketyn neuroverkon tarkkuus on testausaineistoa luokitellessa suurin. Aluksi data valmisteltiin käyttöä varten, luotiin ja koulutettiin käytettävä neuroverkkomalli, arvoitiin sen toimivuutta testausaineiston avulla, tehtiin kuvaajat koulutus- ja arviointiaineiston luokittelun tarkkuudesta ja virheestä koulutuksen aikana jotta voitiin tarkastella neuroverkon mahdollista ylisovittamista, ja kokeiltiin esimerkkikuvan luokittelua koulutetun neuroverkon avulla. Tämän jälkeen tutkimus toteutettiin seuraavasti: Muutettiin yhden hyperparametrin arvoa kerrallaan, luotiin ja koulutettiin jokaisella arvolla neuroverkko, arvioitiin sen toimivuutta testausaineiston avulla, varmistettiin että ylisovittamista ei tapahdu, ja valittiin lopulta hyperparametrille se arvo jolla saadun neuroverkon tarkkuus oli testausaineistoa luokitellessa paras. Tämän jälkeen

siirryttiin muuttamaan seuraavan hyperparametrin arvoa siten, että edelliselle hyperparametrille valittu arvo kiinnitettiin eikä sen arvoa siis enää muutettu. Kaikille hyperparametreille valittiin lähtöarvot, joiden pohjalta niiden arvoja lähdettiin yksitellen muuttamaan. Valitut lähtöarvot olivat

- piilokerrosten määrä: 1,
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800,
- piilokerrosten aktivaatiofunktio: relu,
- optimointialgoritmi: SGD,
- erän koko: 200, ja
- kierrosten määrä: 30.

Näillä lähtöarvoilla muodostetun neuroverkon tarkkuus testausaineistoa luokitellessa oli $0,8466$.

Arvojen muuttaminen aloitettiin piilokerroksen neuronien määrästä. Ensimmäisen eli ainoan piilokerroksen neuronien määrä vähennettin ensin alkuarvosta 800 arvoon 400, ja kun tällä arvolla muodostettu neuroverkko oli koulutettu ja testattu, arvoa nostettiin ensin arvoon 600 ja viimeisenä arvoon 1200. Testatuista arvoista arvolla 1200 saatiin paras tulos: tällä arvolla muodostetun neuroverkon tarkkuus testausaineistoa luokiteltaessa oli $0,8449$. Tämä arvo on kuitenkin pienempi kuin lähtöarvoilla muodostetun neuroverkon tarkkuus, ja tässä neuroverkossa ensimmäisessä piilokerroksessa oli 800 neuronia. Paras tarkkuus saatiin siis ensimmäisen piilokerroksen neuronien määrän ollessa 800, joten tämän hyperparametrin arvo kiinnitettiin kyseiseen arvoon. Tässä vaiheessa parhaan löydetyn neuroverkon tarkkuus testausaineistoa luokiteltaessa oli siis $0,8466$.

Seuraavaksi piilokerrosten määrä nostettiin kahteen. Ensimmäisen piilokerroksen neuronien määrä pidettiin arvossa 800, ja toisen piilokerroksen neuronien määrän testaaminen aloitettiin arvosta 400, jonka jälkeen se nostettiin

ensin arvoon 600, tämän jälkeen arvoon 800 ja lopulta arvoon 1200. Muodostetuista neljästä neuroverkosta parhaan tarkkuuden saavutti neuroverkko, jossa toisen piilokerroksen neuronien määrä oli 600; kyseisen neuroverkon tarkkuus testausaineistoa luokiteltaessa oli $0,8576$. Tämä arvo on suurempi kuin aikaisempi paras saavutettu tarkkuus $0,8466$, joten toisen piilokerroksen neuronien määräksi valittiin 600, tällä arvolla muodostetusta neuroverkosta tuli tässä vaiheessa paras löydetty malli ja sen tarkkuus testausaineistoa luokiteltaessa oli siis $0,8576$.

Seuraavaksi piilokerrosten määrä nostettiin kolmeen. Ensimmäisen piilokerroksen neuronien määrä pidettiin arvossa 800, toisen piilokerroksen neuronien määrä pidettiin arvossa 600 ja kolmannen piilokerroksen neuronien määrän testaaminen aloitettiin arvosta 400, jonka jälkeen se nostettiin ensin arvoon 600, tämän jälkeen arvoon 800 ja lopulta arvoon 1200. Muodostetuista neljästä neuroverkosta parhaan tarkkuuden saavutti neuroverkko, jossa kolmannen piilokerroksen neuronien määrä oli 800; kyseisen neuroverkon tarkkuus testausaineistoa luokiteltaessa oli $0,8626$. Tämä arvo on suurempi kuin aikaisempi paras saavutettu tarkkuus $0,8576$, joten kolmannen piilokerroksen neuronien määräksi valittiin 800, tällä arvolla muodostetusta neuroverkosta tuli tässä vaiheessa paras löydetty malli ja sen tarkkuus testausaineistoa luokiteltaessa oli siis $0,8626$.

Kun sopiva piilokerrosten määrä ja sopivat piilokerrosten neuronien määrät oli löydetty, siirryttiin etsimään sopivaa piilokerrosten aktivaatiofunktiota. Tässä vaiheessa parhaassa löydetyssä mallissa piilokerrosten aktivaatiofunktiona oli lähtöarvona käytetty relu, jolla saavutettu tarkkuus testausaineistoa luokiteltaessa oli siis $0,8626$. Kun piilokerrosten aktivaatiofunktioksi vaihdettiin sigmoid-funktio, muodostetun neuroverkon tarkkuus testausaineistoa luokiteltaessa oli $0,6298$, ja kun aktivaatiofunktioksi asetettiin hyperbolinen tangenttifunktio, muodostetun neuroverkon tarkkuus testausaineistoa luokiteltaessa oli $0,8530$. Suurin tarkkuus saavutettiin siis neuroverkolla, jossa piilokerrosten aktivaatiofunktiona oli relu. Näin ollen piilokerrosten aktivaatiofunktioksi valittiin relu ja tässä vaiheessa parhaan löydetyn mallin tarkkuus testausaineistoa luokiteltaessa oli siis $0,8626$.

Tämän jälkeen siirryttin etsimään sopivaa optimointialgoritmia. Tässä

vaiheessa parhaassa löydetyssä mallissa optimointialgoritmina oli lähtöarvona käytetty SGD, jolla saavutettu tarkkuus testausaineistoa luokiteltaessa oli siis $0,8626$. Kun optimointialgoritmiksi vaihdettiin RMSprop, muodostetun neuroverkon tarkkuus testausaineistoa luokiteltaessa oli $0,8840$, mutta kyseisessä neuroverkossa tapahtui selkeästi ylisovittamista: tämä oli nähtävissä siitä, että hyvin aikaiselta kierrokselta alkaen neuroverkon tarkkuus arviointiaineistoa luokiteltaessa ei enää noussut vaan pysyi kaikilla seuraavilla kierroksilla jokseenkin samana, ja neuroverkon arviointiaineistoa luokiteltaessa tekemä virhe alkoi kasvaa. Lisäksi sen tekemä virhe testausaineistoa luokiteltaessa oli $0,8505$ joka oli yli kaksinkertainen verrattuna tässä vaiheessa parhaan löydetyn neuroverkon vastaavaan virheeseen, joka oli $0,3879$. Näin ollen tämä neuroverkko hylättiin, optimointialgoritmiksi valittiin SGD ja tässä vaiheessa parhaan löydetyn mallin tarkkuus testausaineistoa luokiteltaessa oli siis edelleen $0,8626$.

Seuraavaksi alettiin muuttaa erän kokoa. Aluksi erän koko laskettiin arvoon 10, ja tämän jälkeen se nostettiin arvoon 50. Molemmissa tapauksissa muodostetun neuroverkon tarkkuus testausaineistoa luokiteltaessa oli parempi kuin tässä vaiheessa paras saavutettu tarkkuus: arvolla 10 saatu tarkkuus oli $0,8866$ ja arvolla 50 tarkkuus oli $0,8807$. Kuitenkin molemmissa muodostetuissa neuroverkoissa oli havaittavissa merkkejä ylisovittamisesta, joten ne hylättiin. Tämän jälkeen erän koko nostettiin arvoon 100 ja lopulta arvoon 500. Paras tarkkuus saavutettiin neuroverkolla, jossa erän koko oli 100: tämän neuroverkon tarkkuus testausaineistoa luokiteltaessa oli $0,8727$. Tämä arvo on suurempi kuin aikaisempi paras saavutettu tarkkuus $0,8626$ joten erän kooksi valittiin 100, tällä arvolla muodostetusta neuroverkosta tuli tässä vaiheessa paras löydetty malli ja sen tarkkuus testausaineistoa luokiteltaessa oli siis $0,8727$.

Viimeiseksi siirryttiin etsimään sopivaa kierrosten määrää. Kierrosten määräksi kokeiltiin arvoja 10, 15, 20, 25, 35, 40 ja 45, ja arvo 45 hylättiin, sillä sitä käytettäessä neuroverkossa havaittiin ylisovittamista. Lopuista arvoista paras tarkkuus saavutettiin arvolla 35, jota käytettäessä neuroverkon tarkkuus testausaineistoa luokiteltaessa oli $0,8772$. Tämä arvo on suurempi kuin aikaisempi paras saavutettu tarkkuus $0,8727$ joten kierrosten määräksi

valittiin 35 ja tällä arvolla muodostetusta neuroverkosta tuli tässä vaiheessa paras löydetty malli. Koska kierrosten määrä oli testattavista hyperparametreista viimeinen, tässä vaiheessa parhaasta löydetystä neuroverkkomallista tuli lopullinen paras löydetty neuroverkkomalli. Näin ollen lopullisen parhaan löydetyn neuroverkkomallin hyperparametrien arvot olivat

- piilokerrosten määrä: 3,
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800,
    - toinen piilokerros: 600,
    - kolmas piilokerros: 800,
- piilokerrosten aktivaatiofunktio: relu,
- optimointialgoritmi: SGD,
- erän koko: 100, ja
- kierrosten määrä: 35.

Kyseisen neuroverkon tekemä virhe testausaineistoa luokiteltaessa oli $0,3462$, ja sen tarkkuus testausaineistoa luokiteltaessa oli $0,8772$.

Kun ensimmäinen tutkimusongelma oli ratkaistu eli kun oli selvitetty, millä hyperparametreilla myötäkytketyn neuroverkon tarkkuus on testausaineistoa luokitellessa suurin, siirryttiin käsittelemään toista tutkimusongelmaa. Jotta voitaisiin selvittää, saavutetaanko konvoluutioneuroverkkoa käyttämällä parempi tarkkuus testausaineistoa luokiteltaessa kuin parasta löydettyä myötäkytkettyä neuroverkkoa käyttämällä, oli tarpeen muodostaa konvoluutioneuroverkko, jota voitaisiin käyttää testausaineiston luokittelemiseen. Kun saataisiin tietää mikä on kyseisen konvoluutioneuroverkon tarkkuus testausaineistoa luokiteltaessa, tätä tarkkuutta voitaisiin verrata parhaaseen myötäkytketyllä neuroverkolla saatuun tarkkuuteen $0,8772$, ja tällä tavalla saada vastaus toiseen tutkimusongelmaan.

Aluksi data valmisteltiin käyttöä varten. Tämän jälkeen muodostettiin konvoluutioneuroverkko, joka sisälsi kolme konvoluutiokerrosta ja kolme "max-pooling-alinäytteistyskerrosta. Aktivaatiofunktioksi valittiin "Leaky ReLU", joka on vaihtoehtoinen versio tavallisesta relu- eli "Rectified Linear Unit-funktiosta. Erän kooksi valittiin 64 ja kierrosten määräksi 20. Optimointialgoritmiksi valittiin "Adam", joka kuuluu gradientin laskeutumista käyttäviin optimointialgoritmeihin. Kun malli oli koulutettu, sen toimintaa testattiin testausaineistolla ja havaittiin, että mallin tarkkuus testausaineistoa luokiteltaessa oli $0,9207$. Mallissa oli kuitenkin selvästi tapahtunut ylisovittamista, mikä havaittiin koulutuksen aikana koulutus- ja arviointiaineiston luokittelun tarkkuudesta ja virheestä tehdyistä kuvaajista: tarkkuus arviointiaineistoa luokiteltaessa ei muutaman kierroksen jälkeen muuttunut juuri ollenkaan, ja arviointiaineiston luokittelussa tehty virhe alkoi nopeasti kasvaa. Tämän ongelman korjaamiseksi muodostettiin uusi konvoluutioneuroverkko, jossa käytettiin osittaista kytkentää (engl. dropout). Osittainen kytkentä kytkee tietyn määrän sattumanvaraisesti valittuja verkon neuroneita pois päältä, jotta neuroverkko ei pystyisi opettelemaan koulutusaineistoa ulkoa. Tällä tavalla se pyrkii vähentämään neuroverkossa tapahtuvaa ylisovittamista. Tämän tekniikan käyttö oli ainoa ero alkuperäisen ja uuden konvoluutioneuroverkon välillä. Kun uusi malli oli koulutettu, sen toimintaa testattiin testausaineistolla ja havaittiin, että mallin tekemä virhe testausaineistoa luokiteltaessa oli $0,2316$ ja sen tarkkuus testausaineistoa luokiteltaessa oli $0,9219$. Koulutuksen aikana koulutus- ja arviointiaineiston luokittelun tarkkuudesta ja virheestä tehdyistä kuvaajista havaittiin, että tässä mallissa ei tapahtunut ylisovittamista. Näin ollen malli hyväksyttiin, ja toiseen tutkimusongelmaan saatiin vastaus: konvoluutioneuroverkkoa käyttämällä saavutettiin parempi tarkkuus testausaineistoa luokiteltaessa kuin parasta löydettyä myötäkytkettyä neuroverkkoa käyttämällä.

6.3 Tulosten analysointi ja johtopäätökset

Tutkimuksen tuloksena asetetuille tutkimusongelmille onnistuttiin löytämään vastaukset. Ensimmäisen tutkimusongelman vastaukseksi saatiin, että

myötäkytketyn neuroverkon tarkkuus testausaineistoa luokiteltaessa on suurin seuraavilla hyperparametreilla:

- piilokerrosten määrä: 3,
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800,
  - toinen piilokerros: 600,
  - kolmas piilokerros: 800,
- piilokerrosten aktivaatiofunktio: relu,
- optimointialgoritmi: SGD,
- erän koko: 100, ja
- kierrosten määrä: 35.

Näillä hyperparametreilla muodostetun neuroverkon tekemä virhe testausaineistoa luokiteltaessa oli $0,3462$ ja sen tarkkuus testausaineistoa luokiteltaessa oli $0,8772$.

Toisen tutkimusongelman vastaukseksi saatiin, että konvoluutioneuroverkkoa käyttämällä tosiaan saavutetaan parempi tarkkuus testausaineistoa luokiteltaessa kuin parasta löydettyä myötäkytkettyä neuroverkkoa käyttämällä. Muodostetun konvoluutioneuroverkon tekemä virhe testausaineistoa luokiteltaessa oli $0,2316$ ja sen tarkkuus testausaineistoa luokiteltaessa oli $0,9219$.

Saatuja tuloksia voidaan pitää kolmen seikan takia onnistuneina. Ensimmäiseksi, molempiin asetettuihin tutkimusongelmiin löydettiin tutkimuksen avulla vastaus. Toiseksi, aiemmin mainittiin, että parhaat käytetyllä datasetillä saavutetut tarkkuudet laskettuna oikein luokiteltujen esimerkkien prosentuaalisena määränä testausaineiston kaikista esimerkeistä testausaineistoa luokiteltaessa vaikuttavat olevan noin välillä 83%-97%, jolle myös molempien tutkimuksessa muodostettujen neuroverkkojen prosentuaaliset tarkkkuudet, $87,72\%$ ja $92,19\%$, selvästi kuuluvat. Kolmanneksi, molempien tutkimuksessa muodostettujen neuroverkkojen testausaineistoa luokiteltaessa tekemät

virheet ovat muihin tutkimuksen aikana koulutettuihin neuroverkkoihin verrattuna kohtuullisen pienet.

Vaikka saatuja tuloksia voidaankin pitää melko onnistuneina, on kuitenkin joitakin seikkoja, jotka tutkimuksessa huomioon ottamalla oltaisiin mahdollisesti voitu saada parempia tuloksia, mutta jotka kuitenkin jätettiin tutkimuksen ulkopuolelle. Ensimmäistä tutkimusongelmaa selvitettäessä olisi esimerkiksi mahdollista, että neljää tai useampaa piilokerrosta käyttämällä, piilokerrosten neuronien määrää kasvattamalla yli arvon 1200, useampia aktivaatiofunktioita tai optimointialgoritmeja kokeilemalla, yrittämällä korjata ylisovittamista niissä neuroverkoissa joissa ylisovittamista oli havaittavissa, tai kokeilemalla kaikkia mahdollisia tai ainakin useampia hyperparametrien yhdistelmiä oltaisiin löydetty tutkimuksessa parasta löydettyä neuroverkkoa paremman tarkkuuden saavuttava neuroverkko. Lisäksi on mahdollista, että käytetty datasetin jako koulutusaineistoon ja testausaineistoon saattoi vaikuttaa sopivan hyperparametrien yhdistelmän valitsemiseen, mikä oltaisiin voitu välttää esimerkiksi käyttämällä k-kertaista ristiinvalidointia (engl. k-fold cross-validation). Toista tutkimusongelmaa selvitettäessä taas olisi mahdollista, että muuttamalla muodostetun konvoluutioneuroverkon hyperparametreja olisi saavutettu parempi tarkkuus testausaineistoa luokiteltaessa. Tätä ei kuitenkaan kokeiltu, sillä tutkimusongelmana oli selvittää, löydetäänkö sellainen konvoluutioneuroverkko jonka avulla saavutetaan parempi tarkkuus kuin parhaalla löydetyllä myötäkytketyllä neuroverkolla. Tällainen konvoluutioneuroverkko löydettiin, eikä siis ollut tarpeen löytää kyseistä neuroverkkoa paremman tarkkuuden saavuttavaa konvoluutioneuroverkkoa.

Toteutettu tutkimus toimii osoituksena siitä, että kuvia luokittelevan neuroverkon muodostaminen ei ole erityisen vaikeaa, siihen tarvittavat välineet ovat hyvin saatavilla ja käytettyjen työkalujen avulla tällaisen neuroverkon toteuttaminen onnistuu tavallisella kannettavalla tietokoneella ongelmitta. Kokonaisuutena voidaan sanoa, että tutkimukselle asetetut tavoitteet saavutettiin: onnistuttiin muodostamaan neuroverkko, joka pystyi luokittelemaan käytettyyn datasettiin kuuluvia kuvia, ja molempiin asetettuihin tutkimusongelmiin löydettiin vastaus.

7 Johtopäätökset

Aika jolloin keinotekoisia neuroverkkoja pidettiin vain kikkana tai ajatusharjoituksena on epäilemättä ohi. [3] Monissa yrityksissä neuroverkkojen käyttöä ongelmanratkaisuun ei enää tarvitse perustella johdolle myyntipuheella, sillä niillä on jo niin suuri määrä sovelluskohteita teollisuudessa; neuroverkkoja käytetään nykyisin rutiininomaisesti muun muassa prosessinohjauksessa, tuotannossa, laadunvalvonnassa, tuotesuunnittelussa, tilinpäätösanalyysissa, petosten havaitsemisessa, lainojen hyväksymisessä, tiedonlouhinnassa sekä äänen, käsinkirjoitettujen tekstien ja kasvojen tunnistamisessa. Esimerkiksi kasvojen tunnistamisessa voitaisiin tietysti käyttää myös tavanomaista tietokonetta, joka voitaisiin ohjelmoida kasvojen tunnistamista varten käyttämällä matemaattisia kaavoja, joilla kuvaillaan jokaisia yksittäisiä kasvoja. Neuroverkkoa käytettäessä ei kuitenkaan tarvita tällaista prosessia; neuroverkolle voidaan antaa syötteinä digitaaliset kuvat analysoitavista kasvoista, ja antaa neuroverkon itse ratkaista kulloinkin ratkaistavana oleva ongelma ilman kouluttajan erikseen antamia täsmällisiä kuvauksia tutkittavista kasvoista. [7]

Neuroverkkojen ja muiden koneoppimisen menetelmien suunnittelussa ja ymmärtämisessä biologinen ajatusmalli ei tällä hetkellä ole juurikaan avuksi. Päinvastoin: keinotekoisten neuroverkkojen avulla pystytään muodostamaan joitakin hermoston osia kuvaavia yksinkertaisia malleja, ja tästä syystä keinotekoiset neuroverkot edesauttavat yhä useammin biologisten neuroverkkojen ymmärtämistä. Niiden mahdollistama mallinnus auttaa aivojen toiminnan periaatteiden ymmärtämisessä, ja saattaa jonakin päivänä näin jopa olla hyödyksi koneiden suunnittelussa. Tämän kiehtovan tutkimusalan lisäksi neuroverkkojen uskotaan tuovan uusia näkökulmia siihen, kuinka ohjelmointia ja algoritmien suunnittelua voitaisiin erinäisten päämäärien saavuttamiseksi ysinkertaistaa. Vaikka ne eivät tietenkään tule syrjäyttämään matematiikkaa ja logiikkaa, jotka tulevat aina tarjoamaan mahdolliselle pelkistämiselle systemaattisen pohjan, neuroverkkojen aletaan laajalti luottaa tuovan huomiota mahdollisimman yksinkertaisiin algoritmeihin. [1] [3]

Neuroverkkojen ala on kehittynyt valtavasti viimeisen vuosikymmenen

aikana, mutta neuroverkot ovat silti edelleen kehityksensä alkuvaiheessa. Ne saivat alkunsa 1950-luvulla, niihin kohdistuva laajalle levinnyt kiinnostus alkoi 1980-luvulla, ja nykyisin niiden kuuluisi olla osa työkalupakkia kaikille niille tieteentekijöille, jotka haluavat ottaa kaiken irti saatavilla olevasta datasta muun muassa tekemällä ennusteita tai tunnistamalla kuvioita tai signaaleja. [3] [7] [1] Minkään yhden työkalun tai tieteenalan ei tietenkään voida olettaa pystyvän kaikkeen; differentiaaliyhtälöryhmää ratkaistaessa ei käytetä neuroverkkoa, mutta tunnistamiseen tai ohjaukseen liittyvät ongelmat sopivat niille hyvin. Kaiken kaikkiaan voidaan varmasti sanoa, että keinotekoiset neuroverkot ovat vakavasti otettava ja huomionarvoinen ala. [3]

Kirjallisuutta

[1] Gérard Dreyfus: *Neural Networks: Methodology and Applications, third edition*. Springer, Saksa, 2005.

[2] Ivan Nunes da Silva et al.: *Artificial Neural Networks: A Practical Course*. Springer, Sveitsi, 2017.

[3] Daniel Graupe: *Principles of Artificial Neural Networks*. World Scientific, Singapore, 2007.

[4] N. H. Siddique, Hojjat Adeli: *Computational Intelligence: Synergies of Fuzzy Logic, Neural Networks and Evolutionary Computing*. John Wiley & Sons Inc., Chichester, Iso-Britannia, 2013.

[5] S. Kevin Zhou, Hayit Greenspan, Dinggang Shen: *Deep Learning for Medical Image Analysis*. Academic Press, Lontoo, Iso-Britannia, 2017.

[6] Anthony Zaknich: *Neural Networks for Intelligent Signal Processing*. World Scientific, Singapore, 2003.

[7] Mary M. Poulton: *Computational Neural Networks for Geophysical Data Processing*. Pergamon, New York, 2001.

[8] Wolfgang Ertel: *Introduction to Artificial Intelligence, Second Edition.* Springer International Publishing, 2017.

[9] Miroslav Kubat: *An Introduction to Machine Learning, Second Edition.* Springer International Publishing, 2017.

[10] Chris Nicholson: *A Beginner's Guide to Neural Networks and Deep Learning.* Pathmind, `https://pathmind.com/wiki/neural-network`, 11.6.2020.

[11] Marc Peter Deisenroth, A. Aldo Faisal, Cheng Soon Ong: *Mathematics for Machine Learning.* Cambridge University Press, 2020.

[12] GeoGebra. `https://www.geogebra.org/?lang=fi`, 14.7.2020.

[13] Google Colaboratory. `https://colab.research.google.com/notebooks/intro.ipynb`, 5.9.2020.

[14] Python, Python Software Foundation. `https://www.python.org/`, 5.9.2020.

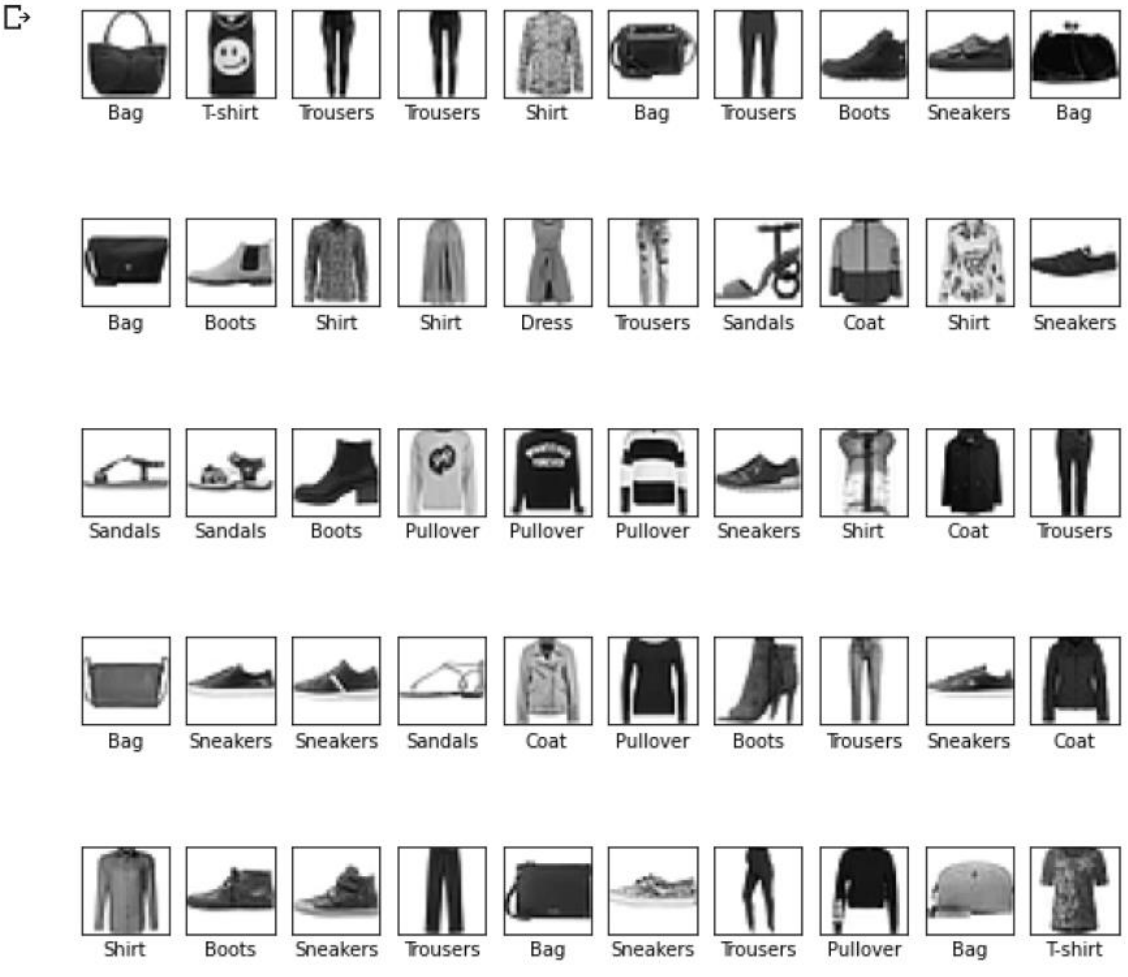
[15] Aditya Sharma: *Convolutional Neural Networks in Python with Keras.* DataCamp, `https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python`, 5.9.2020.

[16] Han Xiao, Kashif Rasul, Roland Vollgraf: *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* Cornell University, `https://arxiv.org/abs/1708.07747`, 5.9.2020.

[17] GitHub, *Fashion-MNIST.* `https://github.com/zalandoresearch/fashion-mnist`, 5.9.2020.

A Tutkimuksen ensimmäisessä osassa käytetty ohjelma

```
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import utils
from tensorflow.keras.preprocessing import image
from google.colab import files
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
%matplotlib inline


#PREPARING DATA FOR NETWORK TRAINING
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
#LIST OF CLASSES
classes = ['T-shirt', 'Trousers', 'Pullover', 'Dress', 'Coat',
           'Sandals', 'Shirt', 'Sneakers', 'Bag', 'Boots']


#SAMPLE OF IMAGES

plt.figure(figsize=(10,10))
for i in range(100,150):
    plt.subplot(5,10,i-100+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(classes[y_train[i]])
```



```
#CONVERTING DATA DIMENSIONS IN DATA SET
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

#NORMALIZING DATA
x_train = x_train / 255
x_test = x_test / 255

#WORKING WITH CORRECT ANSWERS
n = 100
print(y_train[n])

#CONVERTING LABELS TO ONE HOT ENCODING
y_train = utils.to_categorical(y_train, 10)
y_test = utils.to_categorical(y_test, 10)

#CORRECT ANSWER (CLASS LABEL) IN ONE HOT ENCODING FORMAT
print(y_train[n])
```

```
8
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

```
#CREATE NEURAL NETWORK
model = Sequential()
# input layer, 800 neurons, with 784 inputs each
model.add(Dense(800, input_dim=784, activation="relu"))
# second hidden layer
model.add(Dense(600, activation="relu"))
# third hidden layer
model.add(Dense(800, activation="relu"))
# Output  layer, 10 neurons=number of classes in classification problem
# layers are fully connected
model.add(Dense(10, activation="softmax"))

#COMPILE NETWORK
model.compile(loss="categorical_crossentropy", optimizer="SGD",
              metrics=["accuracy"])
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 800)               628000
_________________________________________________________________
dense_5 (Dense)              (None, 600)               480600
_________________________________________________________________
dense_6 (Dense)              (None, 800)               480800
_________________________________________________________________
dense_7 (Dense)              (None, 10)                8010
=================================================================
Total params: 1,597,410
Trainable params: 1,597,410
Non-trainable params: 0
_________________________________________________________________
None
```

```
#TRAINING NEURAL NETWORK
```

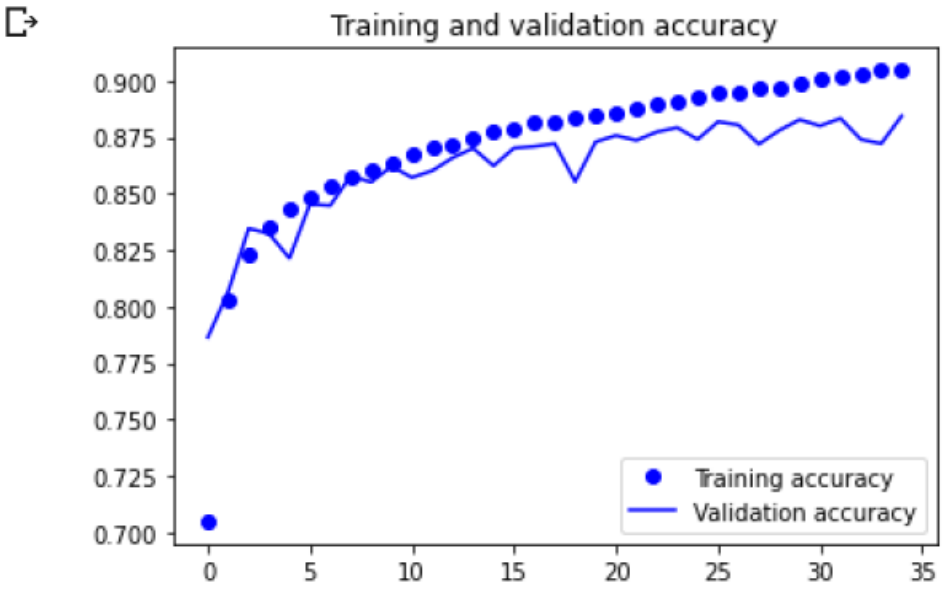
```
history = model.fit(x_train, y_train,
                    batch_size=100,
                    epochs=35,
                    validation_split=0.2,
                    verbose=1)
```

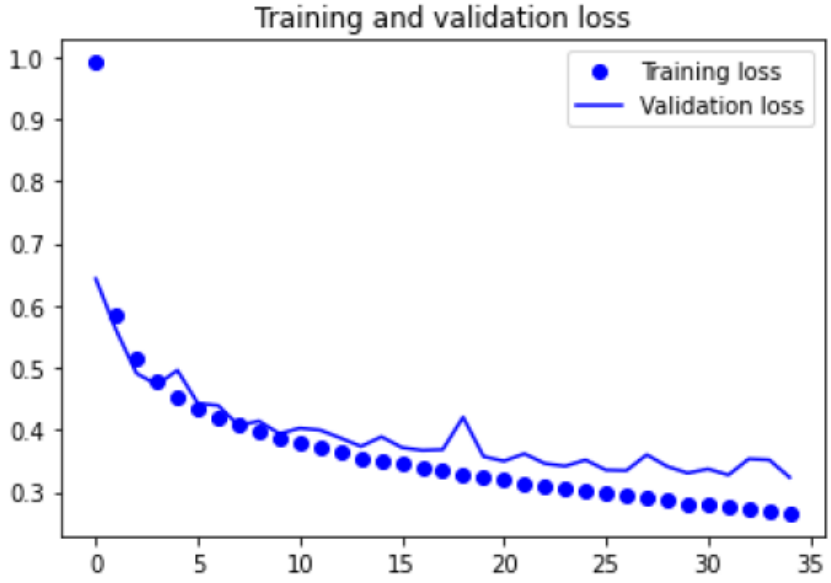
```
Epoch 1/35
480/480 [==============================] - 14s 30ms/step - loss: 0.9916 - accuracy: 0.7050 - val_loss: 0.6432 - val_accuracy: 0.7863
Epoch 2/35
480/480 [==============================] - 14s 29ms/step - loss: 0.5851 - accuracy: 0.8029 - val_loss: 0.5604 - val_accuracy: 0.8066
Epoch 3/35
480/480 [==============================] - 13s 28ms/step - loss: 0.5146 - accuracy: 0.8228 - val_loss: 0.4909 - val_accuracy: 0.8345
Epoch 4/35
480/480 [==============================] - 14s 29ms/step - loss: 0.4788 - accuracy: 0.8349 - val_loss: 0.4727 - val_accuracy: 0.8322
Epoch 5/35
480/480 [==============================] - 14s 29ms/step - loss: 0.4535 - accuracy: 0.8433 - val_loss: 0.4959 - val_accuracy: 0.8213
Epoch 6/35
480/480 [==============================] - 13s 28ms/step - loss: 0.4351 - accuracy: 0.8482 - val_loss: 0.4428 - val_accuracy: 0.8453
Epoch 7/35
480/480 [==============================] - 14s 28ms/step - loss: 0.4212 - accuracy: 0.8529 - val_loss: 0.4392 - val_accuracy: 0.8447
Epoch 8/35
480/480 [==============================] - 14s 28ms/step - loss: 0.4080 - accuracy: 0.8569 - val_loss: 0.4070 - val_accuracy: 0.8578
Epoch 9/35
480/480 [==============================] - 14s 28ms/step - loss: 0.3970 - accuracy: 0.8602 - val_loss: 0.4142 - val_accuracy: 0.8550
Epoch 10/35
480/480 [==============================] - 14s 29ms/step - loss: 0.3880 - accuracy: 0.8635 - val_loss: 0.3938 - val_accuracy: 0.8620
Epoch 11/35
480/480 [==============================] - 16s 32ms/step - loss: 0.3781 - accuracy: 0.8675 - val_loss: 0.4027 - val_accuracy: 0.8571
Epoch 12/35
480/480 [==============================] - 14s 29ms/step - loss: 0.3704 - accuracy: 0.8705 - val_loss: 0.3996 - val_accuracy: 0.8601
Epoch 13/35
480/480 [==============================] - 15s 31ms/step - loss: 0.3653 - accuracy: 0.8714 - val_loss: 0.3868 - val_accuracy: 0.8658
Epoch 14/35
480/480 [==============================] - 14s 28ms/step - loss: 0.3553 - accuracy: 0.8748 - val_loss: 0.3733 - val_accuracy: 0.8700
Epoch 15/35
480/480 [==============================] - 13s 27ms/step - loss: 0.3492 - accuracy: 0.8775 - val_loss: 0.3889 - val_accuracy: 0.8622
Epoch 16/35
480/480 [==============================] - 13s 28ms/step - loss: 0.3455 - accuracy: 0.8786 - val_loss: 0.3715 - val_accuracy: 0.8701
Epoch 17/35
480/480 [==============================] - 13s 27ms/step - loss: 0.3385 - accuracy: 0.8813 - val_loss: 0.3671 - val_accuracy: 0.8708
Epoch 18/35
480/480 [==============================] - 13s 27ms/step - loss: 0.3337 - accuracy: 0.8819 - val_loss: 0.3682 - val_accuracy: 0.8721
Epoch 19/35
480/480 [==============================] - 13s 27ms/step - loss: 0.3274 - accuracy: 0.8839 - val_loss: 0.4205 - val_accuracy: 0.8553
Epoch 20/35
480/480 [==============================] - 13s 27ms/step - loss: 0.3232 - accuracy: 0.8848 - val_loss: 0.3571 - val_accuracy: 0.8728
Epoch 21/35
480/480 [==============================] - 13s 27ms/step - loss: 0.3195 - accuracy: 0.8855 - val_loss: 0.3498 - val_accuracy: 0.8757
Epoch 22/35
480/480 [==============================] - 13s 27ms/step - loss: 0.3145 - accuracy: 0.8880 - val_loss: 0.3616 - val_accuracy: 0.8735
Epoch 23/35
480/480 [==============================] - 13s 27ms/step - loss: 0.3103 - accuracy: 0.8898 - val_loss: 0.3459 - val_accuracy: 0.8773
Epoch 24/35
480/480 [==============================] - 13s 26ms/step - loss: 0.3051 - accuracy: 0.8906 - val_loss: 0.3416 - val_accuracy: 0.8792
Epoch 25/35
480/480 [==============================] - 13s 27ms/step - loss: 0.3019 - accuracy: 0.8931 - val_loss: 0.3516 - val_accuracy: 0.8739
Epoch 26/35
480/480 [==============================] - 14s 28ms/step - loss: 0.2980 - accuracy: 0.8947 - val_loss: 0.3354 - val_accuracy: 0.8817
Epoch 27/35
480/480 [==============================] - 13s 28ms/step - loss: 0.2945 - accuracy: 0.8949 - val_loss: 0.3345 - val_accuracy: 0.8803
Epoch 28/35
480/480 [==============================] - 13s 28ms/step - loss: 0.2894 - accuracy: 0.8972 - val_loss: 0.3600 - val_accuracy: 0.8718
Epoch 29/35
480/480 [==============================] - 13s 28ms/step - loss: 0.2858 - accuracy: 0.8972 - val_loss: 0.3412 - val_accuracy: 0.8779
Epoch 30/35
480/480 [==============================] - 13s 28ms/step - loss: 0.2814 - accuracy: 0.8991 - val_loss: 0.3304 - val_accuracy: 0.8827
Epoch 31/35
480/480 [==============================] - 13s 28ms/step - loss: 0.2784 - accuracy: 0.9004 - val_loss: 0.3371 - val_accuracy: 0.8799
Epoch 32/35
480/480 [==============================] - 14s 28ms/step - loss: 0.2747 - accuracy: 0.9022 - val_loss: 0.3274 - val_accuracy: 0.8832
Epoch 33/35
480/480 [==============================] - 13s 28ms/step - loss: 0.2716 - accuracy: 0.9030 - val_loss: 0.3533 - val_accuracy: 0.8739
Epoch 34/35
480/480 [==============================] - 13s 28ms/step - loss: 0.2683 - accuracy: 0.9045 - val_loss: 0.3518 - val_accuracy: 0.8721
Epoch 35/35
480/480 [==============================] - 13s 28ms/step - loss: 0.2657 - accuracy: 0.9044 - val_loss: 0.3235 - val_accuracy: 0.8843
```

```
#TRAINING QUALITY EVALUATION
#VERIFICATION ON TESTING DATA SET

scores = model.evaluate(x_test, y_test, verbose=1)
print("Percentage of correct answers on testing data set:",
      round(scores[1] * 100, 4))
```

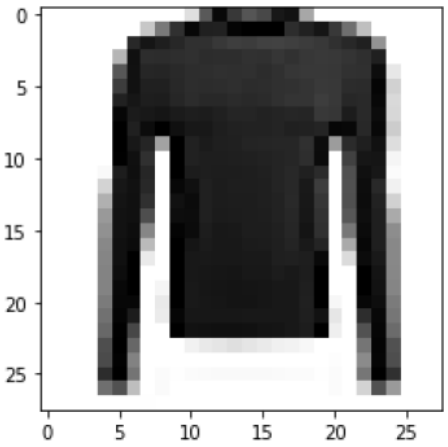
```
313/313 [==============================] - 2s 6ms/step - loss: 0.3462 - accuracy: 0.8772
Percentage of correct answers on testing data set: 87.72
```

```
#PLOT ACCURACY AND LOSS (to check if network is overfitting)

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





```
#USING NEURAL NETWORK FOR CLOTHES IMAGE RECOGNITION

n_rec = 497
plt.imshow(x_test[n_rec].reshape(28, 28), cmap=plt.cm.binary)
plt.show()
```




```
#CHANGE IMAGE DIMENSION AND NORMALIZE IT

x = x_test[n_rec]
x = np.expand_dims(x, axis=0)

#RECOGNITION RUN

prediction = model.predict(x)

#PRINT RESULTS

prediction
```

```
array([[9.8448258e-04, 9.3983545e-04, 9.8863095e-01, 2.5341082e-05,
        7.2852662e-03, 2.7954055e-08, 1.2741750e-03, 1.3676082e-07,
        8.5657206e-04, 3.3342942e-06]], dtype=float32)
```

```
#CONVERT RESULTS FROM ONE HOT ENCODING AND COMPARE IT TO
#THE IMAGE TO KNOW IF IT RECOGNIZED CORRECTLY

prediction = np.argmax(prediction[0])
print("Class number:", prediction)
print("Class name:", classes[prediction])
```

```
Class number: 2
Class name: Pullover
```

B Tutkimuksen ensimmäisessä osassa koulutettujen neuroverkkojen tiedot

Ensimmäinen tutkimusongelma: Millä hyperparametreillä myötäkytketyn neuroverkon tarkkuus on testausaineistoa luokitellessa suurin?

Hyperparametrit joita muutetaan:

- piilokerrosten määrä: 1, 2, 3, 4
- piilokerrosten neuronien määrät: 400, 600, 800, 1200
- kierrosten (epochs) määrä: 10, 15, 20, 25, 30, 40, 45
- erän koko (batch_size): 10, 50, 100, 200, 500
- piilokerrosten aktivaatiofunktio: relu, sigmoid, tanh
- optimointialgoritmi (optimizer): SGD, RMSprop

Muutetaan yhden hyperparametrin arvoa kerrallaan, ja valitaan se arvo jolla tarkkuus on paras. Aloitetaan neuronien/kerrosten määristä, seuraavaksi aktivaatiofunktio ja optimointialgoritmi, ja viimeiseksi kierrosten määrä ja erän koko.

ENSIMMÄINEN TUTKIMUSONGELMA: ALOITETAAN SEURAAVILLA HYPERPARAMETRIEN ARVOILLA

- piilokerrosten määrä: 1
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8466
➔ loss: 0.4427
➔ ylisovittamista?: EI

KIERROS 1:

- piilokerrosten määrä: 1
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 400
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8371
➔ loss: 0.4540
➔ ylisovittamista?: EI

KIERROS 2:

- piilokerrosten määrä: 1
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 600
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8425
➔ loss: 0.4486
➔ ylisovittamista?: EI

KIERROS 3:

- piilokerrosten määrä: 1
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 1200
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8449
➔ loss: 0.4417
➔ ylisovittamista?: EI

TULOS:

➔ Tarkkuus on paras, kun ensimmäisessä piilokerroksessa on **800** neuronia

KIERROS 4:

- piilokerrosten määrä: 2
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 400
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8563
➔ loss: 0.4109
➔ ylisovittamista?: EI

KIERROS 5:

- piilokerrosten määrä: 2
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 600
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8576
➔ loss: 0.4079
➔ ylisovittamista?: EI

KIERROS 6:

- piilokerrosten määrä: 2
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8546
➔ loss: 0.4134
➔ ylisovittamista?: EI

KIERROS 7:

- piilokerrosten määrä: 2
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 1200
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8544
➔ loss: 0.4115
➔ ylisovittamista?: EI

TULOS:

➔ Tarkkuus on paras, kun ensimmäisessä piilokerroksessa on **800** neuronia ja toisessa piilokerroksessa on **600** neuronia

KIERROS 8:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 400
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8569
➔ loss: 0.4021
➔ ylisovittamista?: EI

KIERROS 9:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 600
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8592
➔ loss: 0.3918
➔ ylisovittamista?: EI

KIERROS 10:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8626
➔ loss: 0.3879
➔ ylisovittamista?: EI

KIERROS 11:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 600
  - kolmas piilokerros: 1200
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8567
➔ loss: 0.3978
➔ ylisovittamista?: EI

TULOS:

➔ Tarkkuus on paras, kun ensimmäisessä piilokerroksessa on **800** neuronia, toisessa piilokerroksessa on **600** neuronia ja kolmannessa piilokerroksessa on **800** neuronia

TÄMÄNHETKINEN PARAS NEUROVERKKO:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 600
  - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8626
➔ loss: 0.3879
➔ ylisovittamista?: EI

KIERROS 12:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 600
  - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: sigmoid
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.6298

➔ loss: 1.0084
➔ ylisovittamista?: EI

KIERROS 13:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 600
  - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: tanh
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8530
➔ loss: 0.4086
➔ ylisovittamista?: EI

TULOS:

➔ Tarkkuus on paras, kun ensimmäisessä piilokerroksessa on **800** neuronia, toisessa piilokerroksessa on **600** neuronia ja kolmannessa piilokerroksessa on **800** neuronia, ja piilokerrosten aktivaatiofunktio on **relu**

KIERROS 14:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 600
  - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 200
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): RMSprop

➔ näillä arvoilla tarkkuus testausainestolla: 0.8840
➔ loss: 0.8505
➔ ylisovittamista?: KYLLÄ -> HYLÄTÄÄN (selkeää ja aikaisin alkavaa ylisovittamista)

TULOS:

➔ Tarkkuus on paras, kun ensimmäisessä piilokerroksessa on **800** neuronia, toisessa piilokerroksessa on **600** neuronia ja kolmannessa piilokerroksessa on **800** neuronia, piilokerrosten aktivaatiofunktio on **relu**, ja optimointialgoritmina on **SGD**

KIERROS 15:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 10
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8866
➔ loss: 0.3860
➔ ylisovittamista?: KYLLÄ (selvästi ja melko aikaisin)

KIERROS 16:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 50
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8807
➔ loss: 0.3385
➔ ylisovittamista?: KYLLÄ (ei valtavasti mutta kuitenkin)

KIERROS 17:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 100
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8727
➔ loss: 0.3572
➔ ylisovittamista?: EI

KIERROS 18:

- piilokerrosten määrä: 3

- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 30
- erän koko (batch_size): 500
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8465
➔ loss: 0.4482
➔ ylisovittamista?: EI

TULOS:

➔ Tarkkuus on paras, kun ensimmäisessä piilokerroksessa on **800** neuronia, toisessa piilokerroksessa on **600** neuronia ja kolmannessa piilokerroksessa on **800** neuronia, piilokerrosten aktivaatiofunktio on **relu**, optimointialgoritmina on **SGD** ja erän koko on **100**

KIERROS 19:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 10
- erän koko (batch_size): 100
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8441
➔ loss: 0.4325
➔ ylisovittamista?: EI

KIERROS 20:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 15
- erän koko (batch_size): 100
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8692
➔ loss: 0.3739

➔ ylisovittamista?: EI

KIERROS 21:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 600
  - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 20
- erän koko (batch_size): 100
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8654
➔ loss: 0.3803
➔ ylisovittamista?: EI

KIERROS 22:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 600
  - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 25
- erän koko (batch_size): 100
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8662
➔ loss: 0.3748
➔ ylisovittamista?: EI

KIERROS 23:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
  - ensimmäinen piilokerros: 800
  - toinen piilokerros: 600
  - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 35
- erän koko (batch_size): 100
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8772
➔ loss: 0.3462
➔ ylisovittamista?: EI

KIERROS 24:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 40
- erän koko (batch_size): 100
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8765
➔ loss: 0.3452
➔ ylisovittamista?: EI

KIERROS 25:

- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 45
- erän koko (batch_size): 100
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8780
➔ loss: 0.3409
➔ ylisovittamista?: KYLLÄ

LOPULLINEN PARAS NEUROVERKKO

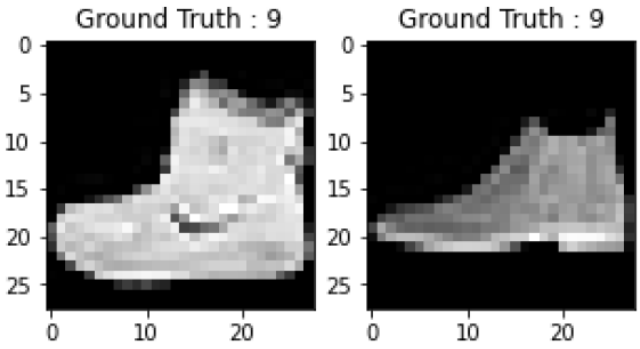
- piilokerrosten määrä: 3
- piilokerrosten neuronien määrät:
    - ensimmäinen piilokerros: 800
    - toinen piilokerros: 600
    - kolmas piilokerros: 800
- kierrosten (epochs) määrä: 35
- erän koko (batch_size): 100
- piilokerrosten aktivaatiofunktio: relu
- optimointialgoritmi (optimizer): SGD

➔ näillä arvoilla tarkkuus testausainestolla: 0.8772
➔ loss: 0.3462
➔ ylisovittamista?: EI

C Tutkimuksen toisessa osassa käytetty ohjelma

```
import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
import numpy as np
from keras.utils import to_categorical
import matplotlib.pyplot as plt
%matplotlib inline

from keras.datasets import fashion_mnist
(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()


#IMAGES
plt.figure(figsize=[5,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))
# Display the first image in testing data
plt.subplot(122)
plt.imshow(test_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(test_Y[0]))
```

```
Text(0.5, 1.0, 'Ground Truth : 9')
```




```
#RESHAPE
train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255.
test_X = test_X / 255.
# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)
# Split training set into training set and validation set
from sklearn.model_selection import train_test_split
train_X,valid_X,train_label,valid_label = train_test_split(train_X,
                                                           train_Y_one_hot,
                                                           test_size=0.2,
                                                           random_state=13)
```

```
#NETWORK
batch_size = 64
epochs = 20
num_classes = 10
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',
                         input_shape=(28,28,1),padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dense(num_classes, activation='softmax'))
```

```
#COMPILE MODEL

fashion_model.compile(loss=keras.losses.categorical_crossentropy,
                      optimizer=keras.optimizers.Adam(),
                      metrics=['accuracy'])
fashion_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 356,234
Trainable params: 356,234
Non-trainable params: 0
_________________________________________________________________
```

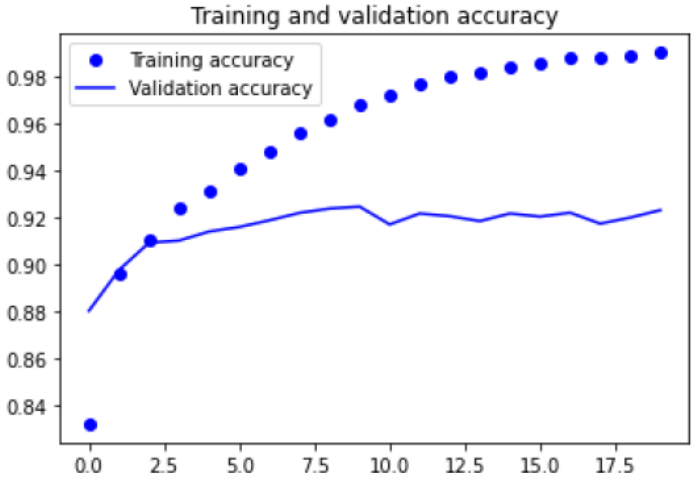
```
#TRAIN MODEL

fashion_train = fashion_model.fit(train_X, train_label,
                                  batch_size=batch_size,epochs=epochs,
                                  verbose=1,validation_data=(valid_X,
                                                             valid_label))
```

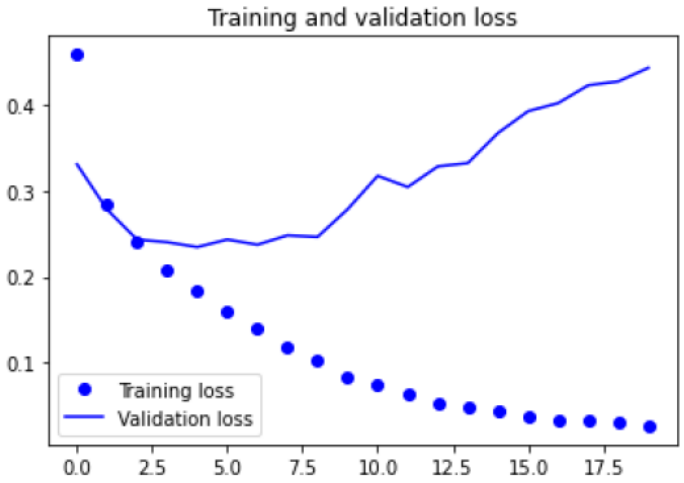
```
Epoch 1/20
750/750 [==============================] - 74s 99ms/step - loss: 0.4583 - accuracy: 0.8322 - val_loss: 0.3308 - val_accuracy: 0.8802
Epoch 2/20
750/750 [==============================] - 76s 101ms/step - loss: 0.2833 - accuracy: 0.8963 - val_loss: 0.2782 - val_accuracy: 0.8976
Epoch 3/20
750/750 [==============================] - 76s 101ms/step - loss: 0.2407 - accuracy: 0.9107 - val_loss: 0.2432 - val_accuracy: 0.9092
Epoch 4/20
750/750 [==============================] - 76s 102ms/step - loss: 0.2070 - accuracy: 0.9242 - val_loss: 0.2402 - val_accuracy: 0.9101
Epoch 5/20
750/750 [==============================] - 77s 102ms/step - loss: 0.1846 - accuracy: 0.9314 - val_loss: 0.2346 - val_accuracy: 0.9139
Epoch 6/20
750/750 [==============================] - 78s 104ms/step - loss: 0.1602 - accuracy: 0.9407 - val_loss: 0.2433 - val_accuracy: 0.9158
Epoch 7/20
750/750 [==============================] - 78s 104ms/step - loss: 0.1399 - accuracy: 0.9479 - val_loss: 0.2373 - val_accuracy: 0.9187
Epoch 8/20
750/750 [==============================] - 77s 103ms/step - loss: 0.1190 - accuracy: 0.9561 - val_loss: 0.2482 - val_accuracy: 0.9218
Epoch 9/20
750/750 [==============================] - 77s 103ms/step - loss: 0.1038 - accuracy: 0.9615 - val_loss: 0.2463 - val_accuracy: 0.9237
Epoch 10/20
750/750 [==============================] - 77s 102ms/step - loss: 0.0844 - accuracy: 0.9681 - val_loss: 0.2785 - val_accuracy: 0.9245
Epoch 11/20
750/750 [==============================] - 75s 100ms/step - loss: 0.0743 - accuracy: 0.9717 - val_loss: 0.3173 - val_accuracy: 0.9169
Epoch 12/20
750/750 [==============================] - 76s 101ms/step - loss: 0.0635 - accuracy: 0.9767 - val_loss: 0.3043 - val_accuracy: 0.9216
Epoch 13/20
750/750 [==============================] - 76s 102ms/step - loss: 0.0536 - accuracy: 0.9797 - val_loss: 0.3283 - val_accuracy: 0.9204
Epoch 14/20
750/750 [==============================] - 77s 102ms/step - loss: 0.0489 - accuracy: 0.9813 - val_loss: 0.3321 - val_accuracy: 0.9183
Epoch 15/20
750/750 [==============================] - 77s 102ms/step - loss: 0.0441 - accuracy: 0.9838 - val_loss: 0.3670 - val_accuracy: 0.9216
Epoch 16/20
750/750 [==============================] - 78s 104ms/step - loss: 0.0383 - accuracy: 0.9857 - val_loss: 0.3924 - val_accuracy: 0.9202
Epoch 17/20
750/750 [==============================] - 77s 103ms/step - loss: 0.0342 - accuracy: 0.9876 - val_loss: 0.4018 - val_accuracy: 0.9219
Epoch 18/20
750/750 [==============================] - 77s 103ms/step - loss: 0.0336 - accuracy: 0.9874 - val_loss: 0.4225 - val_accuracy: 0.9172
Epoch 19/20
750/750 [==============================] - 76s 101ms/step - loss: 0.0303 - accuracy: 0.9889 - val_loss: 0.4270 - val_accuracy: 0.9198
Epoch 20/20
750/750 [==============================] - 75s 100ms/step - loss: 0.0269 - accuracy: 0.9900 - val_loss: 0.4427 - val_accuracy: 0.9230
```

```
#TEST MODEL

test_eval = fashion_model.evaluate(test_X, test_Y_one_hot, verbose=0)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.4970065951347351
Test accuracy: 0.9207000136375427
```

```
#PLOT ACCURACY AND LOSS

accuracy = fashion_train.history['accuracy']
val_accuracy = fashion_train.history['val_accuracy']
loss = fashion_train.history['loss']
val_loss = fashion_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```






```
#MODEL SEEMS TO BE OVERFITTING.
#NEW MODEL TO PREVENT OVERFITTING
batch_size = 64
epochs = 20
num_classes = 10
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',
                         padding='same',input_shape=(28,28,1)))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.4))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dropout(0.3))
fashion_model.add(Dense(num_classes, activation='softmax'))
```

```
fashion_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               262272
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 356,234
Trainable params: 356,234
Non-trainable params: 0
_________________________________________________________________
```

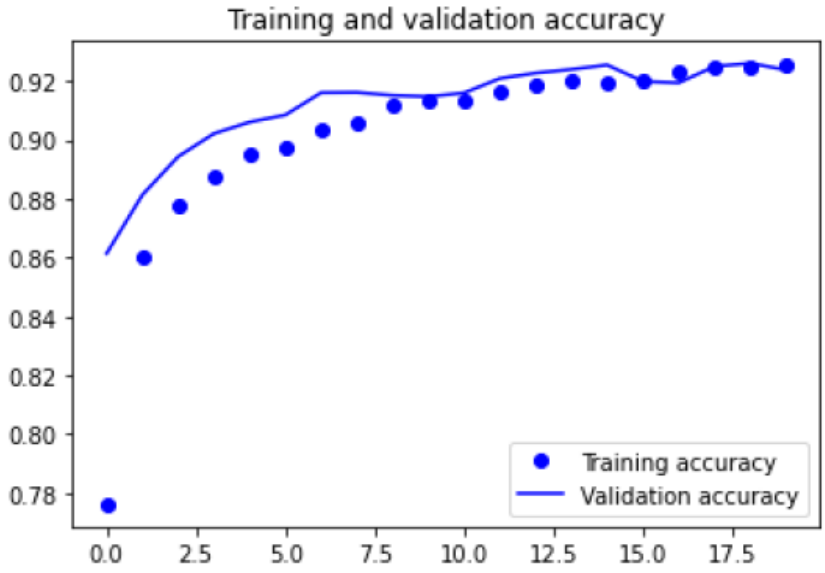
```
#COMPILE

fashion_model.compile(loss=keras.losses.categorical_crossentropy,
                      optimizer=keras.optimizers.Adam(),
                      metrics=['accuracy'])


#TRAIN

fashion_train_dropout = fashion_model.fit(train_X, train_label,
                                          batch_size=batch_size,
                                          epochs=epochs,
                                          verbose=1,
                                          validation_data=(valid_X,
                                                           valid_label))
```

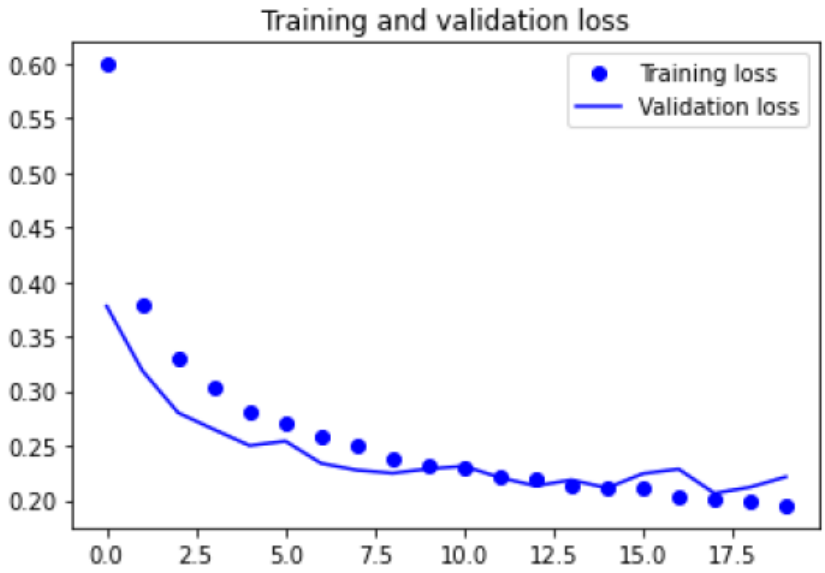
```
Epoch 1/20
750/750 [==============================] - 80s 107ms/step - loss: 0.5995 - accuracy: 0.7761 - val_loss: 0.3779 - val_accuracy: 0.8614
Epoch 2/20
750/750 [==============================] - 81s 109ms/step - loss: 0.3784 - accuracy: 0.8606 - val_loss: 0.3187 - val_accuracy: 0.8813
Epoch 3/20
750/750 [==============================] - 81s 109ms/step - loss: 0.3310 - accuracy: 0.8779 - val_loss: 0.2802 - val_accuracy: 0.8943
Epoch 4/20
750/750 [==============================] - 81s 109ms/step - loss: 0.3040 - accuracy: 0.8873 - val_loss: 0.2649 - val_accuracy: 0.9022
Epoch 5/20
750/750 [==============================] - 82s 110ms/step - loss: 0.2816 - accuracy: 0.8953 - val_loss: 0.2500 - val_accuracy: 0.9061
Epoch 6/20
750/750 [==============================] - 82s 109ms/step - loss: 0.2707 - accuracy: 0.8972 - val_loss: 0.2541 - val_accuracy: 0.9085
Epoch 7/20
750/750 [==============================] - 84s 111ms/step - loss: 0.2581 - accuracy: 0.9033 - val_loss: 0.2338 - val_accuracy: 0.9161
Epoch 8/20
750/750 [==============================] - 84s 112ms/step - loss: 0.2498 - accuracy: 0.9062 - val_loss: 0.2275 - val_accuracy: 0.9162
Epoch 9/20
750/750 [==============================] - 84s 111ms/step - loss: 0.2384 - accuracy: 0.9115 - val_loss: 0.2248 - val_accuracy: 0.9152
Epoch 10/20
750/750 [==============================] - 84s 111ms/step - loss: 0.2321 - accuracy: 0.9130 - val_loss: 0.2287 - val_accuracy: 0.9147
Epoch 11/20
750/750 [==============================] - 84s 112ms/step - loss: 0.2293 - accuracy: 0.9137 - val_loss: 0.2312 - val_accuracy: 0.9160
Epoch 12/20
750/750 [==============================] - 83s 111ms/step - loss: 0.2214 - accuracy: 0.9165 - val_loss: 0.2207 - val_accuracy: 0.9209
Epoch 13/20
750/750 [==============================] - 83s 111ms/step - loss: 0.2197 - accuracy: 0.9184 - val_loss: 0.2131 - val_accuracy: 0.9227
Epoch 14/20
750/750 [==============================] - 84s 112ms/step - loss: 0.2143 - accuracy: 0.9202 - val_loss: 0.2186 - val_accuracy: 0.9240
Epoch 15/20
750/750 [==============================] - 84s 112ms/step - loss: 0.2113 - accuracy: 0.9191 - val_loss: 0.2112 - val_accuracy: 0.9255
Epoch 16/20
750/750 [==============================] - 85s 114ms/step - loss: 0.2118 - accuracy: 0.9203 - val_loss: 0.2244 - val_accuracy: 0.9199
Epoch 17/20
750/750 [==============================] - 85s 113ms/step - loss: 0.2039 - accuracy: 0.9232 - val_loss: 0.2285 - val_accuracy: 0.9194
Epoch 18/20
750/750 [==============================] - 85s 113ms/step - loss: 0.2010 - accuracy: 0.9245 - val_loss: 0.2063 - val_accuracy: 0.9250
Epoch 19/20
750/750 [==============================] - 84s 112ms/step - loss: 0.1981 - accuracy: 0.9248 - val_loss: 0.2119 - val_accuracy: 0.9261
Epoch 20/20
750/750 [==============================] - 83s 111ms/step - loss: 0.1951 - accuracy: 0.9257 - val_loss: 0.2214 - val_accuracy: 0.9238
```

```
#EVALUATE ON TEST SET

test_eval = fashion_model.evaluate(test_X, test_Y_one_hot, verbose=1)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
313/313 [==============================] - 5s 15ms/step - loss: 0.2316 - accuracy: 0.9219
Test loss: 0.23157085478305817
Test accuracy: 0.9218999743461609
```

```
#PLOT ACCURACY AND LOSS

accuracy = fashion_train_dropout.history['accuracy']
val_accuracy = fashion_train_dropout.history['val_accuracy']
loss = fashion_train_dropout.history['loss']
val_loss = fashion_train_dropout.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Training accuracy
Validation accuracy


Training and validation loss
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Training loss
Validation loss