TURUN YLIOPISTO

TIIVISTELMÄ

	Kandidaatintutkielma
X	Pro gradu -tutkielma
	Lisensiaatintutkielma
	Väitöskirja

Oppiaine	Laskentatoimi ja rahoitus	Päivämäärä	24.3.2020
Tekijä	Juuso Juvonen	Sivumäärä	60+liitteet
Otsikko	NEUROVERKOT TALOUDELLISEN AIKASARJAN ENNUSTAMISESSA Empiirinen tutkimus S&P 500 -indeksillä		
Ohjaaja	Prof. Luis Alvarez Esteban		

Tiivistelmä

Osakemarkkinahintojen ennustaminen on haastavaa, koska niiden taustalla on monimutkaista satunnaisuutta sekä dynaamisuutta ja näiden interaktio luo ennustamattomuutta. Tutkielmassa on ennustettu S&P 500 -osakeindeksin päivittäisiä hintoja käyttämällä kolmea eri neuroverkkoa sekä ARIMA-mallia. Menetelmien ennustamiskykyä on testattu soveltamalla keskimääräistä neliövirhettä. Osakeindeksi on ladattu Thomson Reuters Eikon -tietokannasta ajanjaksoilta 23.11.2011–31.10.2019 ja 8.11.2016–31.10.2019. Pidemmän ajanjakson kehitystä kuvaava taloudellinen aikasarja sisältää 2 250 havaintoa ja lyhyemmän 750 havaintoa. Edellisestä on käytetty 1 500 havaintoa ja jälkimmäisestä on käytetty 500 havaintoa verkkojen kouluttamiseen sekä ARIMA-mallin mallintamiseen. Aikasarjojen viimeisiä 750 ja 250 havaintoa on käytetty menetelmien ennustamiseen.

Tutkielmassa verkkojen ennustamiskykyä on lähestytty kahdella tutkimuskysymyksellä: Voidaanko neuroverkoilla ennustaa arvopapereiden hintoja käyttäen historiallista aikasarjaa? Soveltuvatko neuroverkot taloudellisen aikasarjan mallintamiseen ja ennustamiseen? Ensimmäisen tutkimuskysymyksen vastausta etsitään rahoitusteorioista, joista keskeisemmät ovat tehokkaiden markkinoiden hypoteesi sekä aikasarjan momentum-teoria. Empiirisesti on havaittu, että markkinat eivät ole täydellisen tehokkaita, jonka perusteella ensimmäisen tutkimuskysymyksen vastaus on, että neuroverkoilla voidaan ennustaa arvopapereiden hintoja käyttäen historiallista aikasarjaa. Tutkielman tuloksien perusteella toisen tutkimuskysymyksen vastaus on, että kaikki tutkielmassa käytetyt neuroverkot soveltuvat taloudellisen aikasarjan mallintamiseen ja ennustamiseen lyhyemmillä aikasarjoilla. Vain RNN- sekä LSTM-verkko soveltuvat myös pidemmille aikasarjoille.

Työn tulokset osoittavat, että RNN- sekä LSTM-verkkojen ennustamistarkkuudet ovat neuroverkoista tarkimpia, koska niiden tarkkuus eri kouluttamiskertojen välillä pysyy huomattavasti vakaampana kuin MLP-verkolla. Lisäksi ne kykenevät huomioimaan aikasarjan pidemmän aikavälin riippuvaisuuksia niiden muistiominaisuuksista johtuen. Neuroverkkojen ennustamiskyky sekä niiden luotettavuus ovat riittäviä, kun aikasarjat on differentioitu. ARIMA-malli oli tutkielman tarkin ja luotettavin, kun neuroverkoille syötettäviä aikasarjoja ei differentioitu.

Avainsanat	Koneoppiminen, neuroverkot, MLP, RNN, LSTM, ARIMA, taloudellinen aikasarja

TURUN YLIOPISTO
Kauppakorkeakoulu

NEUROVERKOT TALOUDELLISEN AIKASARJAN ENNUSTAMISESSA

Empiirinen tutkimus S&P 500 -indeksillä

Laskentatoimi ja rahoitus
pro gradu -tutkielma

Laatija:
Juuso Juvonen

Ohjaaja:
Prof. Luis Alvarez Esteban

24.3.2020
Turku

SISÄLLYS

KUVIOT

TAULUKOT

1 JOHDANTO

1.1 Johdatus aiheeseen

Ennakointi eli tulevaisuutta koskevan arvion tekeminen on aina kiinnostanut ihmistä. Jo varhaisessa vaiheessa ihminen on pyrkinyt ennakoimaan saaliseläinten liikkeitä, jolloin ennakointi on tukenut päätöksentekoa oikean strategian valinnassa. Nykyään erilaiset arkielämän päätökset vaativat ennakointia, ja taloutta koskevat päätökset ovat hyvä esimerkki tästä. Taloudellisissa päätöksissä on muodostettava arviot tulevasta, mutta ennen kuin niitä kyetään muodostamaan, on päätettävä niiden lähde. Rahoitusmarkkinat tarjoavat erään rikkaan lähteen muodostaa arvioita tulevasta. Eräs menetelmä on ottaa rahoitusmarkkinoilta osakeindeksin tai yksittäisen osakkeen historiallinen aikasarja. Tätä taloudellista aikasarjaa mallinnetaan mallilla, ja jolla myös ennustetaan sen tulevia arvoja. Menetelmän tavoitteena on saada tietoa tulevasta, jotta saavutettaisiin mahdollisimman suuret taloudelliset voitot. Atsalakis ja Valavanis (2009) ovat kirjoittaneet artikkelissaan, että osakemarkkinoiden ennustajat kehittävät menetelmiä indeksiarvojen ja osakehintojen ennustamiselle, missä vähiten monimutkaiset osakemarkkinamallit käyttävät aineiston kokona vain vaadittua vähimmäismäärää. Menetelmien tavoitteena on tuottaa malleilla parhaat ennustamistulokset.

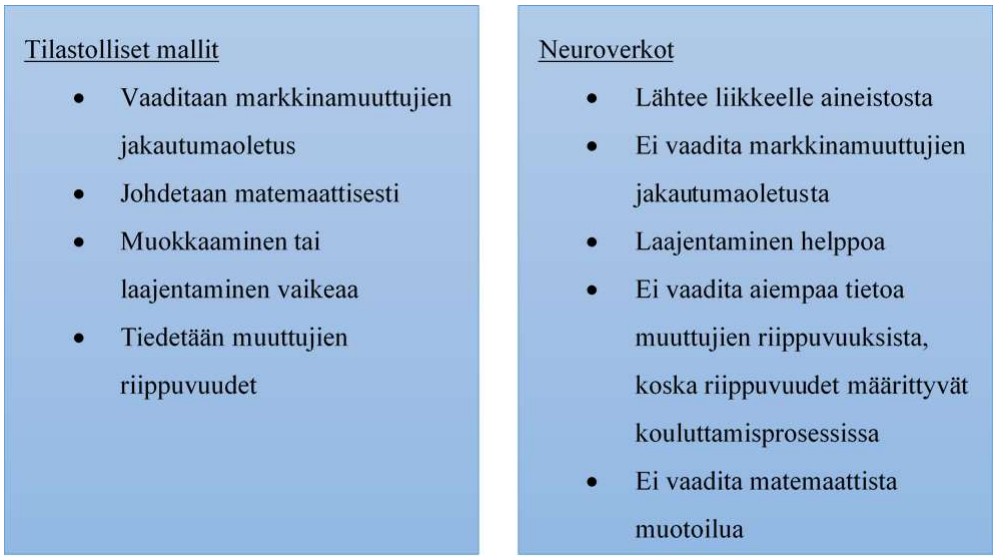
Rubinsteinin (1994) mukaan toimivilla markkinoilla hinnat sisältävät arvokasta informaatiota. Hinnoista johdettuja päätelmiä hyödynnetään laajasti taloudellisessa päätöksenteossa. Tuloksien informaatiotehokkuus riippuu neljästä ehdosta. Ensimmäinen ehto on malli, joka liittää hinnat päätelmiin. Esimerkiksi indeksiarvojen ennustamisessa käytetyn mallin on kaapattava taloudellisen aikasarjan käyttäytyminen, jotta informaatio tulevista arvoista olisi luotettavaa ja arvokasta. Toinen ehto on malli, joka on toteutettavissa suhteellisen helposti oikeaan aikaan. Kolmas ehto on mallin oikein mitatut alkuarvot. Neljäs ehto on tehokkaat markkinat.

Täten tilastolliset mallit ovat suosittuja indeksiarvojen ja osakehintojen ennustamisessa. Eräs tällainen malli on lineaarinen ARIMA (autoregressive integrated moving average model), joka on paljon käytetty taloudellisten aikasarjojen mallintamisessa ja ennustamisessa. Mallilla voidaan kaapata aikasarjan käyttäytyminen jakamalla se autoregressiiviseen sekä liukuvakeskiarvon prosessiin, jonka jälkeen prosessit yhdistetään ja mallilla voidaan ennustaa aikasarjan tulevia arvoja. Lisäksi ARCH (autoregressive conditional heteroscedasticity) ja GARCH (generalized autoregressive conditional

heteroskedasticity) ovat malleja, joita sovelletaan yleisesti rahoituksessa, varsinkin taloudellisten aikasarjojen volatiliteetin (engl. volatility) mallinnuksessa sekä ennustamisessa. Volatiliteetti on eräs tapa mitata riskiä rahoituksessa. Aikasarjan volatiliteetti kuvaa arvojen vaihtelun voimakkuutta tietyllä aikavälillä.

Keinotekoiset neuroverkot (engl. artificial neural networks) ja tietokoneiden laskentatehon kehittyminen sekä laskentatiedon määrän kasvu ovat mahdollistaneet hyvinkin tehokkaiden menetelmien hyödyntämisen osakemarkkinoiden ennustamisessa. Tutkielmassa keinotekoisista neuroverkoista käytetään käsitettä neuroverkot. Ne ovat laskentamalleja, jotka ovat kehitetty aivojen hermoverkkotoiminnan pohjalta. Neuroverkot olivat yksi lupaavimmista sekä tutkituimmista aihealueista 1960-luvulla. Niiden toivottiin olevan läpimurto aineistossa havaittujen riippuvuuksien selvittämisessä. Teknologian kehitys on nostanut uudelleen neuroverkot akateemisen tutkimuksen kentälle viimeisen vuosikymmenen aikana. Ne ovat saaneet paljon huomiota esimerkiksi rahoitustutkimuksissa. Niiden uskotaan kykenevän tarkempiin ennusteisiin kuin esimerkiksi ARIMA-mallilla tehdyt ennusteet. Kritiikki ARIMA-mallia kohtaan johtuu mallin kyvyttömyydestä kaapata kaikki ne ominaisuudet, joita on havaittu taloudellisten aikasarjojen käyttäytymisessä. Tähän käyttäytymiseen viitataan tyylitellyt tosiasiat käsitteellä (engl. stylized facts), ja sen ominaisuudet ovat empiirisesti havaittuja taloudellisten aikasarjojen keskeisiä piirteitä.

Neuroverkkojen toiminta pohjautuu algoritmeihin, jotka muodostavat funktion, jolla aineistoa voidaan mallintaa ja sen tulevia havaintoja ennustaa. Neuroverkot eivät vaadi aineistolta funktion muodostamisessa vastaavanlaisia oletuksia kuten ARIMA-malli. Lisäksi neuroverkoilla kuten myös ARCH- ja GARCH-malleilla on kyky mallintaa aikasarjan epälineaarisuuksia tehden neuroverkoista epälineaarisen mallin (Mostafa ym. 2017, 6–7). Aikasarjan epälineaarisuus tarkoittaa, että sen käyttäytyminen sisältää yllättäviä poikkeamia. Esimerkiksi eri havaintojen välisiä suhteita ei kyetä perustelemaan lineaarisesti, jolloin tulevien havaintojen ennakointi hankaloituu. Kuviossa 1 on esitetty tilastollisten mallien sekä neuroverkkojen keskeiset ominaisuudet.



Kuvio 1 Tilastollisten mallien ja neuroverkkojen ominaisuuksia (mukaellen Mostafa ym. 2017, 6).

Kuvio 1 on käännetty alkuperäisestä lähteestä suomenkielelle ja piirretty uudelleen. Huolimatta kuviossa 1 esitetyistä neuroverkkojen hyvistä ominaisuuksista, ne asettavat myös haasteita verkkojen rakentamiselle.

1.2 Tutkielman tavoitteet, rajaus ja rakenne

Tutkielmassa analysoidaan neuroverkkojen ennustamiskykyä. Tutkimuskysymykset ovat:

- *Voidaanko neuroverkoilla ennustaa arvopapereiden hintoja käyttäen historiallista aikasarjaa?*
- *Soveltuvatko neuroverkot taloudellisen aikasarjan mallintamiseen ja ennustamiseen?*

Lisäksi tutkielmassa ennustetaan ARIMA-mallilla, jotta saadaan lähtötaso neuroverkkojen ennustamiskyvylle, koska ARIMA-malli edustaa ennustamismenetelmien klassista näkökulmaa. Tutkielman neuroverkot ovat MLP, RNN ja LSTM. Neuroverkkojen aineiston mallintaminen eli oppiminen perustuu aineiston eteenpäin syöttämiseen. Suurin ero tutkimuksen verkoilla johtuu niiden erilaisista muistiominaisuuksista. Tämä tarkoittaa aineiston pidemmän aikavälin riippuvuuksien huomioimista. MLP-mallissa ei ole muistiominaisuuksia, kun taas LSTM-mallin muistiominaisuudet ovat kehittyneemmät kuin RNN-mallissa.

Tutkielman aineisto on S&P 500 -indeksin historiallista kehitystä kuvaava taloudellinen aikasarja. Indeksi ladataan Thomson Reuters Eikon -tietokannasta käyttäen

ajanjaksoa 23.11.2011–31.10.2019, joka jaetaan lyhyeen sekä pitkään aikaväliin. Lyhyen aikavälin aikasarja sisältää 750 havaintoa ajalta 8.11.2016–31.10.2019, kun taas pitkän aikavälin aikasarja sisältää 2 250 havaintoa kattaen koko ajanjakson. Havainnot ovat S&P 500 -indeksin peräkkäisten päivien päätöskursseja, joiden valuuttana ovat eurot.

Tutkielma rakentuu johdanto luvun jälkeen seuraavasti: Luvussa 2 esitellään rahoitusteoriaa sekä vastataan ensimmäiseen tutkimuskysymykseen, voidaanko neuroverkoilla ennustaa arvopapereiden hintoja käyttäen historiallista aikasarjaa? Luvussa 3 tehdään katsaus tekoälyyn (engl. artificial intelligence), koneoppimiseen (engl. machine learning) ja syväoppimiseen (engl. deeplearning) sekä esitellään tutkielmassa käytettävät neuroverkot että niiden oppimisen taustalla oleva vastavirta-algoritmi (engl. back propagation algorithm). Luvun lopuksi tehdään kirjallisuuskatsaus aiheesta tehtyihin tutkimuksiin. Luvussa 4 esitellään aineisto, neuroverkkojen rakentamisessa käytetyt parametrit, ARIMA-malli ja menetelmien ohjelmointi. Luvussa 5 perustellaan menetelmille valitut parametrit sekä esitellään tutkielman tulokset että vastataan toiseen tutkimuskysymykseen, soveltuvatko neuroverkot taloudellisen aikasarjan mallintamiseen ja ennustamiseen? Luvussa 6 on tutkielman yhteenveto.

2 RAHOITUSTEORIA

2.1 Informaatiotehokkaat rahoitusmarkkinat

Rahoitusmarkkinoiden sanotaan olevan informaatiotehokkaita (engl. market efficiency), kun arvopaperien hintoihin heijastuu välittömästi uusi ja olennaisesti tärkeä informaatio (Niskanen & Niskanen 2016, 38). Käsite on keskeinen rahoituksessa ja sijoitussalkun hallinnassa. Se voidaan käsittää ohjeena, kuinka muodostetaan odotukset kannattavasta kaupankäynnistä. Lisäksi se antaa lähtökohdan kysymykselle: Onko arvopaperien ennustaminen kannattavaa historiallisesta aikasarjasta?

Eltonin ym. (2014, 411) mukaan tehokkaiden markkinoiden teoria ulottuu vuoteen 1863, jolloin Jules Regnault esitteli kirjassaan teorian aikaisinta muotoa. Regnault oli ensimmäinen henkilö, joka perusteli tehokkaiden markkinoiden ja hintojen välistä yhteyttä. Hänen mukaansa tehokkaat markkinat antoi ymmärtää, että arvopaperihintojen pitäisi noudattaa satunnaiskulkua (engl. random walk).

Vuonna 1900 ranskalainen matemaatikko Louis Bachelier julkaisi väitöskirjansa, jossa hän tutki optiohintojen kauppaa Pariisin pörssissä. Hän kehitti mallin kuvaamaan arvopaperihintojen kehittymistä. Malli noudatti Regnaultin satunnaiskulkua, jonka Bachelier muotoili täsmällisemmin Brownin liikeyhtälöillä. Hän hyödynsi Brownin liikettä jo ennen Albert Einsteinia. Brownin liike on jatkuva-aikainen kuvaus satunnaisprosessista. Brownin liike implikoi, että hinnat kehittyvät satunnaisesti ja ovat gaussisesti jakautuneita, sillä Brownin liikkeen satunnaisuus generoituu normaalijakautumasta. Bachelierin väitöskirjaa voidaan pitää lähtölaukauksena kvantitatiiviselle rahoitukselle ja nykypäivän optioiden hinnoittelumalleille. (Elton ym. 2014, 411.)

2.2 Tehokkaiden markkinoiden hypoteesi

Vuonna 1970 Eugene Fama julkaisi tutkimuksen, jossa hän kokosi yhteen siihen asti kirjoitettuja tehokkaita markkinoita koskevia tutkimuksia ja teorioita. Faman (1970) tutkimus kulminoitui tehokkaiden markkinoiden hypoteesiin (engl. efficient markets hypothesis, lyh. EMH). Hypoteesin mukaan markkinat ovat täydellisen tehokkaat, kun kaikki saatavilla oleva informaatio heijastuu täysin hintoihin.

EMH on yhtenäistänyt ja kiihdyttänyt rahoituksen tutkimusta, koska se on vahvasti yhteydessä empiriaan. Jensenin (1978) mukaan ei ole toista yhtä vankkaa selitystä, jota

empiria tukisi kuin EMH. EMH:n testauksessa markkinoiden tehokkuus voidaan jakaa kolmeen tyyppiin:

- Heikot ehdot täyttävä tehokkuus (engl. weak form efficiency) tarkoittaa, että arvopaperien hintoihin sisältyy kaikki historiallinen informaatio niiden kurssikehityksestä. Täten teknisen analyysin eli hintahistoriaan perustuvilla menetelmillä ei voida saavuttaa keskimääräistä parempia voittoja, koska kaikilla sijoittajilla on sama informaatio käytettävissään. Sen sijaan fundamenttianalyysia eli tilinpäätöksistä johdetuilla menetelmillä voidaan ennustaa mahdollisesti tulevia voittoja.
- Puolivahvat ehdot täyttävä tehokkuus (engl. semistrong form efficiency) tarkoittaa, että kaikki julkinen informaatio, kuten osinko-, voitto ja -investointiedot heijastuvat välittömästi arvopapereiden hintoihin. Tällöin yrityksen taloudellinen analyysi ei ole kannattavaa tuottojen ennustamisessa.
- Vahvat ehdot täyttävä tehokkuus (engl. strong form efficiency) tarkoittaa, että arvopaperien hinnat heijastavat kaikkea saatavilla olevaa informaatiota ja jopa sisäpiiritiedon.

Huolimatta EMH:n asemasta rahoitusteoriassa, on se saanut myös kritiikkiä. Grossman (1976) sekä Grossman ja Stiglitz (1980) väittävät, että täydellisen tehokkaat markkinat ovat mahdottomuus, koska tällöin informaation keräämisellä ei olisi saavutettavissa taloudellista hyötyä. Täten tuottoisa kaupankäynti osakkeilla olisi hyödytöntä, mikä johtaisi markkinoiden romahtamiseen. Tutkijat perustelevat väittämäänsä sillä, että markkinat tarvitsevat jonkin asteisen tehottomuuden, jolla informaation kerääminen hyvitetään sijoittajille. Jopa Fama (1991) toteaa artikkelissaan, että täydellisen tehokkaiden markkinoiden olettamus on väärä, mutta mahdollistaa lähestymisen empirialle.

EMH:n vasta-argumentteja on perusteltu perinteisesti anomalioiden tai tyyliteltyjen tosiasioiden avulla. Anomaliat ovat markkinoilla havaittuja poikkeamia, jotka poikkeavat EMH:sta. Tyylitellyt tosiasiat ovat markkinoiden tilastollisia ominaisuuksia, joita on usein empiirisesti havaittu, ja ne ovat vastakkaisia markkinatehokkuudelle.

2.3 Aikasarjan momentum-teoria

Osakemarkkinoilla arvopaperihinnan trendinomaista kehittymistä kutsutaan momentumiksi (engl. momentum). Empiirisesti on esimerkiksi havaittu, että nousevan osakkeen hinta tahtoo pysyä liikeradallaan, ja tämä pätee myös osakkeen laskevalle hinnalle. Tällaista momentum-anomalian syntymistä on selitetty ihmisten taipumuksella seurata

trendejä. Malkiel (2003) kutsuu tällaista taipumusta *”voittajan vankkurit”* -ilmiöksi (engl. bandwagon effect).

Moskowitz ym. (2012) esittelevät artikkelissaan aikasarjan momentum-teorian (engl. time series momentum), jolla he perustelevat osaketuottojen ennustamista historiallisesta aikasarjasta. Tutkijat tekevät eron heidän ja perinteisen momentumin välillä siten, että perinteisessä momentumissa keskitytään osakkeiden suhteelliseen suorituskyvyntarkasteluun, joka on ensisijaisesti poikkileikkauksellista (engl. cross-sectional) tutkimusta. Tutkijoiden aikasarjan momentum-teoria keskittyy ainoastaan osakkeiden omiin historiallisiin tuottoihin. Tutkijat mainitsevat artikkelissaan, että heidän teoriansa soveltuu suoraan moniin arvopaperien hinnoittelua selittäviin sijoittajien käyttäytymisteorioihin. Tämä vahvistaa näkemystä, että tulevien osakekurssien tai osaketuottojen ennustaminen on kannattavaa, ja kun ennusteet suoritetaan riittävän tarkasti se voi johtaa sijoittajan ylisuuriin voittoihin.

2.4 Taloudelliset aikasarjat

Rahoitustutkimuksissa on havaittu, että taloudellisilla aikasarjoilla on tyypillisiä tilastollisia ominaisuuksia. Lisäksi sarjat käyttäytyvät samankaltaisesti eri markkinoilla. Taloudellisten aikasarjojen ominaisuuksiin viitataan käsitteellä tyylitellyt tosiasiat. Mostafa ym. (2017, 19–20) esittelevät kirjassaan seuraavia tosiasioita:

- Tuotoista muodostettu jakautuma sisältää ”paksummat hännät” kuin normaalijakautumalla.
- Voitto ja tappio epäsymmetria: määrällisesti omaisuuserän arvonlaskut eivät vastaa niiden nousuja. *(Ei sovellu valuuttojen vaihtokursseille.)*
- Tarkastelujakson pidentyessä tuottojakautuma lähentyy normaalijakautuman muotoa. Lisäksi tuottojakautuman muoto muuttuu tuottojen määrän lisääntyessä tai tarkastelujakson vaihtuessa.
- Tuottojen väliaikaisuus: tuottosarjoilla ilmenee korkeaa vaihtelua millä tahansa tarkastelujaksolla.
- Volatiliteetin klusteroituminen (engl. volatility clustering): tuottojen suurista muutoksista seuraa suuret vaihtelut, ja pienistä muutoksista pienet vaihtelut.
- Tuottosarjoilla on heteroskedastisuutta.

Edellä mainitut teoriat sekä tyylitellyt tosiasiat ovat vastakkaisia markkinatehokkuuden määritelmälle. Esimerkiksi EMH-hypoteesin taustalla oleva Brownin liikkeen oletukset tuottojen normaalijakautuneisuudesta ovat ristiriidassa tyyliteltyjen tosiasioiden kanssa. Toisaalta oletus tuottojen tai hintojen riippumattomuudesta ei täyty, koska momentum-anomalian perusteella arvopaperien hinnat tahtovat pysyä liikeradallaan. Tämä merkitsee, että menneet arvot sisältävät informaatiota tulevista arvoista. Lisäksi volatiliteetin klusteroituminen kertoo, että taloudellisten aikasarjojen hinnat eivät noudata satunnaiskulkua. ARCH- ja GARCH-mallit ovat suosittuja volatiliteetin mallintamisessa, koska ne kykenevät kaappaamaan taloudellisten aikasarjojen volatiliteetin klusteroitumisen (Franses & Dijk 2000, 135–137).

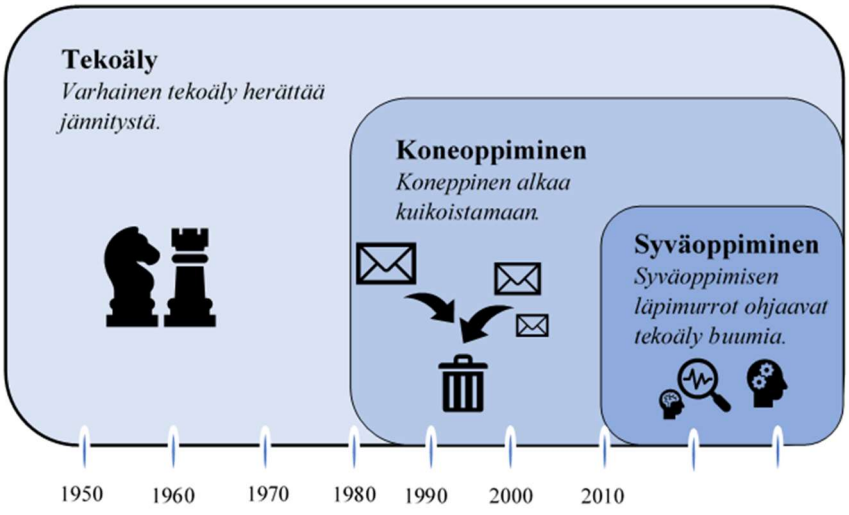
Yhdessä aikasarjan momentum-teoria sekä tyylitellyt tosiasiat antavat teoreettista pohjaa vastata tutkielman ensimmäiseen tutkimuskysymykseen, voidaanko neuroverkoilla ennustaa arvopapereiden hintoja käyttäen historiallista aikasarjaa? Näiden teorioiden pohjalta voidaan katsoa, että neuroverkoilla voidaan ennustaa arvopapereiden hintoja käyttäen historiallista aikasarjaa, koska käytännössä markkinat eivät ole täydellisen tehokkaita.

3 KONEOPPIMINEN

3.1 Tekoäly ja koneoppiminen

Tekoäly (engl. artificial intelligence, lyh. AI) on yksi uusimmista tieteen ja tekniikan tutkimusaloista, jonka kukoistus on alkanut toisen maailmansodan jälkeen. Se on lähtenyt liikkeelle halusta ymmärtää, kuinka ihmisen ajattelu toimii. Koska ihmisen ajattelu ei ole selitettävissä yksinkertaisella tavalla, ovat tekoälyn kehittymiseen vaikuttaneet useat eri tieteet kuten filosofia, matematiikka, psykologia ja tietotekniikka. Esimerkiksi filosofit ovat luoneet ajatuksen, että mieli on verrattavissa koneeseen. Matemaatikot ovat tarjonneet työkaluja käsitellä logiikkaa, epävarmuutta ja todennäköisyyksiä. Lisäksi he ovat luoneet perustan algoritmilaskennan ymmärtämiselle. Tietokoneinsinöörit ovat tarjonneet tehokkaita laskentakoneita, jotka ovat mahdollistaneet tekoälyohjelmat. Tiivistetysti tekoäly pyrkii mallintamaan ihmisen älykkyyttä keinotekoisesti. (Russell ja Norvig 2010, 1–30.)

Tekoäly soveltuu moniin tehtäviin, jotka vaativat ongelmanratkaisua sekä automatisointia. Kun tekoälyn toiminta ei ole valmiiksi ohjelmoitua, on kyse koneoppimisesta (engl. machine learning). Koneoppimisessa tietokone oppii itsenäisesti käyttämällä matemaattisia algoritmeja. Tällöin tietokoneelle ei tarvitse määritellä jokaista tilannetta erikseen. Koneoppimisen tuorein suuntaus on ollut syväoppiminen (engl. deeplearning), jossa neuroverkkojen kerroksien lukumäärää on kasvatettu. Tällä on tavoiteltu parempaa verkkojen oppimiskykyä. Kuviossa 2 havainnollistetaan näiden käsitteiden välistä yhteyttä sekä niiden kehittymistä viimeisen kuuden vuosikymmenen aikana.



Kuvio 2 Tekoälyn kehittyminen (mukaellen Nvidia 2019).

Kuvio 2 on käännetty alkuperäisestä lähteestä suomenkielelle ja piirretty uudelleen. Kuviossa 2 tekoäly, koneoppiminen ja syväoppiminen eroavat siten, että koneoppiminen on tekoälyn osa-alue, ja syväoppiminen on eräs koneoppimistekniikka.

Viime vuosikymmenen aikana on saatu alustavaa näyttöä siitä, että koneoppimistekniikat kykenevät tunnistamaan epälineaariset rakenteet rahoitusmarkkinatiedoissa (ks. Huck 2009; Huck 2010; Moritz & Zimmermann 2014; Kraus, Do & Huck 2017; Atsalakis & Valavis 2009). Rahoituksessa tämä on johtanut koneoppimistekniikoiden laajamittaiseen hyödyntämiseen. Tekniikoita hyödynnetään anomalioiden havaitsemisessa, salkunhoidossa, luotonarvioinnissa, riskienhallinnassa, markkinoiden ja uutisten analysoinnissa sekä algoritmikaupassa. López de Pradon (2018, 14) mukaan monet taloudellisia operaatioita koskevat päätökset on tehtävä ennalta määriteltyjen sääntöjen perusteella. Täten koneoppimistekniikat ovat algoritmikaupankäynnin taustalla. Automaattinen algoritmikaupankäynti on vallannut rahoitusmarkkinat, muuttaen ne erittäin nopeiksi ja laajasti kytketyiksi tietojenvaihtoverkoiksi. Neuroverkot ovat eräs tällainen nopea ja tarkka koneoppimistekniikka, joka on saanut valtavasti huomioita rahoitustutkimuksissa.

3.2 Neuroverkot

Neuroverkot ovat alun perin kehitetty mallintamaan aivojen toimintaa. Mallintamisella on pyritty kaappaamaan aivojen laskennallinen toiminta, koska se poikkeaa perinteisten tietokoneiden laskentatavasta. Neuronimalli on yksinkertainen kuvaus tästä laskentatavasta. Mallissa neuroni on laskentayksikkö, joka prosessoi annettua informaatiota. Tapauksessa, jossa neuroverkko sisältää vain yhden neuronin käytetään käsitettä perseptroni. (Haykin 2009, 1.)

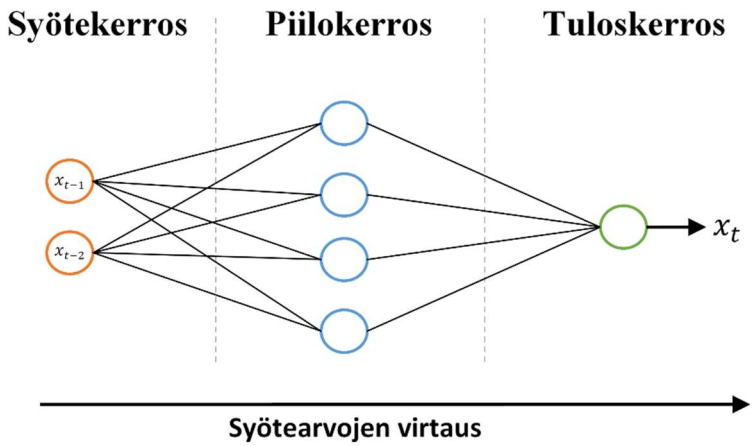
Todellisuudessa aivojen hermoverkko on niin kompleksisinen, että neuroverkot eivät kykene esittämään tai selittämään täydellisesti aivojen toimintaa. Lisäksi neuroverkot sisältävät laskennallisia ominaisuuksia, jotka eivät ole linjassa biologisten hermoverkkojen kanssa. Tästä syystä aiheelle läheiset tutkimukset voidaan jakaa laskennalliseksi neurotieteeksi sekä neuroverkkotieteeksi. Laskennallisessa neurotieteessä hyödynnetään neuroverkkoja, kun tutkitaan biologisten hermoverkkojen ominaisuuksia. Neuroverkkotieteessä tutkitaan ja kehitetään järjestelmiä, jotka kykenevät oppimaan monimutkaisia toimintoja, huolimatta ovatko ne linjassa biologisten hermoverkkojen kanssa. Tutkielmassa käytetään jälkimmäistä jaottelua. (Mostafa ym. 2017, 51.)

Neuroverkkojen laskennallinen tehokkuus syntyy niiden rakenteesta sekä oppimisetä yleistämiskyvystä. Niiden rakenne voidaan jakaa kolmeen luokkaan:

- Yksikerroksiset eteenpäin syöttävät neuroverkot
- Monikerroksiset eteenpäin syöttävät neuroverkot
- Takaisinkytketyt neuroverkot

Kaikkia näiden luokkien neuroverkkoja yhdistää informaation eteenpäin syöttäminen (engl. feed forward), joka tarkoittaa informaation etenevää virtausta syötekerroksesta (engl. input layer) kohti tuloskerrosta (engl. output layer). Neuroverkot rakentuvat syöte-, piilo- ja tuloskerroksesta. Yksikerroksiset neuroverkot rakentuvat ainoastaan syöte- ja tuloskerroksesta. Neuroverkkojen kaikki kerrokset sisältävät lukumäärältään yhden tai useamman solmun (engl. node). Solmuista käytetään erilaisia nimityksiä ja ne vaihtelevat sen mukaan, mikä kerros on kyseessä. Syötekerros sisältää solmuja, joita kutsutaan syötteiksi (engl. inputs), joiden kautta aineiston arvot eli havainnot etenevät seuraavan kerroksen solmuille. Piilokerroksien (engl. hidden layers) sekä tuloskerroksen solmut ovat neuroneja (engl. neurons). Ero näiden kahden kerrostyypin neuronien välillä selittyy niiden laskennallisista ominaisuuksista. Täten piilokerrosten neuroneja kutsutaan piiloyksiköiksi (engl. hidden units tai hidden neurons) ja tuloskerroksen neuroneja ulostuloiksi (engl. outputs). Neuroverkoissa piilokerrosten lukumäärä voi olla yksi tai useampi. Tapauksessa, jossa neuroverkot sisältävät useamman piilokerroksen puhutaan syväoppimisesta (engl. deeplearning), koska verkkojen oppiminen tapahtuu näissä kerroksissa. Aiheen kirjallisuudessa kerroksiksi lasketaan vain neuroneja sisältävät kerrokset, joissa laskenta suoritetaan. (Haykin 2009, 21–23.)

Neuroverkkojen sanotaan olevan täysin kytkettyjä (engl. fully connected), kun kaikki solmut eri kerroksien välillä on kytketty toisiinsa. Tällöin esimerkiksi piilokerroksen solmuihin virtaa informaatio kaikista edellisen kerroksen solmuista. Kuviossa 3 on havainnollistettu esimerkin kaltaista tilannetta, jossa on täysin kytketty eteenpäin syöttävä neuroverkko.



Kuvio 3 Kaksikerroksinen eteenpäin syöttävä neuroverkko (MLP).

Kuvion 3 syötekerros sisältää kaksi syötettä, jonka kautta syötteet etenevät kaikille piilokerroksen neljälle piiloyksikölle. Viimeisenä kerroksena on tuloskerros yhdellä ulostulolla, joka on kytketty edellisen kerroksen piiloyksiköihin.

Toisinaan neuroverkoissa saatetaan käyttää kerroksien välillä osittaista kytkentää (engl. dropout), jolloin kytkennät solmujen välillä vaihtelevat satunnaisesti. Tällaisella kytkennällä voidaan tavoitella verkon parempaa oppimiskykyä, ja se on eräs neuroverkon hyperparametreista. Neuroverkkojen oppimiskykyyn voidaan vaikuttaa vaihtamalla piilokerrosten sekä neuronien lukumäärää. Ne ovat myös verkon hyperparametreja, joita käsitellään tarkemmin luvussa 4. Neuroverkkojen rakenne on suoraan yhteydessä niiden oppimiskykyyn.

Neuroverkon oppiminen eli kouluttaminen voi olla joko ohjattua (engl. supervised learning) tai ohjaamatonta oppimista (engl. unsupervised learning). Ohjaamaton oppiminen rajataan tutkielman ulkopuolelle. Neuroverkon ohjatussa oppimisessa verkolle annetaan syöte-tavoite-pareja. Syötteistä verkko laskee aluksi satunnaisia ulostuloarvoja. Näitä ennustearvoja verrataan tavoitearvoihin laskemalla näiden väliset virheet eri ajanhetkillä. Virheiden perusteella verkko optimoi funktion parametreja, joilla syötteet kuvautuvat ulostuloarvoiksi. Parametrien optimointi on ulostulo- ja tavoitearvojen välisten virheiden minimointia. (Haykin 2009, 36–37.)

Ohjattua oppimista voidaan verrata tilastotieteessä käytettävien menetelmien mallinnusvaiheeseen, jolloin menetelmän parametrit optimoidaan aineistoon sopiviksi, jotta saadaan aikaiseksi sen paras mahdollinen ennustamiskyky. Tällöin aineistosta käytetään sitä jaksoa, jota kutsutaan englanninkielisessä kirjallisuudessa in-sample, kun taas

aineiston out-of-sample jaksoa käytetään mallien ennustamisessa. Jälkimmäinen jakso on se osa aineistosta, jota ei ole käytetty mallinnusvaiheessa parametrien optimointiin.

Neuroverkkojen ennustamiskyky syntyy sen yleistämiskyvystä. Verkon yleistämiskyvyllä tarkoitetaan kohtalaisten ulostuloarvojen tuottamista, kun syötearvot on otettu aineiston out-of-sample jaksosta. Neuroverkkojen yleistämiskyvyn ominaisuuksia ovat:

- Funktion muodostaminen, jolla syötearvot kuvautuvat ulostuloarvoiksi.
- Adaptiivisuus, joka mahdollistaa verkon toiminnan epästationaarisilla aikasarjoilla.
- Epälineaarisuuksien mallintaminen, kun aikasarjaa ajava prosessi on epälineaarinen. (Haykin 2009, 2–4).

Neuroverkot ovat läheistä sukua ei-parametrisille menetelmille (engl. nonparametric methods). Tämä on kollektiivinen sana suurelle joukolle tekniikoita, joilla voidaan arvioida esimerkiksi kahden havainnon välinen suhde määrittelemättä sille tiettyä parametrimuotoa. (Franses & Dijk 2000, 210.)

Yleistämiskyvyn ominaisuudet ovat tehneet neuroverkkoista suositun menetelmän taloudellisen aikasarjojen ennustamisessa, mikä näkyy rahoitustutkimuksissa niiden lisääntyneenä käyttönä. Hawley ym. (1990) mainitsevat artikkelissaan neuroverkkojen mahdollisia käyttökohteita, joita vielä ei ole nähty rahoitustutkimuksissa. Ne ovat kohteita, jotka liittyvät yritysrahoitusten, rahoitusinstituutioiden sekä sijoitusammattilaisten tehtäviin. Tehtäviä ovat esimerkiksi taloudellinen simulointi, ennustaminen, arvioin-ti, listautumisannin hinnoittelu, arbitraasi mahdollisuuksien tunnistaminen, turvallisuusriskien profilointi ja veronkiertäjien löytäminen.

Toisaalta neuroverkoilla on myös puutteita. Verkkojen suurin puute on, että tulosten muodostuminen ei ole läpinäkyvää (engl. black box). Esimerkiksi tätä prosessia on mahdotonta purkaa osiin, ja tämän seurauksena on vaikea sanoa, kuinka tulokset on saavutettu. Eräs verkkojen heikkous on, että ne saattavat yli- tai alisovittaa aineistoa. Tätä puutetta voidaan ehkäistä kokeile ja erehdy -taktiikalla, johon palataan luvussa 4. Neuroverkot kärsivät optimaalisen kouluttamisalgoritmin puuttumisesta, joka takaisi optimaalisen ulostulo- ja tavoitearvovirheiden minimin löytämisen. Lisäksi neuroverkkojen rakentamiselle ei ole yleispätevää teoriaa, joten niiden rakentaminen on haasteellista ja niiden tuottamien tulosten vertailu muihin tutkimuksiin on hankalaa. (Hamid & Iqbal 2004.)

3.2.1 Perseptroni

Ensimmäiset neuroverkot olivat hyvin yksinkertaisia, havaintoja eteenpäin syöttäviä, jotka sisälsivät vain syötekerroksen ja yhden neuronin tuloskerroksessa. McCulloch ja Pitts (1943) esittelevät artikkelissaan mallin, jolla he kuvaavat aivojen hermosolun eli neuronin toimintaa. Rosenblatt (1958) kehitti tämän mallin ympärille perseptronin, jolla on ollut keskeinen merkitys neuroverkkojen kehityksen historiassa. Perseptroni on yksikerroksinen neuroverkko, ja sitä on alun perin sovellettu luokittelua koskeviin ongelmiin. Luokittelu voidaan jakaa kahteen vaiheeseen. Ensimmäisessä vaiheessa syötearvoista lasketaan niiden painotettu summa, johon lisätään neuronin vakiotermi b. Tämä vaihe lasketaan kaavalla:

$$f(x) = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{w}_i + b, \tag{3.1}$$

missä syötevektori $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$, painovektori $\mathbf{w} = (w_1, w_2, \dots, w_n) \in \mathbb{R}^n$ ja b on vakiotermi. Funktion $f(x)$ arvoa voidaan sanoa aktivointifunktion kynnysarvoksi s. Toisessa vaiheessa perseptronin tulosarvo $y(x)$ määräytyy aktivointifunktiosta

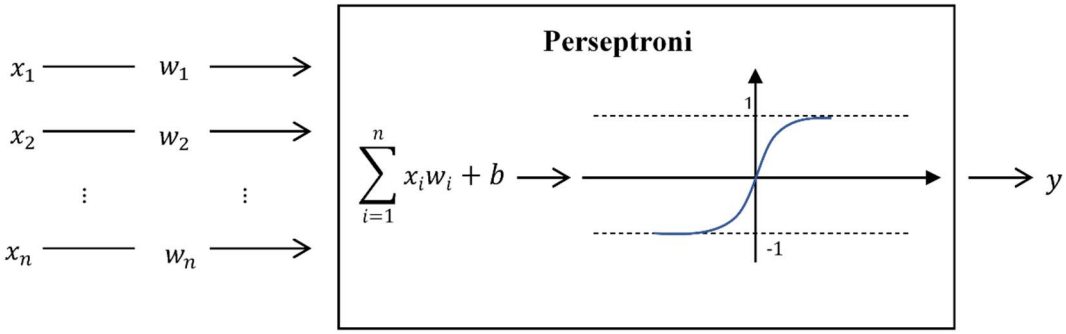
$$h(s) = \begin{cases} 1, & \text{jos s} > 0 \\ 0, & \text{jos s} \leq 0, \end{cases} \tag{3.2}$$

missä $h(s)$ on yksikköporrasfunktio. Perseptronin funktio

$$y(x) = f\big(\, h(s)\big), \tag{3.3}$$

missä y: $\mathbb{R}^n \rightarrow \{0,1\}$. Perseptroni kykenee kaavan 3.2 luokitteluun, kun syötejoukko on lineaarisesti eroteltavissa, ja tämä ehto on seurausta aktivointifunktion valinnasta. (Tuominen 2020.)

Aktivointifunktio voi vaihdella eri neuroverkkojen välillä, ja monikerroksisissa neuroverkoissa se voi vaihdella eri kerroksien välillä. Tuomisen (2020) mukaan aktivointifunktioiden tehtävä on muuttaa lineaariset syötearvot epälineaarisiksi, jotta verkkojen käyttöön saataisiin ominaisuuksia kuten epälineaarisuus ja jatkuvasti derivoituvuus. Kuvio 4 esittää tilannetta, kun aktivointifunktio on hyperbolinen tangentti.



Kuvio 4 Perseptronin toiminta (mukaellen Mostafa ym. 2017, 52).

Kuvio 4 on piirretty uudelleen käyttäen hyväksi alkuperäistä lähdettä. Kuvion 4 aktivointifunktio on hyvä esimerkki funktiosta, joka mahdollistaa epälineaarisen mallinnuksen. Lisäksi kuvion 4 aktivointifunktio mahdollistaa, että arvot ovat jatkuvia välillä [-1,1]. Tutkielmassa käytetään seuraavista aktivointifunktioista kahden jälkimmäisen lisäksi lineaarista aktivointifunktiota:

- Sigmoid-funktiolla $\sigma: \mathbb{R} \to [0, 1]$,

$$\sigma(x) = \frac{1}{1 + e^{-ax}}. \tag{3.4}$$

- Hyperbolisella tangentti funktiolla $\tanh: \mathbb{R} \to [-1, 1]$,

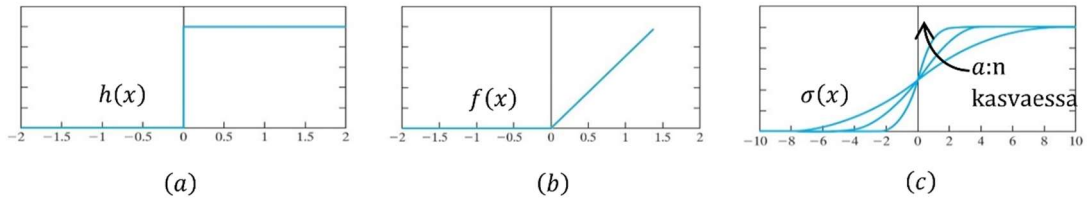
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{3.5}$$

- ReLu-funktiolla $f(x): \mathbb{R} \to [0, \infty[$,

$$f(x) = \max\{0, x\}. \tag{3.6}$$

Sigmoid-funktio on edellä esitetty, koska hyperbolinen tangentti funktio eroaa edellisestä vain skaalauksen osalta. Sigmoid-funktion arvot rajoittuvat välille [0,1]. (Tuominen 2020.)

Kuviossa 5 on esitetty yksikköporras-, relu- ja sigmoid-funktioiden kuvaajat. Kohdassa c on havainnollistettu, kuinka kaavan 3.4 a:n arvot vaikuttavat sigmoid-funktion jyrkkyyteen.



Kuvio 5 Aktivointifunktiot (mukaellen Haykin 2009, 13).

(a) yksikköporrasfunktio (Heavisiden funktio), (b) ReLu-funktio (engl. Rectified Linear Unit) ja (c) Sigmoid-funktio (logistinen funktio)

Kuvio 5 on piirretty uudelleen käyttäen hyväksi alkuperäistä lähdettä lisäten siihen a:n vaikutus.

3.2.2 Monikerroksinen perseptroni (MLP)

Neuroverkkoja käsittelevän aiheen innostus pysähtyi, kun Minsky ja Papert (1969) julkaisivat aihetta käsittelevän kirjan vuonna 1969. Kirjassaan tutkijat väittävät, että heidän perseptronille löytämät matemaattiset rajoitukset pätevät myös sen muunnelmille, joita ovat monikerroksiset neuroverkot. Aihe heräsi uudelleen henkiin 1980-luvun puolessa välissä, kun vastavirta-algoritmi (engl. back-propagation algorithm) kehitettiin ja otettiin käyttöön monikerroksissa neuroverkoissa (Haykin 2009, 124).

Monikerroksinen perseptroni (engl. multi-layer preceptron) on yleistetty versio perseptronista. Tutkielmassa verkosta käytetään lyhennettä MLP. MLP-verkkoa on käytetty menestyneesti ratkaisemaan vaikeita ja monimutkaisia ongelmia. Verkon menestys perustuu sen perusominaisuuksiin, jotka voidaan tiivistää seuraavasti:

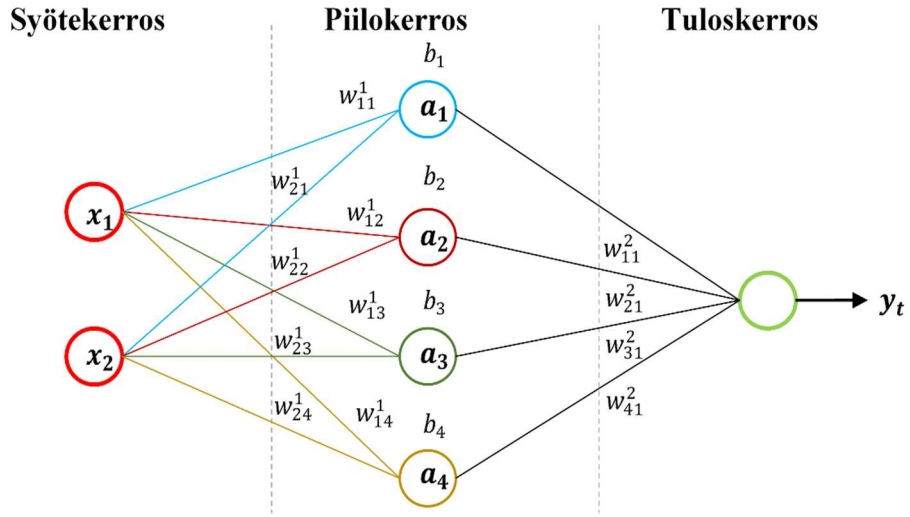
- Verkon neuronit käyttävät epälineaarisia aktivointifunktioita, jotka ovat jatkuvasti derivoituvia.
- Verkko sisältää yhden tai useamman piilokerroksen.
- Verkossa kerroksien väliset solmut ovat laajasti kytkettyjä toisiinsa, ja niiden laajuutta määrittelevät synapsiset painot eli solmujen väliset painot.

MLP-verkon hyödyt syntyvät perusominaisuuksista, mutta ne asettavat myös haasteen nähdä tulosarvojen muodostuminen. Aktivointifunktioiden epälineaarisuus sekä solmujen laajat kytkennät tekevät yhdessä verkon teoreettisen analyysin hankalaksi. Lisäksi piiloyksiköt tekevät verkon oppimisprosessin visualisoinnin vaikeaksi. Täten MLP-verkon oppimisprosessi ei ole yksikäsitteinen, koska verkon on muodostettava useita mahdollisia

funktioita, joilla syötearvot kuvautuvat ulostulosarvoiksi sekä päätös on tehtävä näiden funktioiden väliltä. (Haykin 2009, 123.)

Bishopin (2006, 229) mukaan keskeinen ero perseptronin ja MLP-verkon välillä selittyy sillä, että MLP käyttää piiloyksiköissään jatkuvia sigmoid-funktion epälineaarisuuksia, kun taas perseptroni käyttää yksikköporrasfunktion epälineaarisuuksia. Sigmoid-funktion käyttö tarkoittaa, että MLP-verkkoa kuvaava funktio on jatkuvasti derivoituva verkon parametrien kuten solmujen välisten painojen sekä vakiotermien suhteen, mikä on keskeistä verkon kouluttamiselle. Eräs suosittu algoritmi, jota käytetään neuroverkkojen kouluttamiseen, on vastavirta-algoritmi, joka esitellään luvussa 3.3.

Kuviossa 6 on esitelty MLP-verkon rakenne, jossa on kaksi kerrosta, kaksi syötettä, neljä piiloyksikköä ja yksi ulostulo.



Kuvio 6 MLP-verkon rakenne (mukaellen Tuominen 2020).

Kuvio 6 on piirretty uudelleen käyttäen hyväksi alkuperäistä lähdettä. Kuvion 6 syöte on vektori $\mathbf{x} = (x_1, x_2)\ \epsilon\mathbb{R}^2$. Kerroksien painot ovat w_{ij}^l, jossa yläindeksi l on verkon kerros, alaindeksit i ja j ovat solmuja joihin painot kiinnittyvät. Esimerkiksi kuvion 6 ensimmäisen kerroksen paino on w_{11}^1 ja se ja se kiinnittyy syötesolmuun x_1 sekä piiloyksikköön a_1. Kun taas w_{21}^2 on toisen kerroksen paino, joka kiinnittyy piiloyksikköön a_2 ja ulostuloon y_t. Tapauksessa, jossa ulostuloja on useita niin alaindeksi $j > 1$. Vakiotermi voidaan esittää muodossa b_j^l, jossa l on kerros ja j on kerroksen neuroni. Syötteet verkon solmuille voidaan esittää muodossa a_j^l, jossa l on kerros ja j on solmu. Kuvion 6 tapauksessa, alkio $x_1 = a_1^0$, koska

$$a_j^l = \varphi(z_j^l) = \varphi(\sum_{i=1}^{N_{l-1}} w_{ij}^l \, a_i^{l-1} + b_j^l), \tag{3.7}$$

kun

$$z_j^l = \sum_{i=1}^{N_{l-1}} w_{ij}^l \, a_i^{l-1} + b_j^l. \tag{3.8}$$

Kaavoissa 3.7 ja 3.8 N_{l-1} on l kerroksen solmujen lukumäärä, z_j^l on l kerroksen neuronin j summa ja φ on käytetty aktivointifunktio, joka voi vaihdella kerroksien välillä. Tuomisen (2020) mukaan piiloyksiköt voidaan tulkita funktioiksi: $f_j^l: \mathbb{R}^{N_{l-1}} \rightarrow \mathbb{R}^{N_{l+1}}$, $j \epsilon \{1,2, \dots, N\}$,

$$f_j^l(v) = \left(\varphi_l\left(g_j^l(v)\right), \dots, \varphi_l\left(g_j^l(v)\right)\right), \tag{3.9}$$

missä g_j^l on yleensä edellisen kerroksen painotettu summa, johon on lisätty vakiotermi eli

$$g_j^l(v) = \sum_{i=1}^{N_{l-1}} w_{ij}^l \, v_i + b_j^l, \tag{3.10}$$

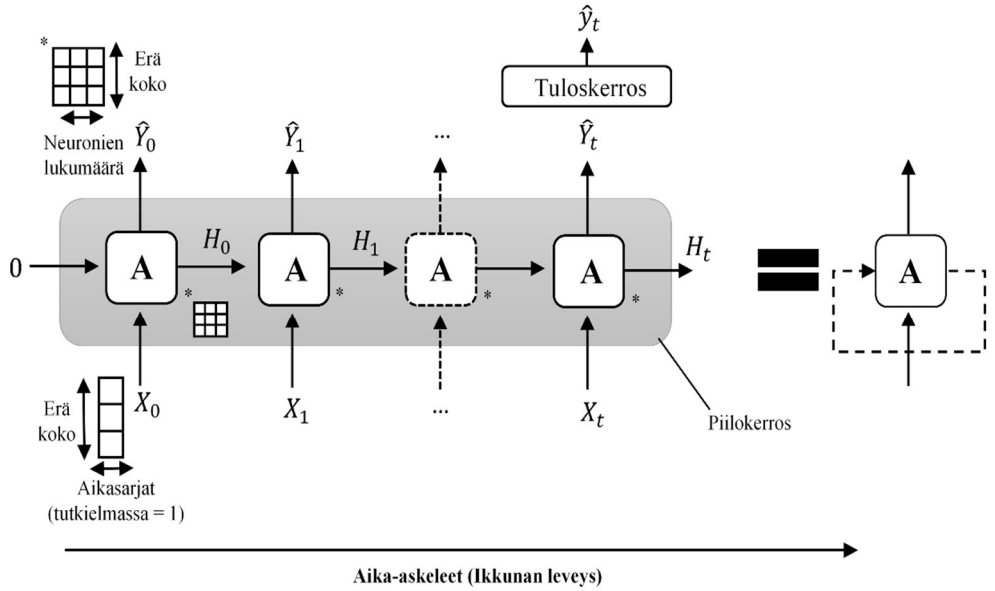
ja φ_l on kerroksen l aktivointifunktio. Tuloskerroksen funktiolle f_j^L arvojoukko on $\mathbb{R}$. MLP-verkkoon liittyviä määritelmien ja merkintöjen esittämistä perustellaan sillä, että ne luovat lähtökohdan esittää vastavirta-algoritmi luvussa 3.3.

3.2.3 Takaisinkytketty neuroverkko (RNN)

Takaisinkytketty neuroverkko (engl. recurrent neural network, lyh. RNN) voidaan nähdä MLP-verkon laajennuksena, missä piilokerros sisältää takaisinkytkennän. Takaisinkytkentä mahdollistaa, että RNN-verkko hyödyntää koko syötearvojen historiaa jokaisen ulostuloarvon laskemisessa, kun taas MLP-verkko voi laskea vain annetuista syötearvoista niitä vastaavat tulosarvot (Graves 2012, 20). Tämä tarkoittaa, että RNN-verkolla on muistiominaisuus, jossa menneet syötearvot vaikuttavat ulostuloarvojen muodostamiseen.

Täten RNN-verkko soveltuu sekvenssimuotoisen informaation prosessointiin. Verkon rakenne voidaan nähdä useana kopiona yhdestä verkosta, joka toimii silmukka periaatteella. Kuviossa 7 on esitetty RNN-verkon toiminta. Kuvion vasemmassa reunassa ovat syötearvovektori $\mathbf{X}_0$ ja tulosarvomatriisi $\widehat{\mathbf{Y}}_0$ sekä niiden dimensioihin vaikuttavat tekijät. Neuroverkot edellyttävät saavansa syötearvot tietyssä muodossa, jota tarkastellaan

tarkemmin luvussa 4. RNN-verkon kouluttaminen tapahtuu samoin kuin MLP-verkolla. Kouluttamisessa kuvion 7 ensimmäinen muistisolu A ottaa syötteinä syötevektorin $\mathbf{X}_0$ sekä luvun 0. Luku 0 tarkoittaa, että aiemmilla syötteillä ei ole vielä vaikutusta tulosarvomatriisin $\widehat{\mathbf{Y}}_0$ muodostamiseen. Kun ensimmäinen muistisolu A on laskenut tulosarvomatriisin $\widehat{\mathbf{Y}}_0$, kopioituu tämä matriisi syötteelle $\mathbf{H}_0$, joka toimii syötteenä seuraavalle muistisolulle A syötevektorin $\mathbf{X}_1$ lisäksi. Tulosarvomatriisien kopioitumisesta muistisolujen syötteeksi syntyy RNN-verkon muistiominaisuus, koska edellisen aika-askeleen syötearvot vaikuttavat matriisien **H** kautta seuraavien tulosarvomatriisien muodostamiseen.



Kuvio 7 RNN-verkon rakenne ja toiminta, kun verkko sisältää yhden piilokerroksen (mukaellen Coursera 2020).

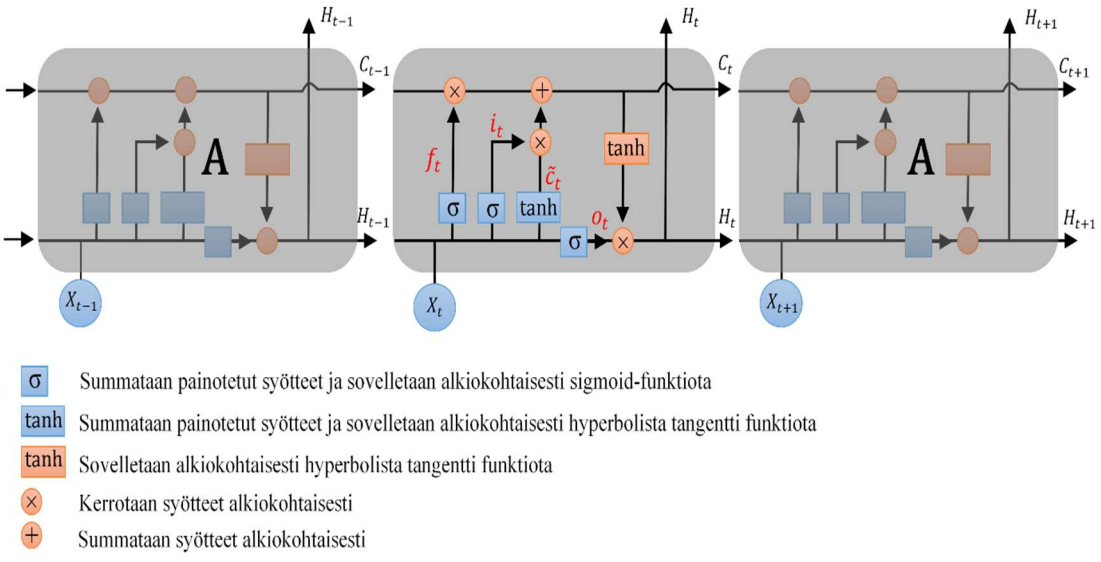
Kuvio 7 on piirretty uudelleen käyttäen hyväksi alkuperäistä lähdettä lisäten siihen suomenkielisiä lisäkuvauksia. Kuvion 7 viimeinen muistisolu A laskee tulosarvomatriisin $\widehat{\mathbf{Y}}_t$, joka etenee tuloskerroksen neuronille, joka tuottaa ulostuloarvon.

Kritiikki RNN-verkkoa kohtaan johtuu sen rakenteen rajallisuudesta. Esimerkiksi aika-askelien määrän kasvaessa riittävän suureksi verkko ei kykene kuvaamaan syötearvoista ulostuloarvoja. Kirjallisuudessa tätä ongelmaa kutsutaan häviäväksi gradientiksi (engl. vanishing gradient problem), jota ovat tutkineet Hochreiter ym. (2001). Ongelma syntyy kouluttamisessa, jolloin virheiden arvot räjähtävät tai katoavat. Tämän seurauksena verkko ei kykene optimoimaan solmujen välisiä painoja ja kouluttaminen epäonnistuu.

3.2.4 Pitkäkestoinen työmuisti (LSTM)

Pitkäkestoinen työmuisti (engl. long short term memory, lyh. LSTM) on kehitetty RNN-verkon häviävän gradientti ongelman ratkaisemiseksi. Tämä ongelma tarkoittaa, että RNN-verkko ei kykene huomioimaan aineiston pitkän aikavälin riippuvuuksia (Hochreiter ja Schmidhuber 1997). Ongelman ratkaisemiseksi on ehdotettu useita vaihtoehtoja, mutta parhaiten tähän ongelmaan on vastannut LSTM-verkko. Verkko kykenee huomioimaan aineiston pitkän aikavälin riippuvaisuudet paremmin kuin RNN-verkko johtuen sen kehittyneemmästä muistisolusta. LSTM-verkon muistisolut toimivat kuten tietokoneen muistipiirit.

Kuviossa 8 on esitetty LSTM-verkon rakenne, jossa yhtä muistisolua vastaa lohko A. Muistisolu rakentuu neljästä yksiköstä, joita ovat syöteportti (engl. input gate), tulosportti (engl. output gate), unohdeportti (engl. forget gate) ja takaisinkytketty neuroni (engl. self-recurrent neuron). Portit kontrolloivat muistisolun sekä viereisten solujen keskinäistä vuorovaikutusta. Syöteportti ohjaa sitä, että muuttaako muistisoluun tuleva signaali solun tilaa. Toisaalta tulosportti voi ohjata muistisolun tilaa sen suhteen, muuttuuko toisen muistisolun tila. Lisäksi unohdeportti voi valita muistaako vai unohtaako edellisen tilansa. Kuviossa 8 syötevektori on $\mathbf{X}_t$ ajanhetkellä t, jonka muistisolu saa syötteenä. Muistisolun laskema tulosarvo hetkellä t on H_t, joka kopioituu seuraavalle muistisolulle syötteenä H_t. i_t ja $\widetilde{c_t}$ ovat syöteportin ja muistisolun ehdokastilaa kuvaavat arvot ajanhetkellä t. f_t ja C_t ovat unohdeportin ja muistisolun ehdokastilaa kuvaavat arvot ajanhetkellä t. o_t ja H_t ovat tulosportin ja muistisolun arvot ajanhetkellä t. (Bao ym. 2006.)



Kuvio 8 LSTM-verkon rakenne ja toiminta (mukaellen Colah's Blog 2015).

Kuvio 8 on piirretty uudelleen käyttäen hyväksi alkuperäistä lähdettä lisäten siihen suomenkielisiä lisäkuvauksia. Kuvion 8 vasemmassa alareunassa on kuvattu ne matemaattiset operaatiot, joita muistisolut käyttävät tulosarvojen laskemisessa. Matemaattisia operaatioita on kahdenlaisia, joita kuvataan punaisella ja sinisellä väreillä. Punaiset operaatiot edustavat toimintoja, kuten vektorien lisäys, kun taas siniset operaatiot ovat LSTM-verkon piilokerrosten oppimista. (Colah's Blog 2015.)

3.3 Vastavirta-algoritmi

Vastavirta-algoritmi oli virstanpylväs neuroverkkojen kouluttamiselle ja niiden laskentatehon hyödyntämiselle. Algoritmin toiminta voidaan jakaa kahteen vaiheeseen:

1. Etenevässä virtausvaiheessa solmujen väliset painot ovat niiden alustamisen jälkeen vakioita, ja informaatio virtaa verkon lävitse kerros kerrokselta, kunnes se saavuttaa tulosarvon. Täten tämän vaiheen muutokset rajoittuvat ainoastaan verkon neuronien aktivointipotentiaaliin sekä niiden ulostuloihin.
2. Vastavirta vaiheessa tavoitetulosta verrataan etenevän virtausvaiheen ulostuloon, joka on luonteeltaan satunnainen johtuen painojen satunnaisesta alustamisesta. Näiden tuloksien välinen virhe levitetään vastakkaiseen suuntaan kuin etenevässä virtauksessa. Levittämisessä verkon solmujen välisiin painoihin tehdään säätöjä, joiden tavoitteena on säätää painot siten, että tulosten välinen virhe on mahdollisimman pieni. Syöte- ja tuloskerrosten kohdalla säätöjen laskeminen on suoraviivaista, mutta piilokerrosten kohdalla se on paljon haastavampaa. (Haykin 2009, 124.)

Tutkielmassa esitetään ainoastaan matemaattisesti MLP-verkon vastavirta-algoritmi. Valintaa perustellaan sillä, että valitun verkon algoritmi on yksinkertaisempi esittää kuin RNN- tai LSTM-verkojen vastaavat algoritmit. Varsinkin LSTM-verkon muistisolujen portit tekevät matemaattisen esittämisen haasteelliseksi. Lisäksi MLP-verkon vastavirta-algoritmin esittely antaa lähtökohdan ymmärtää, kuinka verkot oppivat.

Neuroverkko koulutetaan antamalla sille aineistosta muodostettuja syöte-tavoite-pareja. Syötteistä verkko laskee tuloksia, joita verrataan tavoitearvoihin käyttämällä valittua virhefunktiota. Verkon kouluttaminen tähtää virhefunktion arvon minimoimiseen muuttamalla verkon painoja ja vakiotermejä esimerkiksi vastavirta-algoritmin avulla. Eräs keino minimoida virhefunktiota on gradienttimenetelmä (engl. gradient descent). Menetelmä vaatii virhefunktion osittaisderivaatat kaikkien painojen ja vakiotermien suhteen. Vastavirta-algoritmilla saadaan laskettua nämä osittaisderivaatat, jonka jälkeen

gradienttimenetelmä kertoo, mihin suuntaan funktion parametreja tulisi muuttaa, jotta virhefunktion arvo minimoituu. Virhefunktion minimointi on iteratiivinen prosessi, jossa lasketaan useita osittaisderivaattoja eri kerroksien funktion parametreille, jonka seurauksena algoritmin toiminta on usein hidasta. Neuroverkon kouluttaminen sisältää seuraavat vaiheet:

1. Verkolle syötetään koulutusaineistosta muodostetut syöte-tavoite-parit.
2. Verkon neuronikohtaiset summat z_j^l ja ulostulot a_j^l lasketaan.
3. Virhefunktion osittaisderivaatat lasketaan kerroskerrokselta niitä vastaavalla syötteellä.
4. Neuronien parametreja säädetään gradienttimenetelmän avulla. (Tuominen 2020.)

Neuroverkkojen kouluttamisessa käytetään usein virhefunktiona keskineliövirhettä (engl. Mean Squared Error, lyh. MSE). MSE lasketaan kaavalla

$$E_A = \frac{1}{2N}\sum_{x\in A}\|y(x) - \hat{y}(x)\|^2, \tag{3.11}$$

missä tavoitearvo on y, ulostuloarvo on $\hat{y}$ ja N on syötejoukon A lukumäärä. Vastavirta-algoritmia käytetään virhefunktion E osittaisderivaattojen $\frac{\partial E}{\partial w}$ ja $\frac{\partial E}{\partial b}$ laskemiseen. Osittaisderivaatat lasketaan kerroskerrokselta verkon kaikille painoille ja vakiotermeille. Tuloskerroksen osittaisderivaatat lasketaan painojen w_j^L ja vakiotermien b_j^L suhteen, joissa yläindeksi viittaa verkon L. kerrokseen. Osittaisderivaatat painojen suhteen tuloskerroksen neuronille j saadaan kaavalla

$$\frac{\partial E}{\partial w_{ij}^L} = \delta_j^L a_j^{L-1}, \tag{3.12}$$

missä j indeksiin liittyvää deltaa δ_j^L eli virhettä merkitään usein

$$\delta_j^L = \left(y_j - \hat{y}_j\right)\varphi'(z_j^L), \tag{3.13}$$

mikä saadaan laskettua kaavalla

$$\delta_j^L = \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L}\varphi'\left(z_j^L\right). \tag{3.14}$$

Osittaisderivaatat vakiotermien suhteen tuloskerroksen neuronille j, saadaan kaavalla

$$\frac{\partial E}{\partial b_j^L} = \frac{\partial E}{\partial z_j^L}\frac{\partial z_j^L}{\partial b_j^L} = \delta_j^L. \tag{3.15}$$

Osittaisderivaatat painojen suhteen piilokerroksen l neuronille j, saadaan kaavalla

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{ij}^l} = \delta_j^l a_i^{l-1}, \tag{3.16}$$

missä derivaatan sekä osittaisderivaatan ketjusääntöä että kaavoja 3.7 ja 3.8 käyttäen saadaan

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} = \sum_{k=1}^{N_{l+1}} \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} =$$

$$\sum_{k=1}^{N_{l+1}} \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_k^l} = \sum_{k=1}^{N_{l+1}} \delta_k^{l+1} w_{jk}^{l+1} \varphi'(z_j^l), \tag{3.17}$$

joten

$$\frac{\partial E}{\partial w_{ij}^l} = a_i^{l-1} \varphi'(z_j^l) \sum_{k=1}^{N_{l+1}} \delta_k^{l+1} w_{jk}^{l+1}. \tag{3.18}$$

Vastaavasti osittaisderivaatat vakiotermien suhteen piilokerroksen l neuronille j, saadaan kaavalla

$$\frac{\partial E}{\partial b_j^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \times 1 \tag{3.19}$$

ja

$$\frac{\partial E}{\partial b_j^l} = \varphi'(z_j^l) \sum_{k=1}^{N_{l+1}} \delta_k^{l+1} w_{jk}^{l+1}. \tag{3.20}$$

Edellä esitetyt kaavat havainnollistavat, miksi virhefunktion valinta on tärkeä. Esimerkiksi, jos derivoinnin tulokset ovat pieniä niin verkon parametrit oppivat vähän ja neuronien oppiminen on hidasta. (Tuominen 2020.)

Virhefunktion minimin etsiminen alkaa gradienttimenetelmässä painoille, jotka ovat satunnaisesti alustettu. Menetelmässä etsitään globaali minimi konvekseille virhefunktioille ja lokaali minimi ei-konvekseille virhefunktioille. Virhefunktion gradientti kertoo nopeimman kasvusuunnan painoille, kun taas tämän vastavektori nopeimman vähenemisen suunnan. Verkon oppimisnopeutta säädetään parametrilla α, joka on käytännössä sopivien askelien ottamista kohti virhefunktion minimiä. Yksittäiselle syötteelle neuronien uudet parametrit ovat

$$w_{ij}^l \rightarrow w_{ij}^l - \alpha \frac{\partial E}{\partial w_{ij}^l} \tag{3.21}$$

ja

$$b_j^l \rightarrow b_j^l - \alpha \frac{\partial E}{\partial b_j^l}. \tag{3.22}$$

Gradienttimenetelmiä on monia, ja eräs tällainen menetelmä on stokastinen gradienttimenetelmä (engl. Stochastic Gradient Descent, lyh. SGD), jota tutkielmassa käytetään. Menetelmässä lasketaan virhefunktion arvo ja neuronien painot päivitetään jokaisen syötejoukon jälkeen. Menetelmän etuja ovat nopea tieto verkon oppimisen tilasta, koska verkko oppii käytännössä koko ajan. Menetelmän heikkouksia ovat verkon hidas oppiminen. Syy tähän johtuu laskemisen suuresta määrästä sekä siitä, että parametrien painojen arvot saattavat heilua niiden päivittämisen aikana, mikä johtaa hitaaseen virhefunktion minimin löytämiseen. Toisaalta parametrien heiluminen voi johtaa pienempään minimiin kuin perinteisessä gradienttimenetelmässä, missä jokaista syötettä kohden lasketaan virhefunktio. Parametrien päivitys tapahtuu vasta, kun kaikki syötteet on käyty läpi, jolloin parametrit päivitetään virheiden kokonaissumman perusteella. Minimierä gradienttimenetelmä (engl. mini batch gradient descent) on kahden edellisen menetelmän välimuoto. Menetelmässä syötteet jaetaan osajoukkoihin, jonka jälkeen ne syötetään verkolle joukkoina virhefunktion laskemiseksi ja parametrien päivittämiseksi. Tämä joukkoihin jakaminen vähentää parametrien heilumista, joka tarkoittaa vakaampaa virhefunktion pienemistä kohti minimiä. (Tuominen 2020.)

Käsitteisiin yli- ja alisovittaminen (engl. under- and overfitting) viitataan, kun neuroverkkojen kouluttamisessa ja ennustamisessa ilmenee ongelmia. Ylisovittaminen on kyseessä, kun verkko toimii hyvin kouluttamisessa, mutta ei kykene ennustamaan. Tapauksessa, jossa virhefunktion pieneminen hidastuu ja sen tarkkuus huononee kouluttamisen edetessä, on kyse alisovittamisesta. Ylisovittamista voidaan ehkäistä kasvattamalla opetusjoukkoa keskeyttämällä kouluttaminen tarpeeksi aikaisin tai käyttämällä osittaista kytkentää kerroksien välillä. Alisovittamista voidaan ehkäistä kokeile ja erehdy -taktiikalla sekä seuraamalla verkon oppimiskäyrää, josta tutkielmassa käytetään kouluttamiskäyrä käsitettä. (Tuominen 2020.)

3.4 Kirjallisuuskatsaus

Atsalakis ja Valavanis (2009) esittävät artikkelissaan valtavan katsauksen tutkimuksista, joissa älykkäitä ja oppivia järjestelmiä (engl. soft computing) on sovellettu osakemarkkinoiden ennustamiseen. Älykkäät ja oppivat järjestelmät ovat menetelmiä, jotka ovat syntyneet perinteisen matemaattis-tilastollisen mallinnuksen rinnalle ja joihin neuroverkot voidaan katsoa kuuluvan. Artikkelissa tutkijat kertovat kootusti, mitä yli 100:ssa tutkimuksessa on tehty sekä niiden tulokset. Tutkijoiden mukaan heidän artikkelinsa hyöty on johdonmukainen esitys sekä luokittelu älykkäille ja oppiville järjestelmille. Lisäksi

heidän artikkelinsa tarjoaa lähtökohdan vertailla tutkimuksien menetelmiä, jos niitä sovelletaan samoille osakemarkkinoille kuin mitä tutkimuksissa on sovellettu. Tällöin on mahdollista tuottaa arvokkaita tuloksia, jotka voivat tarjota lisätietoja markkinoiden käyttäytymisestä sekä menetelmien suoritukseen vaikuttavista tekijöistä että niiden aineiston herkkyydestä.

Tutkijoiden mukaan tutkimuksissa käytetyillä menetelmillä on ennustettu yhden tai useamman osakemarkkinaindeksin tuottoja. Menetelmät ovat käyttäneet keskimääräisesti neljää tai kymmentä syötemuuttujaa, joista suosituimpia ovat olleet indeksin päivittäiset avaus- tai päätöskurssit. He tulkitsevat, että tämä tukee käsitystä, että älykkäät ja oppivat järjestelmät kuten neuroverkot käyttävät ennustamisessa yksinkertaista syöteaineistoa. Tutkijat esittelevät artikkelissaan, mitä tutkimuksissa on tehty jäsentelemällä menetelmien käyttö osakokonaisuuksiin, jotka kattavat aineiston käsittelyn, otoskoon, menetelmän parametrit, kouluttamisalgoritmin ja ennusteiden arvioinnin. Pääsääntöisesti tutkimuksissa aineistoja on käsitelty joko normalisoimalla tai skaalaamalla ne tietylle välille. Aineiston normalisointi on sisältänyt logaritmisen muunnoksen, ja skaalaus on tehty usein nollan ja ykkösen välille. Tutkijoiden tärkeä havainto on, että lähes kaikki tutkimukset, jotka ovat viitanneet aineiston käsittelyyn, ovat katsoneet sen olevan hyödyllistä ja välttämätöntä. Tutkimuksissa otoskoot ovat vaihdelleet 40 ja 6 000 päivittäisen havainnon välillä. Tutkijat mainitsevat, että suuri määrä aineistoa on edellytyksenä, kun yritetään ennustaa vakiintunutta osakeindeksiä, koska indeksi muuttuu pitkällä aikavälillä. Tutkimuksien menetelmistä 60 % on ollut eteenpäin syöttäviä tai takaisinkytkettyjä neuroverkkoja. Verkoissa on käytetty keskimäärin yhtä tai kahta piilokerrosta huolimatta siitä, että joissain tutkimuksissa on käytetty useita kymmeniä syötemuuttujia. Menetelmien ennustamiskyvyn mittareita ovat olleet keskineliövirheen neliöjuuri (engl. Root Mean Square Error, lyh. RMSE), absoluuttinen keskivirhe (engl. Mean Absolute Error, lyh. MAE) ja ennusteen keskineliövirhe (engl. Mean Squared Prediction Error, lyh. MSPE). Lisäksi osassa tutkimuksissa on käytetty tilastollisia indikaattoreita kuten autokorrelaatiota, korrelaatiokerrointa, keskimääräistä absoluuttista poikkeamaa, neliökorrelaatiota sekä keskihajontaa. (Atsalakis & Valavanis 2009.)

Artikkelin johtopäätöksessä Atsalakis ja Valavanis (2009) ovat todenneet, että tutkimukset osoittavat älykkäiden ja oppivien järjestelmien kuten neuroverkkojen soveltuvan osakemarkkinoiden ennustamiseen. He ovat perustelleet johtopäätöstään tutkimusten tehdyillä kokeilla, joissa älykkäät ja oppivat järjestelmät ovat olleet tarkempia ennustamisessa kuin tavanomaiset matemaattis-tilastolliset menetelmät kuten ARIMA-malli.

Tutkijoiden mukaan ongelmia kuitenkin esiintyy neuroverkoilla, koska verkkojen rakenne on määriteltävä esimerkiksi piiloyksiköiden lukumäärän osalta.

Niaki ja Hoseinzade (2013) ovat artikkelissaan ennustaneet S&P 500 -indeksin päivittäisiä suuntia. Artikkelissa he ovat käyttäneet MLP-verkkoa ja verranneet sen ennustamiskykyä logistiseen malliin. Tutkijat ovat käyttäneet MLP-verkon syötemuuttujina taloudellisia mittareita, jotka ovat parhaiten selittäneet indeksin päivittäisiä suuntia. Syötemuuttujat he ovat valinneet 27 erilaisesta taloudellisesta mittarista. Tutkijoiden mukaan S&P 500 -indeksiä selittävät parhaiten dollarin vaihtokurssit kolmen muun valuutan suhteen. Taloudellisten mittareiden aineisto on kattanut vuosien 1994 ja 2008 välisen ajan sisältäen 3 650 päivittäistä havaintoa. MLP-verkko on koulutettu käyttämällä 80 %:a havainnoista, jonka toimivuutta arvioitu 10 %:lla havainnoista ja suorituskykyä on arvioitu 10 %:lla. Tutkimuksen MLP-verkko on rakennettu kahdella kerroksella. Piilokerros on sisältänyt 60 piiloyksikköä ja ulostulokerros yhden ulostulon. Artikkelin tulokset ovat osoittaneet MLP-verkon kykenevän tarkempiin ennusteisiin kuin logistinen malli.

Malliaris ja Malliaris (2013) ovat tehneet vastaavanlaisen tutkimuksen kuin Niaki ja Hoseinzade. Artikkelissaan he ovat tutkineet MLP-verkon ennustamiskykyä kuuden vuosikymmenen aikana. Tutkijat ovat kouluttaneet MLP-verkkoa 14 syötemuuttujalla, jotka he ovat johtaneet S&P 500 -indeksin päivittäisten päätöskurssien hinnoista. Verkon kouluttamisvaiheen aineisto on sisältänyt kuusi peräkkäistä 10 vuoden ajanjaksoa. Se on alkanut vuodesta 1950 ja päättynyt vuoteen 2010. Tutkijoiden menetelmä on jakaantunut kouluttamis- ja ennustamisvaiheisiin, joita he ovat toistaneet 6 kertaa. Kouluttamisvaiheessa tutkijat ovat kouluttaneet MLP-verkkoa 10 vuoden ajanjaksolla, jonka päätteeksi he ovat ennustaneet S&P 500 -indeksin päivittäiset suunnat seuraavalle vuodelle. Ennustetut suunnat ovat kertoneet tuleeko indeksi nousemaan vai laskemaan edelliseen päivään verrattuna. Tutkijat ovat rakentaneet verkon kahdesta kerroksesta, jossa piilokerros on sisältänyt 9 piiloyksikköä ja tuloskerros yhden ulostulon. Tutkimuksessa ei ole mainittu verkon kouluttamisalgoritmista mitään. Tutkimuksen tulokset ovat osoittaneet että, MLP-verkko kykeni ennustamaan indeksin päivittäisiä suuntia usean vuosikymmenen aikana. Verkon ennustamistarkkuus tippui alle 50 %:n ainoastaan laskusuuntien osalta viimeisenä ennustamisvuotena.

Yhä lisääntyneessä määrin on rahoitustutkimuksissa alettu käyttämään usean älykkään ja oppivan järjestelmän erilaisia yhdistelmiä osakehintojen ennustamisessa. Ne voivat olla joko useamman neuroverkon yhdistelmiä tai vastaava kuten Baon ym. (2017) artikkelin neuroverkko. Näitä menetelmien yhdistelmiä kutsutaan hybridineuroverkoiksi.

Bao ym. (2017) ovat artikkelissaan ennustaneet kuuden eri osakemarkkinaindeksin päivittäisiä hintoja käyttämällä hybridineuroverkkoa, joka on muodostunut LTSM-verkon lisäksi aallokemuunnoksesta (engl. wavelet transform, lyh. WT) sekä pinotuista autoenkoodereista (engl. stacked autoencoders, lyh. SAEs). Tutkijoiden mukaan WT on suodattanut aikasarjaa poistamalla siitä kohinaa, jonka jälkeen SAEs on oppinut aikasarjan taustalla olevat piilevät ominaisuudet, ja LSTM-verkko on ennustanut yhdenpäivän ennusteet. SAEs on neuroverkko, joka muodostuu useasta yksikerroksisesta neuroverkosta. Se on ollut tutkimuksen pääasiallinen neuroverkko. Artikkelissa tutkijat ovat verranneet heidän hybridineuroverkkonsa ennustamiskykyä RNN- ja LSTM-verkkoihin, joiden neuronien tai kerroksien lukumäärää he eivät ole maininneet. Tutkijat ovat käyttäneet hybridineuroverkon syötemuuttujina 15 eri taloudellista mittaria, jotka ovat riippuneet esimerkiksi ennustettavasta indeksistä, useista teknisistä mittareista sekä makrotaloudellisista muuttujista. Eräs kuudesta ennustettavasta osakeindeksistä on S&P 500 -indeksi. Tutkijat käyttävät ennustamisessa indeksejä, jotka ovat otettu maantieteellisesti eri osakemarkkinoilta, koska he haluavat havainnollistaa heidän hybridineuroverkkonsa kyvykkyyttä mukautua eri osakemarkkinoille. Tutkimuksen tulokset ovat osoittaneet, että ehdotettu hybridineuroverkko on tarkempi ennustamisessa kuin RNN- ja LSTM-verkot. Ennustamistarkkuutta mitattiin käyttämällä suhteellista keskivirhettä sekä MAPE:a (engl. mean absolute percentage error).

Makridakis ym. (2018) ovat artikkelissaan vertailleet erilaisia menetelmiä useiden aikasarjojen ennustamisessa. Menetelminä tutkijat ovat käyttäneet kahdeksaa perinteistä tilastollista mallia kuten ARIMA sekä 10:tä koneoppimistekniikkaa sisältäen MLP, RNN ja LSTM. Tutkijoiden pääasiallinen tavoite on ollut tilastollisten mallien sekä koneoppimistekniikoiden ennustamiskyvyn vertaaminen tehden ennusteita eri pituisille ajanjaksoille. Heidän aineistonsa on muodostunut 1 045 aikasarjasta, jotka ovat liittyneet yritysten liiketoimintaan. Kyseisiä aikasarjoja on käytetty M3-kilpailun aineistona. Makridakis-kilpailuista (engl. M-Competitions) on tullut tärkeä vertailukohta testattaessa ja vertailtaessa erilaisten menetelmien ennustamiskykyä (Ahmed ym. 2010). Tutkimuksessa aikasarjojen havainnot ovat olleet kuukausittaisia, ja niiden pituudet ovat vaihdelleet 81 kuukaudesta 126 kuukauteen. Menetelmien mallintamisessa jokaisesta aikasarjasta on vähennetty aluksi 18 kuukautta, jonka jälkeen niitä on käytetty menetelmien ennustamisessa.

Makridakis ym. (2018) ovat käyttäneet MLP-verkossa yhtä piilo- sekä tuloskerrosta. Syötekerroksen syötemuuttujien eli syötesolmujen lukumäärän he ovat määritelleet k-

kertaisella arvioinnilla (engl.10-fold validation). MLP-verkon syötemuuttujien lukumäärä $N = [1, 2, \dots, 5]$, joiden syötteet ovat olleet Y_{t-5}, Y_{t-4}, Y_{t-3}, Y_{t-2} ja Y_{t-1}, kun he ovat ennustaneet aikasarjojen Y_t arvoja. Tämän he ovat tehneet kaikille aikasarjoille, joista on vähennetty 18 kuukautta. Piiloyksiköiden lukumäärän he ovat määrittäneet kaavalla $2N + 1$. Verkon kouluttamisessa he ovat käyttäneet SCG-menetelmää (engl. Scaled Conjugate Gradien -method), jonka oppimisaste on ollut 0,1 ja 1 välillä. Oppimiskierrokset ovat olleet 500. Piiloyksiköiden aktivointifunktio on ollut sigmoid ja ulostulo lineaarinen. Ennen kuin tutkijat ovat kouluttaneet MLP-verkon, he ovat testanneet aineiston muokkaamisen vaikutuksia ennustamistuloksiin useilla vaihtoehtoisilla tavoilla. Tutkimuksessa he ovat päätyneet muokkaamaan aineistoa Box-Cox-muunnoksella sekä poistamalla aikasarjoista niiden trendit että kausivaihtelut. Tätä he ovat soveltaneet kaikissa koneoppimistekniikoissa. Lisäksi he ovat skaalanneet aikasarjat välille 0 ja 1 MLP-verkon tapauksessa, johtuen sen epälineaarisista aktivointifunktioista sekä verkon nopeammasta oppimisesta. Tutkijat ovat rakentaneet RNN- ja LSTM-verkot yhdellä piilosekä tulokerroksella. He eivät ole käyttäneet k-kertaista arviointia syötemuuttujien määrittelyssä, vaan he ovat päätyneet käyttämään 3 syötemuuttujaa, 6 piiloyksikköä ja 1 ulostuloa. Verkkojen kouluttaminen on suoritettu 500 kierroksella, kun oppimisaste on ollut 0.001 ja optimointialgoritmi on ollut RMSProp (engl. Root Mean Square Propagation). Verkkojen neuronit ovat käyttäneet lineaarisia aktivointifunktioita.

Tutkimuksessa menetelmien ennustamiskykyä on arvioitu esimerkiksi käyttämällä SMAPE:a (engl. Symmetric Mean Absolute Percentage Error). SMAPE mahdollistaa menetelmien vertailun, koska se yhdistää eri aikasarjoilla saadut tulokset yhdeksi luvuksi (Ahmed ym. 2010). Tutkijat ovat ennustaneet koneoppimistekniikoilla 10 kertaa, jotta he ovat saaneet laskettua ennustamisvirheiden keskiarvon. Tutkimuksen tulokset ovat osoittaneet, että neuroverkot eivät kykene yhtä tarkkoihin ennusteisiin kuin tilastolliset mallit kuten ARIMA. Tutkijat ovat nostaneet esille, että tulokset voivat johtua aikasarjoista, joita he ovat käyttäneet. Makridakis ym. (2018) ovat korostaneet artikkelissaan, että tarvitaan objektiivisia ja puolueettomia tapoja testata ennustamismenetelmien suorituskykyä. Tämä voidaan saavuttaa mittavilla ja avoimilla kilpailuilla, jotka mahdollistavat tarkoituksenmukaiset vertailut sekä niiden lopulliset päätelmät. Taulukossa 1 on koottu kirjallisuuskatsauksessa käytettyjä artikkeleja, ja niiden keskeisemmät tulokset.

Taulukko 1 Yhteenveto kirjallisuuskatsauksen artikkeleista.

Kirjoittajat	Artikkeli	Vuosi	Menetelmä(t)	Aineisto	Artikkelin johtopäätös
Atsalakis ja Valanis	Surveying stock market forecasting techniques – Part II: Soft computing methods.	2009	Artikkelikatsaus	Yli 100 tutkimusta, joissa käytetään älykkäitä ja oppivia järjestelmiä osakemarkkinoiden ennustamisessa	Älykkäät ja oppivat järjestelmät ovat soveltuvia osakemarkkinoiden ennustamiseen
Niaki ja Hoseinzade	Forecasting S&P 500 index using artificial neural networks and design of experiments	2013	Päämenetelmä: MLP	Syötemuuttujat: Parhaiten indeksin päivittäisiä suuntia selittävät taloudelliset mittarit	1) Dollarin vaihtokurssi kolmen muun valuutan suhteen selittävät parhaiten S&P 500 -indeksin päivittäisiä suuntia
			Vertailumenetelmä: Logistinen malli	Ulostulo: S&P 500 -indeksi	2) Tilastollisen analyysin perusteella MLP on tarkempi ennustamisessa kuin logistinen malli
Malliaris ja Malliaris	Neural network forecasting with the S&P 500 index across decades.	2013	MLP	Syötemuuttujat: S&P 500 -indeksin päätöskursseista johdetut useat mittarit	MLP-verkko kykenee ennustamaan S&P 500-indeksiä useiden vuosikymmenien ajan
				Ulostulo: S&P 500 -indeksi	
Bao ym.	A deep learning framework for financial time series using stacked autoencoders and long-short term memory.	2017	Päämenetelmä: Hybridineuroverkko	Syötemuuttujat: Yhtä osakemarkkinaindeksiä kohden 15 taloudellista mittaria	Hybrineuroverkko kykenee ennustamaan tarkemmin indeksien päivittäisiä päätöskursseja kuin vertailumenetelmät
			Vertailumenetelmät: LSTM ja RNN	Ulostulo: Jokin 6:sta osakemarkkinaindeksistä	
Makridakis ym.	Statistical and Machine Learning forecasting methods: Concerns and ways forward.	2018	Tilastolliset menetelmät: 8 menetelmää mm. ARIMA	1045 kuukausittaista aikasarjaa	Perinteiset tilastolliset menetelmät ovat tarkempia ennustamisessa kuin koneoppimismenetelmät
			Koneoppimismenetelmät: 10 menenetelmää mm. MLP, RNN, LSTM ja random forest		

4 AINEISTO JA MENETELMÄT

4.1 Aineiston esittely

Tutkielman aineistona käytettävä taloudellinen aikasarja otetaan Standard & Poor 500 indeksistä. S&P 500 -indeksi on Yhdysvaltojen seuratuin osakemarkkinoita kuvaava osakeindeksi. Indeksi on muodostettu markkina-arvoltaan 500 suurimman yhtiön osakkeista, jotka ovat listattu NYSE- ja NASDAQ-pörsseissä (Hull 2012, 61). Osakkeiden markkina-arvot määrittävät sen, kuinka osakkeiden arvoja painotetaan indeksin muodostamisessa. Indeksi ladataan Thomson Reuters Eikon -tietokannasta ja ajanjaksoksi valitaan 23.11.2011–31.10.2019, joka sisältää 2 250 päivittäistä havaintoa indeksin kaupankäyntipäiviltä. Havainnot ovat indeksin päivittäisiä päätöskursseja, joiden valuuttana ovat eurot. Päivittäisiä päätöskursseja käytetään, koska niiden käyttö on suosittua aiheen tutkimuksissa (Atsalakis & Valavanis 2009). Eurojen valintaa perustellaan luvun 5 tuloksien esittelyn yhteydessä. Neuroverkot vaativat aineistolta, että se sisältää riittävästi tietoa verkkojen kouluttamiseksi ja ulostulojen ennustamiseksi (Graves 2012 ,30). Täten tutkielman taloudellinen aikasarja jaetaan lyhyen sekä pitkän aikavälin aikasarjoiksi. Lyhyen aikavälin aikasarja sisältää 750 havaintoa ajalta 8.11.2016–31.10.2019 ja pitkän aikavälin aikasarja kattaa koko ajanjakson 2 250 havainnolla. Eri pituiset aikasarjat takaavat, että neuroverkoilla on riittävästi tietoa sekä ne mahdollistavat tarkastelun, kuinka RNN- ja LSTM-verkkojen muistiominaisuudet vaikuttavat verkkojen ennustamiskykyyn.

Neuroverkot edellyttävät aikasarjalta, että se on muokattava sopivaan muotoon ennen kuin se syötetään verkoille. Täten verkot oppivat tehokkaasti kuvaamaan aikasarjan prosessia, jonka pohjalta verkko kykenee ennustamaan. Toisaalta ARIMA-mallin käyttö edellyttää aikasarjan olevan stationaarinen. Tutkielman eri aikavälien taloudellisia aikasarjoja muokataan kolmessa vaiheessa. Ensimmäisessä vaiheessa ne skaalataan välille, jolla neuroverkot kykenevät oppimaan ne tehokkaasti. Brownleen (2019) mukaan on kaksi tapaa aikasarjan skaalaukselle. Tavat ovat standardisointi ja normalisointi. Standardisointi soveltuu tapaukseen, jossa aikasarjan havainnot ovat normaalisti jakautuneita. Normalisointia käytetään, kun havainnot eivät ole normaalisti jakautuneita. Normalisointi mahdollistaa neuroverkkojen paremman oppimisen tapauksessa, jossa havaintojen arvot ovat suuria. Standardisoinnin ja normalisoinnin lisäksi aikasarja voidaan pyrkiä saattamaan stationaariseksi differentioimalla sarja, jolloin sen trendi ja kausivaihtelut saattavat poistua. Toisaalta se voidaan muokata logaritmiseksi tuottosarjaksi. Näillä kahdella

edellä mainitulla tavalla on kuitenkin kääntöpuolensa. Lopez de Pradon (2019, 75–66) mukaan nämä tavat kadottavat alkuperäisen aikasarjan informaatiota.

Tutkielman neuroverkot koulutetaan sekä lyhyen että pitkän aikavälin aikasarjoilla. Molempia aikasarjoja muokataan normalisoimalle ne välille 0 ja 1 ennen verkkojen kouluttamista. Lisäksi aikasarjat differentioidaan kertaalleen, jonka jälkeen ne normalisoidaan välille 0 ja 1. Täten neuroverkkojen kouluttaminen tehdään neljälle eri aikasarjalle ja näiden kouluttamiskertojen pohjalta verkoilla ennustetaan. Taulukon 2 tulokset puoltavat aikasarjojen muokkausta, koska siinä esitellyt tilastolliset tunnusluvut osoittavat, että aikasarjojen havainnot eivät ole normaalijakautuneita sekä yksittäisten havaintojen arvot ovat suuria. Esimerkiksi 750 havainnon aikasarjasta laskettu jakautuman vinous on 0,49 ja huipukkuus on -0,92, jolloin jakautuma on oikealla vino ja sen huippu on litteä. Normaalijakautuman tapauksessa vastaavat tunnusluvut ovat nollia. Lisäksi 750 havainnon aikasarjan suurin havainto on 2 739 ja pienin 1 941.

Taulukko 2 Taloudellisen aikasarjan tilastolliset tunnusluvut.

Tunnusluvut

S&P 500 aikasarja[a)]	Keskiarvo	Keskihajonta	Vinous	Huipukkuus	Minimi	Maksimi
750	2 327,9	192,1	0,49	-0,92	1 940,9	2 738,7
2 250	1 696,2	551,1	0,05	-1,29	780,5	2 738,7

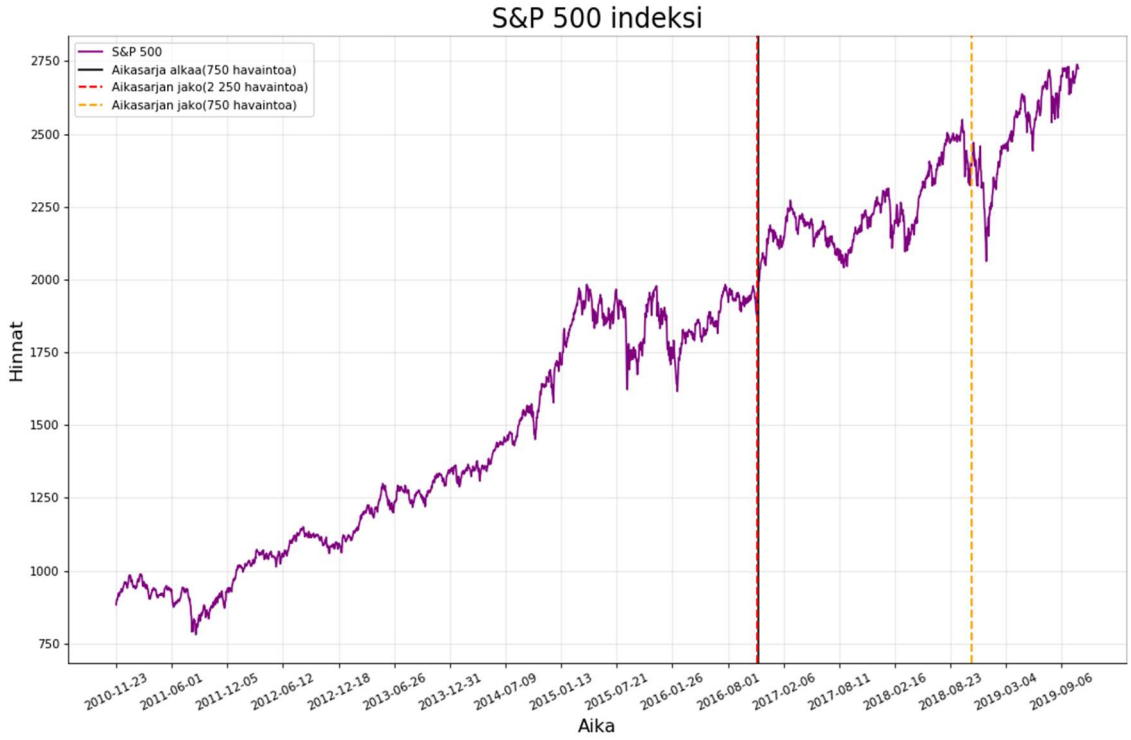
ADF-testi

S&P 500 aikasarja[a)]	ADF-arvo	p-arvo	Kriittiset arvot *=1%, **=5%, ***=10%
750	-1,21	0,67	-3,44* -2,87** -2,57***
log750	-1,39	0,59	-3,44* -2,87** -2,57***
d750	-29,63	<0,001	-3,44* -2,87** -2,57***
2 250	0,08	0,97	-3,43* -2,86** -2,57***
log2250	-0,71	0,84	-3,43* -2,86** -2,57***
d2250	-14,01	<0,001	-3,43* -2,86** -2,57***

[a)] X = aikasarjan havaintojen lukumäärä, logX = logaritmoitu aikasarja, dX = kertaalleen differentioitu aikasarja.

Taulukossa 2 on laajennetun Dickey-Fuller -testin (lyh. ADF) tulokset tutkielman aikasarjoille sekä niiden logaritmoiduille että kertaalleen differentioiduille versioille. ADF-testin tulokset osoittavat, että aikasarjat 750, log750, 2 250 ja log2250 sisältävät yksikköjuuren, mikä merkitsee niiden olevan epästationaarisia. Taulukossa 2 epästationaarisuus on nähtävissä, koska p>0,05. Tämä tarkoittaa, että H_0 hypoteesia ei voida hylätä ja aikasarjat sisältävät yksikköjuuren. Kertaalleen differentioidut aikasarjat näyttävät olevan stationaarisia, koska p<0,05, jonka perusteella H_0 hypoteesi hylätään. Kertaalleen differentioidut aikasarjat kadottavat informaatiota, koska differentiointi poistaa aikasarjoista niiden ensimmäisen havainnon. ADF-testin tulokset puoltavat, että aikasarjat differentioidaan. ADF-testin tulokset esitellään, koska ne auttavat vastaamaan tutkielman toiseen tutkimuskysymykseen: Soveltuvatko neuroverkot taloudellisen aikasarjan mallintamiseen ja ennustamiseen?

Toisessa vaiheessa aikasarjat jaetaan kahteen osaan. Jaon ensimmäisestä osasta käytetään käsitettä koulutusaineisto (engl. training data), jolla neuroverkot koulutetaan. Jaon toista osaa käytetään neuroverkkojen ennustamisessa, mistä käytetään käsitettä testausaineisto (engl. testing data). Lisäksi eräissä neuroverkkojen tutkimuksissa on näiden välillä osa, josta käytetään käsitettä arviointiaineisto (engl. validation data). Tämä osa rajataan tutkielman ulkopuolelle. Tutkielman aikasarjat jaetaan suhteessa 66,667 % koulutusaineistoon ja 33,333 % testausaineistoon. Tämä jakosuhde on lähellä suhdetta, jota käytetään yleisesti aiheen tutkimuksissa. Tutkimuksissa on yleisesti käytetty suhdetta 70 % koulutusaineistoon ja 30 % testausaineistoon, mikä soveltuu pienemmän koon aineistoille. Tutkielman jakosuhteeseen on päädytty, koska se tuotti menetelmien ennustamisessa parempia tuloksia kuin tutkimuksissa yleisesti käytetty jakosuhde. Lyhyempien aikasarjojen koulutusaineistot sisältävät 500 päivittäistä havaintoa ja testausaineistot 250. Pidempien aikasarjojen koulutusaineistot sisältävät 1500 päivittäistä havaintoa ja testausaineistot 750. Differentiointi poistaa aikasarjojen koulutusaineistoista niiden ensimmäiset havainnot, mikä koskee d750- ja d2250-aikasarjoja. Kuviossa 9 esitetään S&P 500 -indeksin aikasarja sekä sen jakosuhteet.



Kuvio 9 S&P 500 -indeksi.

Kuvion 9 y-akselilla ovat havaintojen arvot euroina ja x-akselilla ajanjakso, johon arvot sijoittuvat. Indeksin aikasarja näyttää käyttäytyvän nousevan trendin mukaisesti koko sen tarkastelujaksolla. Kuvion 9 punainen katkoviiva on pidemmän aikasarjan jakosuhde. Koulutusaineistot koostuvat punaisen katkoviivan vasemmanpuoleisesta osasta ja testiaineistot katkoviivan oikeanpuoleisesta osasta. Musta viiva kuvaa ajankohtaa, josta 750 havainnon aikasarja alkaa ja keltainen katkoviiva on sarjan jakosuhde. Testiaineistojen pohjalta neuroverkoilla ennustetaan indeksin päivittäisiä arvoja aikaväleille, jotka vastaavat punaisen sekä keltaisen katkoviivan oikeanpuoleisia osia.

Kolmannessa vaiheessa aikasarjojen havainnot syötetään neuroverkoille käyttäen liukuva ikkuna menetelmää (engl. sliding window method). Se tunnetaan tilastotieteessä viivemenetelmänä. Neuroverkkojen kouluttamisvaiheessa tähän menetelmään vaikuttavat verkkojen hyperparametrit, joilla säädetään sitä tapaa, kuinka koulutusaineisto syötetään verkoille. Näitä hyperparametreja tarkastellaan tarkemmin seuraavassa luvussa. Tiivistetysti yhden verkon tapauksessa menetelmällä syötetään verkolle havaintojen viivästettyjä arvoja, joista verkko laskee ulostuloarvon. Tämän jälkeen verkko vertaa ulostuloarvoa koulutusaineiston havainnoista saatuun tavoitearvoon, ja laskee näiden välisen virheen, jonka perusteella verkon parametreja säädetään. Neuroverkon ennustamisvaiheessa testiaineiston havainnot syötetään verkolle käyttäen samaa havaintojen viivettä kuin millä verkkoa on koulutettu, mutta tässä vaiheessa verkon parametrit ovat vakioita.

4.2 Tutkielman menetelmien esittely

4.2.1 Tutkielman neuroverkot ja niiden hyperparametrit

Tutkielmassa käytetään MLP-, RNN- ja LSTM-verkkoja. Pääpiirteissään näiden verkkojen kouluttaminen eli ohjattu oppiminen on verrattavissa perinteiseen tilastolliseen mallintamiseen, jossa mallin parametrit sovitetaan koulutusaineistoon sopiviksi. Neuroverkoilla on kahdenlaisia parametreja. Näitä ovat hyperparametrit sekä varsinaiset parametrit eli solmujen väliset painot sekä niiden vakiotermit. Hyperparametrit vaikuttavat neuroverkon rakenteen kautta varsinaisiin parametreihin, jotka taas vaikuttavat suoraan ulostuloarvojen muodostamiseen. Hyperparametrit ohjaavat liukuvan ikkunan menetelmää sekä verkon rakennetta että kouluttamista.

Hyperparametrit liukuvan ikkunan menetelmässä, joka vastaa syötekerrosta ovat ikkunan leveys (engl. window size) eli havaintojen viiveiden määrä, eräkoko (engl. batch size) ja sekoituspuskurin koko (engl. suffle buffer size). Eräkoko on niputettu joukko ikkunoita, jotka syötetään neuroverkolle ennen kuin varsinaisten parametrien optimointi aloitetaan, jonka jälkeen verkolle syötetään uusi joukko ikkunoita, kunnes kaikki ikkunat on syötetty verkolle. Sekoitus-puskurin koko tarkoittaa ikkunoiden järjestyksen sekoittamista keskenään ennen kuin ne syötetään neuroverkolle. Esimerkiksi tapauksessa, jossa sekoituspuskurin koko on 5 ja koulutusaineistosta muodostettuja ikkunoita on 10 kappaletta. Tällöin 10 ikkunasta otetaan satunnaisesti 5 kappaletta, joiden järjestys vaihdetaan satunnaisesti, jonka jälkeen ikkunat syötetään verkolle. Erä ja sekoituspuskurin kokoa käytetään ainoastaan neuroverkkojen kouluttamisvaiheessa, jotta saataisiin aikaiseksi verkon parempi oppiminen. Näiden hyperparametrien toimivuuteen vaikuttavat koulutusaineiston koon lisäksi esimerkiksi verkon kouluttamisessa käytetty virhefunktion valinta.

Hyperparametrit, jotka koskevat verkon rakennetta ovat piilokerrosten, syötemuuttujien, piiloyksiköiden ja ulostulojen lukumäärät sekä kahden jälkimmäisen aktivointifunktiot. Hyperparametrit verkon kouluttamisessa ovat gradienttimenetelmä, oppimisnopeus (engl. learning rate), momentti, virhefunktio ja kierrokset (engl. epoch). Momentti on oppimisnopeuteen liittyvä hyperparametri ja kierrokset ovat verkon oppimiskierroksia.

Neuroverkkoihin kohdistuneissa tutkimuksissa ei ole yhtenäistä näkemystä, kuinka valita oikeat hyperparametrit (Atsalakis & Valavanis 2009). Täten suurin osa tutkielman hyperparametreista valitaan ristihaulla (engl. grid search) sekä kokeile ja erehdy -taktiikalla. Näiden lähtökohtana käytetään Moroneyn (2019) koodien hyperparametri valintoja. Ristihaun taustalla on algoritmi, joka avustaa oikeiden hyperparametrien

etsimisessä. Algoritmi kokeilee useita erilaisia hyperparametri kombinaatioita, joilla se kouluttaa valittua neuroverkkoa sekä ennustaa sillä. Algoritmin toiminta päättyy sen tulostaessa kombinaatiot, joilla verkko on mahdollisesti saavuttanut pienimmät ennustamisvirheet. Tuloksien esittelyn yhteydessä perustellaan tarkemmin hyperparametrien valintaprosessia sekä niiden vaikutuksia tutkielman tuloksiin. Taulukossa 3 on esitetty kootusti tutkielman neuroverkkojen parhaaksi todetut hyperparametrit neljälle aikasarjalle.

Taulukko 3 Käytetyt neuroverkkojen hyperparametrit.

MLP(m), RNN(r) ja LSTM(l)							
S&P 500 aikasarja [a)]	Syötekerros			Piilokerros		Tuloskerros	
	Ikkunan leveys	Erä koko	Sekoituspuskurin koko	Neuronit	Aktivointifunktio	Neuronit	Aktivointifunktio
750	1	6	1	250	relu (m) tanh (r, l)	1	linear
d750	1	6	250	250	relu (m) tanh (r, l)	1	linear
2 250	1	6	1	250	relu (m) tanh (r, l)	1	linear
d2250	1	6	1 000	250	relu (m) tanh (r, l)	1	linear

S&P 500 aikasarja [a)]	Kouluttamisen hyperparametrit				
	Oppimisnopeus[1]	Momentti	Kierrokset	Virhefunktio	Gradienttimenetelmä
750	0,1	0,9	250	MSE	SGD
d750	0,01	0,5	250	MSE	SGD
2 250	0,1	0,9	250	MSE	SGD
d2250	0,01	0,5	250	MSE	SGD

[a)] X = aikasarjan havaintojen lukumäärä, dX = kertaalleen differentioitu aikasarja.
[1)] Oppimisnopeus-sarakkeen rivien arvot on skaalattu kertomalla rivit 10^{-2}.

Taulukon 3 sekoituspuskurin koko, oppimisnopeus ja momentti vaihtelevat ainoastaan aikasarjojen välillä, jotta verkkojen vertailu olisi mahdollisimman läpinäkyvää. RNN- ja LSTM-verkot eroavat MLP-verkosta piilokerroksen aktivointifunktion osalta. RNN- ja LSTM-verkoissa piiloyksiköiden aktivointifunktiot ovat hyperbolisia tangentteja, kun taas MLP-verkon aktivointifunktio on ReLu. Kaikki verkot sisältävät yhden piilokerroksen 250 neuronilla. Verkkojen tuloskerros sisältää ainoastaan yhden neuronin lineaarisella aktivointifunktiolla. Yksi ulostulo on riittävä, koska tutkielmassa ennustetaan yhden päivän hintoja. Lineaarinen aktivointifunktio takaa, että ulostuloarvot ovat vertailukelpoisia tavoitearvoihin. Gradienttimenetelmä on stokastinen gradient descent (SGD). Kouluttamisvaiheen virhefunktio on MSE ja kierrosten lukumäärä on 250.

4.2.2 ARIMA(p, d, q)-mallit

Useat aikasarjat voidaan katsoa prosesseina, joissa prosessin käyttäytyminen riippuu aikasarjan menneistä havainnoista eli viivearvoista (Kahra & Kanto 1999, 8). Eräs laajalti käytetty tilastollinen menetelmä on ARMA(p, q)-malli, jolla on kyetty kaappaamaan stationaaristen aikasarjojen käyttäytyminen tehden mallista suositun ennustamismenetelmän. Mallissa yhdistyvät autoregressiivinen AR(p)- ja liukuvan keskiarvon MA(q)-prosessit. Prosesseissa p ja q ovat asteita, jotka määrittävät mallin viivearvojen lukumäärät. ARMA(p, q)-mallin yhtälö on

$$y_t = \mu_0 + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t, \qquad \varepsilon_t \sim \text{iid}(0, \sigma^2), \tag{4.1}$$

missä y_t on aikasarjan havainnon arvo ajanhetkellä t ja μ_0 on vakiotermi. Havainnon y_t arvon oletetaan riippuvan lineaarisesti viivästettyjen havaintojen arvoista. Kun aikasarjasta on vähennetty sen keskiarvo niin $\mu_0 = 0$. AR(p)-prosessin parametri on ϕ_i, jossa alaindeksi i viittaa monesko p aste on kyseessä. MA(q)-prosessin parametri on θ_i, jossa i alaindeksi viittaa monesko q aste on kyseessä. Yhtälössä 4.1 ε_t on virhetermin satunnaismuuttuja, joka on identtisesti ja riippumattomasti jakautunut (engl. identically and independently distributed, lyh. iid). Identtisesti jakautunut viittaa siihen, että satunnaismuuttujan arvot ovat generoituneet samasta todennäköisyysjakautumassa ja normaalijakautuman tapauksessa käytetään lyhennettä nid (Kahra & Kanto 1999, 4). Käyttämällä viivästysoperaattoria B sekä polynomeja $\phi(B) = 1 - \phi_1 B - \cdots - \phi_p B^p$ ja $\theta(B) = 1 + \theta_1 B + \cdots + \theta_q B^q$ voidaan ARMA-malli kirjoittaa muodossa

$$\phi(B) y_t = \theta(B) \varepsilon_t. \tag{4.2}$$

Jotta malli soveltuisi mallintamaan ja ennustamaan stationaarista aikasarjaa on sen vastattava stationaarista prosessia. Täten mallin parametrien on täytettävä sekä stationaarisuus- että invertoituvuusehdot. Stationaarisuuden käsite voidaan jakaa heikkoon ja vahvaan stationaarisuuden muotoon. Aikasarja on heikosti stationaarinen, kun se täyttää seuraavat ehdot:

1. Aikasarjassa ei ole trendiä, jolloin sen keskiarvo on ajasta riippumaton ja äärellinen.
2. Aikasarjan varianssi on ajasta riippumaton ja äärellinen.
3. Aikasarjan havaintojen välinen kovarianssi riippuu vain viiveistä, ei havainnointihetkestä.

Aikasarja täyttää vahvat stationaarisuusehdot, kun sitä kuvaavan prosessin todennäköisyysjakautuma ei muutu ajassa. Esimerkiksi tällöin normaalijakautuman kaikki neljä momenttia, keskiarvo, varianssi, vinous ja huipukkuus ovat vakioita eri ajanhetkillä. Mainittakoon, että invertoituvuusehto tarkoittaa esimerkiksi MA(1)-prosessin kääntämistä AR(∞)-prosessiksi, jota ei tarkemmin tarkastella tutkielmassa. (Kahra & Kanto 1999, 6.)

ARMA-malli ei teoreettisesti sovellu suoraan epästationaaristen aikasarjojen mallintamiseen ja ennustamiseen. Kuitenkin epästationaarisia aikasarjoja voidaan mallintaa menetelmillä, jotka ovat kehitetty stationaarisille aikasarjoille. Tässä aikasarjojen epästationaarisuus poistetaan jollakin muunnoksella. Eräs suosittu ja käytetty muunnos on aikasarjan differentiointi. Täten ARMA(p, q)-malli voidaan laajentaa ARIMA(p, d, q)-malliksi, jossa I(d)-prosessin sanotaan olevan integroitunut d astetta (Franses & Dijk 2000, 25). Aste d kertoo differentiointien lukumäärän, jolla aikasarja saatetaan stationaariseksi.

Box ym. (2016, 204) mukaan ARIMA-mallin kehittäminen sellaiseksi, että se kykenee kuvaamaan aikasarjan riippuvuusrakenteen, saavutetaan parhaiten yleensä kolmivaiheisella iteratiivisella menetelmällä. Tämä Box-Jenkins -menetelmän vaiheet ovat mallin identifiointi, estimointi ja sen diagnostiset tarkistukset. Mallin identifioinnissa tarkastellaan aikasarjan graafista kuvaajaa, joka saattaa selventää aikasarjan rakennemuutoksia esimerkiksi trendien ja kausivaihteluiden osalta. Aikasarjan rakennemuutokset voidaan poistaa tietyissä tapauksissa käyttämällä differentiointia. Mallin estimointivaiheessa vaihtoehtoiset ARIMA-mallit pyritään mallintamaan aikasarjaan sopiviksi eri parametreilla. Näitä estimoituja malleja vertaillaan käyttämällä erilaisia kriteerejä. Kriteerit koskevat säästäväisyyttä, stationaarisuutta, invertoitavuutta ja sovitekriteerejä. Säästäväisyyden ajatuksena on valita sellainen malli, joka ei käytä liikaa parametreja. Stationaarisuutta ja invertoitavuutta arvioidaan esimerkiksi tarkastelemalla ARIMA-mallien autokorrelaatio- ja osittaisautokorrelaatiofunktioiden (lyh. ACF ja PACF) kuvaajia, joita sitten verrataan niiden teoreettisiin vastineisiin. Sovitekriteerit kuten Akaiken informaatiokriteeri (AIC) ja Bayesilainen informaatiokriteeri (BIC) ovat mallin valinnassa käytettyjä mittareita. Mittarit laskevat mallin sovittamiskykyä sakottamalla mittarin tulosta, kun malli sisältää useita parametreja. Kriteerien käyttäminen varmistaa, että mallien estimoidut parametrit sijaitsevat sallitulla alueella, jolloin mallilla kyetään ennustamaan luotettavasti. Mallin diagnostiset tarkistukset tehdään mallin estimoinnin jälkeen esimerkiksi tarkastelemalla mallin residuaalien kuvaajia. Tässä voidaan käyttää residuaalien ACF-, PACF- ja jakautumakuvaajien lisäksi jakautuman tilastollisia tunnuslukuja. Kuvaajista voidaan havaita, jos malli ja sen parametrit eivät kykene kaappaamaan aikasarjan käyttäytymistä.

Tutkielman ARIMA-mallit mallinnetaan sekä 750 että 2 250 havainnon aikasarjoilla. Aikasarjat jaetaan koulutus- ja testausaineistoihin käyttämällä samaa jakosuhdetta kuin neuroverkoilla. Aikasarjoja ei muokata, koska ARIMA-malli sisältää differentioinnin. Tutkielman ARIMA(1,1,0)-mallin parametrien valinta perustellaan luvussa 5. ARIMA-mallin käyttämisellä tavoitellaan riittävää lähtötasoa neuroverkkojen ennustamiskyvyn arvioinnille, koska ARIMA-malli edustaa ennustamismenetelmien suosittua klassista näkökulmaa.

4.3 Menetelmien ohjelmointi

Tutkielman neuroverkot ohjelmoidaan Python-ohjelmointikielellä, jonka jakelijana toimii ilmainen ja avoimen lähdekoodin Anaconda. Integroituna ohjelmointiympäristönä (engl. integrated development environment, lyh. IDE) käytetään PyCharmia. Python tarvitsee neuroverkkojen ohjelmoinnissa erilaisia ohjelmistokirjastoja. Eräs näistä ohjelmistokirjastoista on TensorFlow, joka on suunniteltu koneoppimiseen. Se on ilmainen ja perustuu avoimeen lähdekoodiin. Tästä syystä se on erittäin käytetty neuroverkkojen ohjelmoinnissa. Muut ohjelmistokirjastot, joita Python tarvitsee ovat nähtävissä liitteiden ohjelmointikoodeista. Pythonia käytetään verkkojen ohjelmoinnissa, koska se on noussut viimevuosina eniten käytetyksi neuroverkkojen ohjelmointikieleksi. Anacondaa käytetään Pythonin jakelijana, koska se mahdollistaa helpon TensorFlow-asennuksen. TensorFlow-versiot ovat 2.1 CPU sekä GPU. CPU (engl. Central Processing Unit) tarkoittaa, että TensorFlow käyttää tietokoneen prosessoria neuroverkkojen laskennassa, kun taas GPU (engl. Graphics Processing Unit) tarkoittaa näytönohjaimen CUDA-ytimien laskentatehon hyödyntämistä. Valinta näiden versioiden välillä perustellaan luvussa 5. Kaikki tutkielman ohjelmistokirjastojen versiot ovat nähtävissä liitteistä.

Neuroverkkojen ohjelmointikoodit otetaan kahdesta eri lähteestä. Pääsääntöisesti tutkielmassa käytetään Moroneyn (2019) neuroverkkojen koodeja, koska ne ovat suunniteltu TensorFlow 2.0:lle. Näihin koodeihin tehdään tarvittavia muokkauksia sekä niitä täydennetään Jason Brownleen (2019) koodeilla, jotka ovat ohjelmoitu käyttäen Keras-ohjelmistokirjastoa ja ne vaativat muokkausta toimiakseen TensorFlow 2.1:lla. ARIMA-mallit ohjelmoidaan käyttäen Brownleen (2019) ARIMA-koodeja. Lisäksi edellä mainittujen lähteiden koodeista rakennetaan neuroverkkojen ristihakualgoritmi. Koodit suoritetaan käyttämällä Intelin i5-9600K prosessoria nopeudella 4,6 GHz. RAM-muistina on 32 gigatavua DDR4 3200 MHz. Näytönohjaimena on Nvidia GeForce RTX 2070.

5 EMPIIRINEN TESTAUS JA TULOKSET

5.1 Menetelmien kouluttaminen ja niillä ennustaminen

Neuroverkot koulutettiin edellisessä luvussa esitetyillä hyperparametreilla, joiden etsintä aloitettiin käyttämällä ristihakualgoritmia. Etsintä suoritettiin jokaiselle neuroverkolle jokaista aikasarjaa kohden. Algoritmin ehdottamia parhaita hyperparametrien kombinaatioita tarkasteltiin tarkemmin kokeile ja erehdy -taktiikalla. Taktiikka voidaan jakaa kouluttamis- ja ennustamisvaiheisiin. Kouluttamisvaiheessa verkkojen oppimiskykyä seurataan kouluttamiskäyrän avulla. Kouluttamiskäyrä kertoo esimerkiksi, kuinka hyvin verkot oppivat sekä mahdollisista ali- tai ylisovittamisesta johtuvista ongelmista. Ennustamisvaiheessa verkkojen ennustamistarkkuutta arvioidaan valitulla mittarilla. Yhdessä edellä esitellyt vaiheet antavat suuntaa oikeiden hyperparametrien valinnalle. Neuroverkkojen ja ARIMA-mallien ennustamistarkkuuden mittaamisessa käytettiin keskineliövirheen neliöjuurta (engl. Root Mean Squared Error, lyh. RMSE)

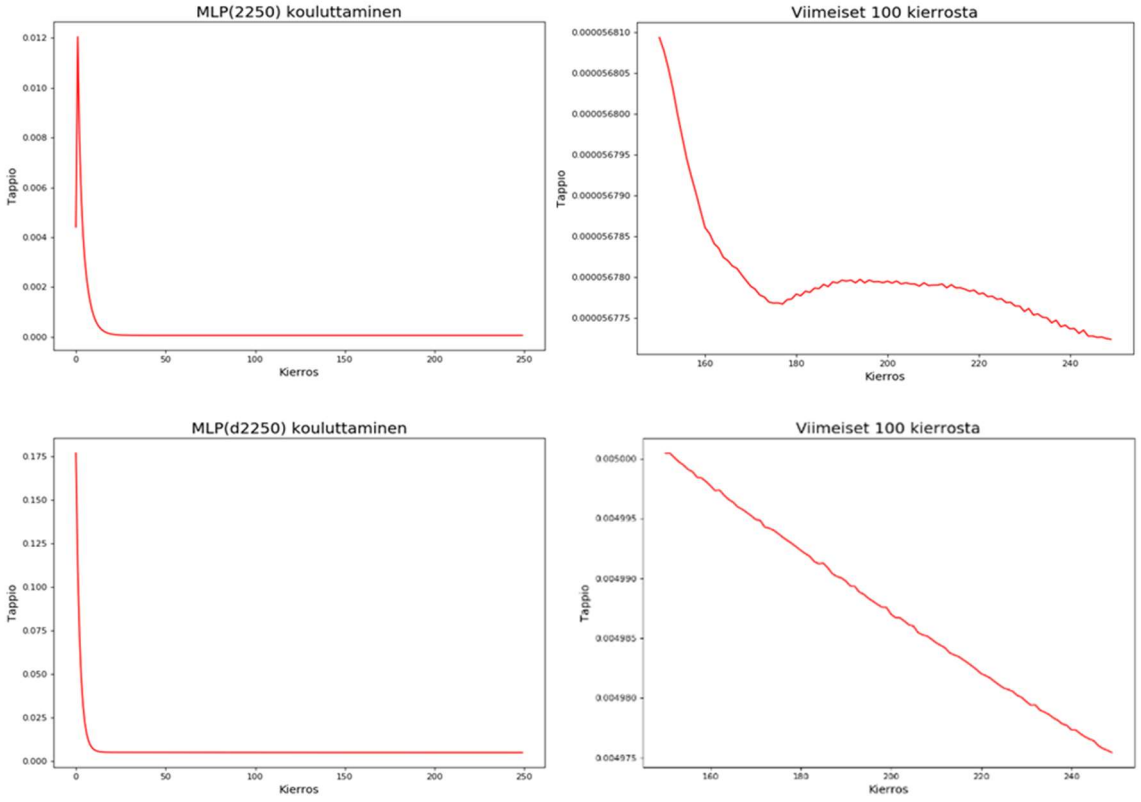
$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}, \tag{5.1}$$

missä arvo y_i on testiaineiston havainto i, arvo $\hat{y}_i$ on verkon ennuste havainnolle i ja N on havaintojen lukumäärä. Ennen virheiden laskemista neuroverkkojen ennusteet skaalattiin niiden alkuperäisille väleille.

Sopivat ARIMA(p, d, q)-mallin parametrit molemmille aikasarjoille valittiin käyttämällä vastaavanlaista ristihakualgoritmia kuin neuroverkoilla. Algoritmi mallintaa useita ARIMA-malleja käyttäen erilaisia parametrikombinaatioita. Jokaisen mallin mallinnuksen jälkeen algoritmi ennustaa kyseisellä mallilla. Algoritmin toiminta päättyy, kun se tulostaa mallin parametrit, joilla on ollut pienin RMSE. RMSE:n perusteella valittiin tarkempaan vertailuun 750 havainnon aikasarjalle ARIMA$(1,1,0)$-, ARIMA$(0,1,1)$- sekä ARIMA(2,1,0)-mallit. Pidemmän aikasarjan tapauksessa valittiin ARIMA$(1,1,0)$-, ARIMA$(0,1,1)$- sekä ARIMA(0,1,2)-mallit. Näiden mallien RMSE-arvot olivat pienimmät useista kymmenistä muista malleista. Valittuja malleja vertailtiin hyödyntämällä osittain Box-Jenkins -menetelmää. Täten päädyttiin käyttämään molemmilla aikasarjoilla ARIMA$(1,1,0)$-mallia antamaan lähtötaso neuroverkkojen ennustamiskyvylle.

5.2 Kouluttamisen tulokset

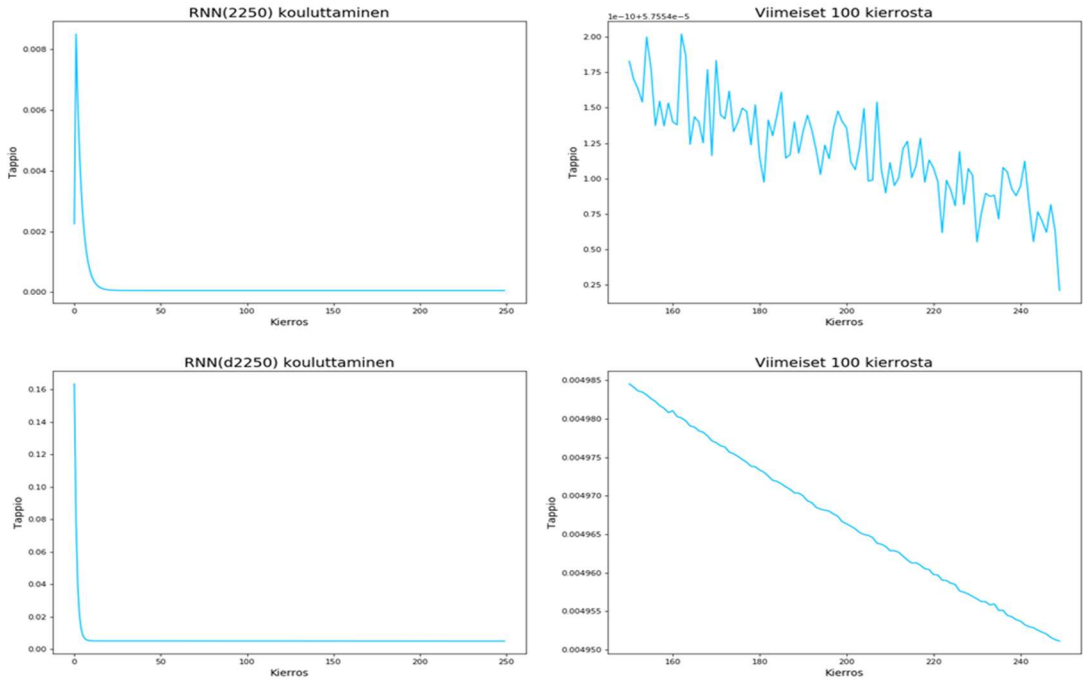
Neuroverkkojen koodit suoritettiin CPU:lla, koska CPU suoriutui nopeammin niiden laskennasta. Tämä nousi esille neuroverkkojen ristihakualgoritmin suorittamisessa. CPU:n ja GPU:n laskennallisia aikoja sekä käytettyjä hyperparametri kombinaatioita on esitetty liitteissä. Neuroverkot ovat luonteeltaan satunnaisia, johtuen painoarvojen satunnaisesta alustamisesta. Tämä tarkoittaa, että verkot aloittavat varsinaisten parametrien optimoinnin jokaisella kouluttamiskerralla hieman eri lähtökohdista. Tämä johtaa tilanteeseen, jossa kouluttamisen tuloksena saadut varsinaiset parametrit eroavat muista kouluttamiskertojen parametreista, mikä johtaa erilaisiin ennustamistuloksiin kouluttamiskertojen väleillä. Lisäksi verkkojen satunnainen oppiminen näkyy kouluttamiskäyrien muodon vaihteluna eri kouluttamiskertojen väleillä. Täten tutkielman neuroverkot koulutettiin 50 kertaa jokaista aikasarjaa kohden, ja jokaisen kouluttamisvaiheen jälkeen verkoilla ennustettiin. Tämä prosessi mahdollistaa luotettavammat tulokset sekä havainnollistaa, miten luotettavia eri verkot ovat. Kuviossa 10 ovat MLP-verkon kouluttamiskäyrät, kun verkkoa on koulutettu 2 250 havainnon aikasarjalla sekä sen differentioidulla versiolla. Ylemmät käyrät vastaavat 50 kouluttamiskerrasta 28. kertaa ja alempi 16. kertaa.



Kuvio 10 MLP-verkon kouluttamiskäyrät.

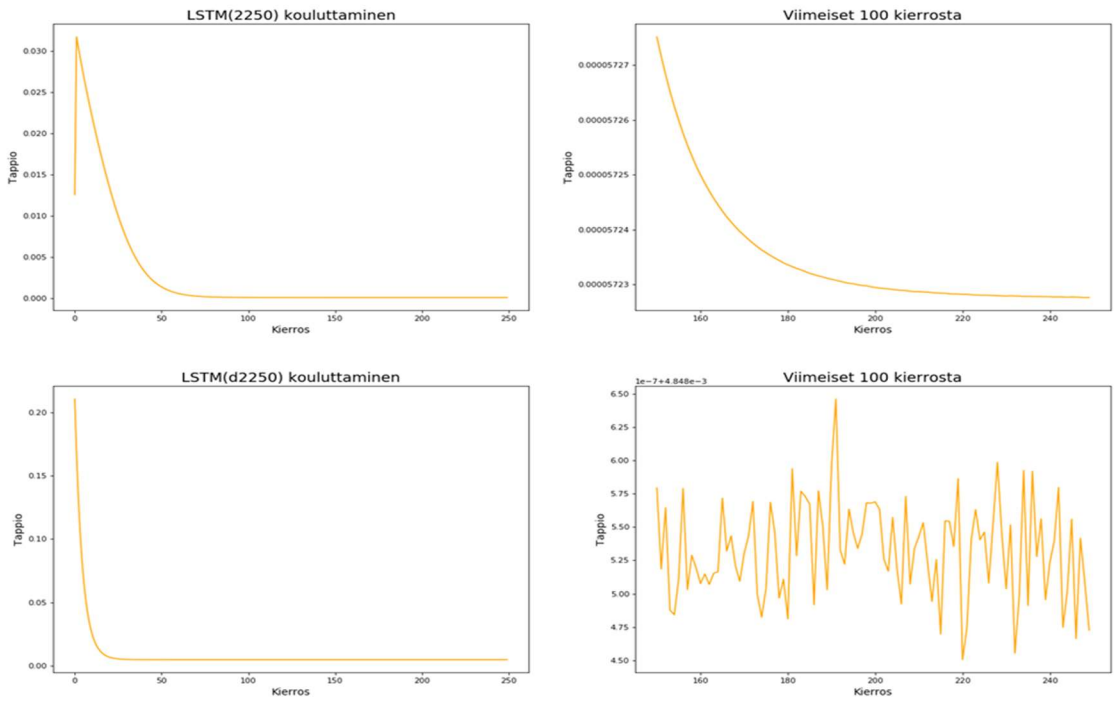
Kuvion 10 kouluttamiskäyrät vastaavat RMSE-tulosten keskiarvoja, joita esitellään luvussa 5.3. Kuvion 10 y-akseleilla ovat kouluttamisessa käytetty MSE:n arvot eli tappiot, ja x-akseleilla ovat kierrosten lukumäärät. Kuvion 10 vasemmanpuoleiset käyrät eroavat oikeista siinä, että ne sisältävät kaikki kouluttamiskierrokset, kun oikeanpuoleiset käyrät sisältävät viimeiset 100 kouluttamiskierrosta. Kuvion 10 vasemmanpuoleiset käyrät laskevat jyrkästi ensimmäisten 30 kierroksen aikana, mikä viittaa nopeaan oppimiseen ja liian suureen oppimisasteeseen. Vasemmanpuoleisten käyrien suurin ero on ylemmän käyrän jyrkkä kasvu ensimmäisten kouluttamiskierrosten aikana. Tätä voidaan pitää hyväksyttävänä RMSE-tulosten valossa. Ylempää käyrää vastaavat viimeiset 100 kierrosta näyttävät pienenentyvät hyvin alle 180 kierrokseen asti, jonka jälkeen käyrä nousee 20 kierroksen ajan. Tämä tarkoittaa, että verkko ei opi hyvin. Differentioidun aikasarjan viimeiset 100 kouluttamiskierrosta vähenevät lineaarisesti. Tämä viittaa siihen, että verkko ei kykene oppimaan riittävästi 250 kierroksen aikana. Toisaalta kierrosten lisääminen ei takaa, että verkon ennustamistarkkuus paranee, mikä havaittiin kokeile ja erehdy -taktiikalla. Lisäksi kouluttamiskierrosten rajoittaminen 250 kierrokseen mahdollistaa verkkojen paremman vertailun sekä niiden ajallinen ajoaika on vielä kohtalainen. MLP-verkolle ei löydetty sopivampia hyperparametreja pidemmän aikasarjan tapauksessa. Neuroverkkojen kouluttamiskäyrät lyhyempien aikasarjojen osalta on esitelty liitteissä.

Kuviossa 11 ovat RNN-verkon kouluttamiskäyrät, kun verkkoa on koulutettu 2 250 havainnon aikasarjalla sekä sen differentioidulla versiolla. Ylemmät käyrät vastaavat 50 kouluttamiskerrasta 6. kertaa ja alempi 33. kertaa. Kuvion 11 RNN-verkon kouluttamiskäyrät ovat esitettynä kuten MLP-verkolla. Kuvion 11 kouluttamiskäyrät vastaavat RMSE-tulosten keskiarvoja, joita esitellään luvussa 5.3. Kuvion 11 kolme neljästä käyrästä vastaavat muodoltaan MLP-verkon vastaavia käyriä. Huolimatta kuvion 11 oikeanpuolen ylemmän käyrän satunnaisesta sahaamisesta, tappio pienenee viimeisten 100 kierroksen aikana, jolloin verkko oppii. Kuvioiden 10 ja 11 kouluttamiskäyrien ero on, että RNN verkko saavuttaa pienemmät tappio arvot 250 kierroksen aikana.



Kuvio 11 RNN-verkon kouluttamiskäyrät.

Kuviossa 12 ovat LSTM-verkon kouluttamiskäyrät, kun verkkoa on koulutettu 2 250 havainnon aikasarjalla sekä sen differentioidulla versiolla. Ylemmät käyrät vastaavat 50 kouluttamiskerrasta 12. kertaa ja alempi 1. kertaa. Kuvion 12 LSTM-verkon kouluttamiskäyrät ovat esitettynä kuten MLP- ja RNN-verkoilla. Kuvion 12 kouluttamiskäyrät vastaavat RMSE-tulosten keskiarvoja, joita esitellään luvussa 5.3.



Kuvio 12 LSTM-verkon kouluttamiskäyrät.

Kuvion 12 vasemmanpuoleiset käyrät eroavat kahden edellisen verkon kouluttamiskäyristä siinä, että LSTM-verkolla tappio vähenee tasaisemmin. Täten verkko oppii tasaisemmin kuin kaksi edellistä verkkoa. Tämä näkyy hyvin kuvion 12 oikean puolen ylemmästä käyrästä. Kuvion 12 oikean puolen alempi käyrä kertoo, että verkko ei opi merkittävästi viimeisen 100 kouluttamiskierroksen aikana. Kouluttamiskäyrän sahaaminen vähentäisi LSTM-verkon luotettavuutta eri ennustamiskertojen välillä, jos tappio ei olisi niin pieni kuin mihin verkko on päätynyt. Käyrän sahaaminen voitaisiin estää rajoittamalla LSTM-verkon kouluttamiskierrokset 100 kertaan.

Neuroverkkojen hyperparametreja pyrittiin säätämään siten, että kaikkien verkkojen kouluttamiskäyrät olisivat muistuttaneet kuvion 12 oikeaa yläkulmaa, jonka muoto vastaa kouluttamiskäyrien ideaalitilannetta. Tämä johti siihen, että jokaisella neuroverkolla hyperparametrit olivat erilaiset jokaista aikasarjaa kohden. Tällöin lähes kaikilla verkoilla RMSE-tulosten vaihtelu lisääntyi 50 ennustamiskerran aikana. Täten päädyttiin käyttämään hyperparametrien valinnassa sellaisia arvoja, jotka paransivat RMSE-tulosten tarkkuutta, mutta heikensivät kouluttamiskäyrän muotoa.

Ristihakualgoritmilla löydetyt parhaat kolme ARIMA-mallia 750 sekä 2 250 havainnon aikasarjoille estimoitiin ja niitä vertailtiin käyttämällä AIC ja BIC arvoja. Taulukossa 4 on esitetty mallien informaatiokriteerien tulokset.

Taulukko 4 ARIMA-mallien informaatiokriteerit.

Informaatiokriteerit		
S&P 500 aikasarja: 750 havaintoa		
Malli	AIC	BIC
ARIMA(1,1,0)	4 331,64	4 344,28
ARIMA(0,1,1)	4 343,71	4 336,03
ARIMA(2,1,0)	4 329,58	4 346,43
S&P 500 aikasarja: 2 250 havaintoa		
Malli	AIC	BIC
ARIMA(1,1,0)	12 356,76	12 356,76
ARIMA(0,1,1)	12 340,76	12 356,69
ARIMA(0,1,2)	12 341,23	12 363,08

Taulukon 4 tulokset osoittavat, että mallien erot ovat pieniä molemmilla aikasarjoilla, kun tarkastellaan AIC ja BIC informaatiokriteerien arvoja. Täten sopivan mallin valinta keskittyi diagnostisiin tarkasteluihin.

Taulukossa 5 on esitelty ARIMA-mallien residuaalien tilastolliset tunnusluvut sekä lyhyen että pitkän aikavälin aikasarjoille. Pienimmät residuaalien keskiarvot saavutettiin ARIMA(2,1,0)- ja ARIMA(0,1,2)-malleilla.

Taulukko 5 ARIMA-mallien residuaalien tilastolliset tunnusluvut.

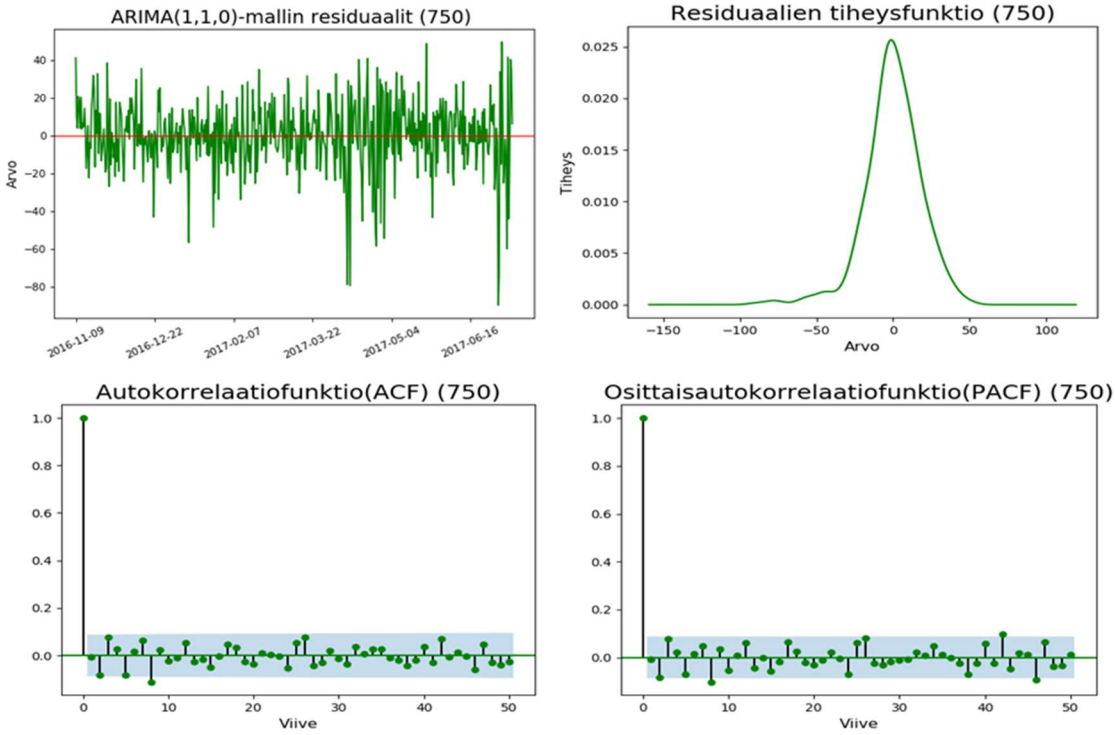
Laajemmat tulokset ARIMA-mallien estimoinneista ovat liitteissä.

Residuaalien tilastolliset tunnusluvut						
S&P 500 aikasarja: 750 havaintoa						
Malli	Keskiarvo[1)]	Keskihajonta	Vinous	Huipukkuus	Minimi	Maksimi
ARIMA(1,1,0)	0,64	18,46	-0,815	2,80	-89,74	49,59
ARIMA(0,1,1)	0,84	18,45	-0,837	2,86	-89,77	49,61
ARIMA(2,1,0)	0,15	18,38	-0,864	2,88	-89,28	53,00
S&P 500 aikasarja: 2 250 havaintoa						
Malli	Keskiarvo[2)]	Keskihajonta	Vinous	Huipukkuus	Minimi	Maksimi
ARIMA(1,1,0)	-0,291	14,812	-0,273	5,672	-99,673	92,479
ARIMA(0,1,1)	-0,312	14,811	-0,273	5,661	-99,604	92,402
ARIMA(0,1,2)	-0,037	14,807	-0,317	5,662	-101,365	89,819

[1)] Keskiarvo-sarakkeen rivien arvot on skaalattu kertomalla rivit 10^{-2}.
[2)] Keskiarvo-sarakkeen rivien arvot on skaalattu kertomalla rivit 10^{-3}.

Taulukossa 5 ARIMA(1,1,0)-mallin residuaalien jakautuman vinous ja huipukkuus ovat pienimmät, kun aikasarja sisälsi 750 havaintoa. ARIMA(0,1,1)-mallin residuaalien huipukkuus on pienin, kun aikasarja sisälsi 2 250 havaintoa. Residuaalien ollessa normaalijakautuneita ARIMA-mallin parametrit kykenevät kaappaamaan aikasarjan taustalla olevan prosessin.

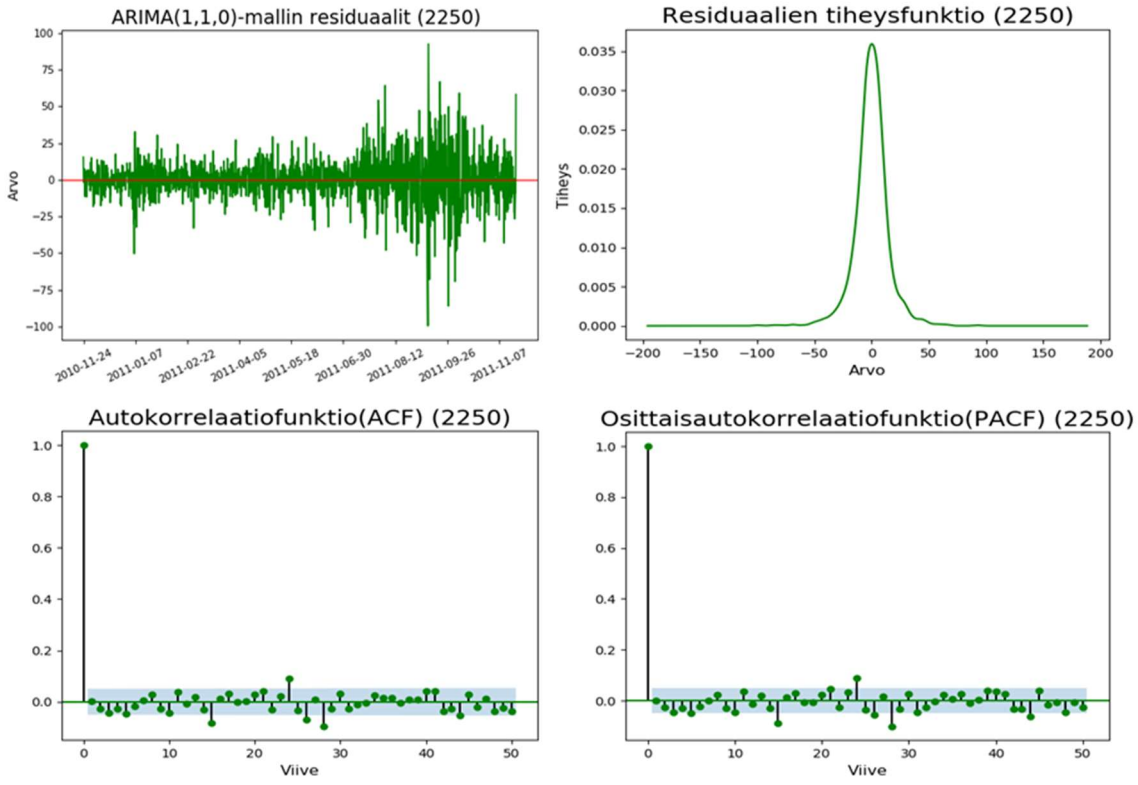
Kuviossa 13 ovat ARIMA(1,1,0)-mallin residuaalien aikasarja, jakautuman muoto, ACF- ja PACF-kuvaajat, kun aikasarja sisälsi 750 havaintoa.



Kuvio 13 ARIMA(1, 1, 0)-mallin residuaalit.

Kuvion 13 ARIMA(1,1,0)-mallin residuaalien aikasarja näyttää vaihtelevan nollan ympärillä suhteellisen vakioisesti muutamaa piikkiä lukuun ottamatta. Residuaalien jakautuman muoto vahvistaa, että aikasarjan keskiarvo on likimain nolla, mutta ei normaalijakautunut. Residuaalien ACF- ja PACF-kuvaajat näyttävät, että kriittisen rajan ylittäviä estimoituja autokorrelaatiokertoimia ei esiinny muutamaa piikkiä lukuun ottamatta 50 viipymän aikana. Nämä muutamat piikit voidaan tulkita satunnaisvaihtelusta johtuvaksi. Koska ARIMA(1,1,0)-mallin residuaalit muistuttivat eniten normaalijakautuneita 750 havainnon aikasarjalla, päädyttiin käyttämään ARIMA(1,1,0)-mallia neuroverkkojen ennustamiskyvyn lähtökohtana lyhyempien aikasarjojen tapauksessa.

Kuviossa 14 ovat ARIMA(1,1,0)-mallin residuaalien aikasarja, jakautuman muoto, ACF- ja PACF-kuvaajat, kun aikasarja sisälsi 2250 havaintoa.



Kuvio 14 ARIMA(1, 1, 0)-mallin residuaalit.

Kuviossa 14 residuaalien vaihtelu voimistuu loppupää arvojen osalta. Residuaalien jakautuman muoto on keskittynyt enemmän nolla ympärille kuin kuvion 13 vastaava. Residuaalien ACF- ja PACF-kuvaajat näyttävät, että kriittisen rajan ylittäviä estimoituja autokorrelaatiokertoimia ei esiinny säännöllisesti 50 viipymän aikana. Satunnaisesti ylittävät piikit voidaan tulkita satunnaisvaihtelusta johtuvaksi. Koska pidemmän aikasarjan ARIMA-malleilla ei ollut merkittäviä eroja residuaalien tilastollisissa tunnusluvuissa,

päädyttiin painottamaan mallien ennustamisvirhettä. Täten ARIMA(1,1,0)-mallia käytetään neuroverkkojen ennustamiskyvyn lähtökohtana pidempien aikasarjojen tapauksessa.

5.3 Ennustamisen tulokset

Neuroverkkojen ennustamiskykyä arvioitiin laskemalla yhteen kaikki RMSE-tulokset 50 ennustamiskerrasta, joista laskettiin niiden keskiarvo, varianssi, keskihajonta, minimi ja maksimi. Tämä tehtiin jokaisella verkolla jokaista aikasarjaa kohden ja niiden tulokset ovat taulukossa 5. Lisäksi taulukossa 5 on ARIMA-mallien RMSE-tulokset ja menetelmien ajoajat, käytettiin CPU:ta niiden laskemisessa.

Taulukko 6 Menetelmien RMSE-tulokset.

Neuroverkot koulutettiin 50 kertaa ja jokaisen kouluttamiskerran jälkeen niillä ennustettiin. ARIMA-mallit mallinnettiin kerran, jonka jälkeen niillä ennustettiin.

S&P 500 aikasarja: 750, 50 krt						
Neuroverkko	Aika[a)]	Keskiarvo[2)]	Varianssi	Keskihajonta	Minimi	Maksimi
MLP	23 min	29,612	1,748	1,322	27,063	32,667
RNN	29 min	**25,417**	<0,001	0,01	25,395	25,439
LSTM	43 min	27,432	0,076	0,275	26,899	28,044

S&P 500 aikasarja: d750, 50 krt						
Neuroverkko	Aika[a)]	Keskiarvo[2)]	Varianssi	Keskihajonta	Minimi	Maksimi
MLP	24 min	4,836	1,877	1,37	2,427	7,799
RNN	30 min	4,363	3,407	1,846	1,052	8,663
LSTM	42 min	**1,164**	0,05	0,223	0,833	2,056

S&P 500 aikasarja: 750, 1 krt		
ARIMA	Aika[a)]	RMSE[2)]
ARIMA(1,1,0)	3,59 s	**25,309**
ARIMA(0,1,1)	4,15 s	25,318
ARIMA(2,1,0)	7,27 s	25,431

S&P 500 aikasarja: 2250, 50 krt						
Neuroverkko	Aika[a)]	Keskiarvo[2)]	Varianssi	Keskihajonta	Minimi	Maksimi
MLP	67 min	99,324	191,518	13,839	72,455	138,652
RNN	84 min	**21,576**	0,002	0,05	21,46	21,698
LSTM	115 min	21,849	0,08	0,283	21,24	22,637

S&P 500 aikasarja: d2250, 50 krt						
Neuroverkko	Aika[a)]	Keskiarvo[2)]	Varianssi	Keskihajonta	Minimi	Maksimi
MLP	71 min	4,073	1,075	1,037	2,057	6,558
RNN	91 min	3,742	2,034	1,426	0,888	7,495
LSTM	126 min	**0,965**	0,03	0,174	0,69	1,422

[1)] Ajallinen kesto, kun on käytetty CPU:ta tulosten laskemisessa.
[2)] Sarakkeen pienin arvo on lihavoitu jokaiselle aikasarjalle.

Taulukko 6 Menetelmien RMSE-tulokset. (Jatkuu)

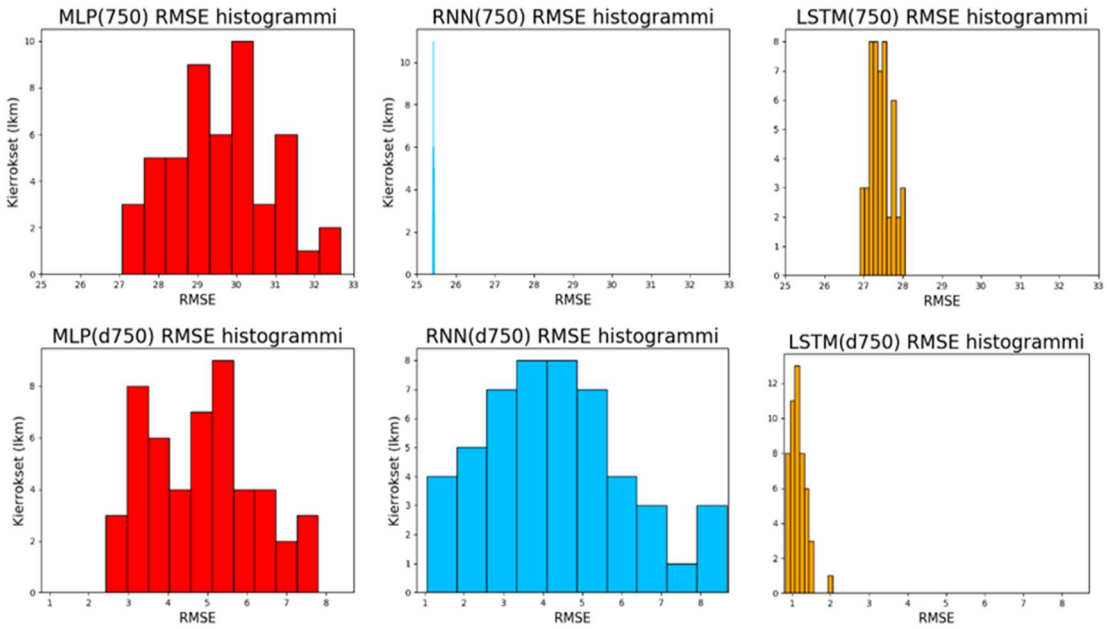
Neuroverkot koulutettiin 50 kertaa ja jokaisen kouluttamiskerran jälkeen niillä ennustettiin. ARIMA-mallit mallinnettiin kerran, jonka jälkeen niillä ennustettiin.

S&P 500 aikasarja: 2250, 1 krt		
ARIMA	Aika[a)]	RMSE[2)]
ARIMA(1,1,0)	10,74 s	**21,039**
ARIMA(0,1,1)	9,43 s	21,040
ARIMA(0,1,2)	24,58 s	21,047

[1)] Ajallinen kesto, kun on käytetty CPU:ta tulosten laskemisessa.
[2)] Sarakkeen pienin arvo on lihavoitu jokaiselle aikasarjalle.

Huomion arvoista taulukossa 5 on, että RNN-verkko kykeni ennustamaan neuroverkoista parhaiten 750 sekä 2 250 havainnon aikasarjoilla, koska RMSE:n keskiarvot olivat pienimmät ja ne eivät vaihdelleet yhtä voimakkaasti kuin kahdella muulla verkolla. RNN-verkko on näiden aikasarjojen luotettavin neuroverkko. ARIMA(1,1,0)-malli oli tarkin ennustamismenetelmä 750 sekä 2 250 havainnon aikasarjoilla, ja niiden ajoaika kesti vain useita sekunteja. Eri pituisten aikasarjojen differentiointi näytti merkittävästi parantavat neuroverkkojen ennustamistarkkuutta, jolloin kaikki neuroverkot suoriutuivat tarkemmin kuin ARIMA-mallit. LSTM-verkko saavutti differentioiduilla aikasarjoilla tarkimmat ja luotettavimmat ennusteet. MLP-verkko näyttää taulukon 5 tuloksien perusteella menettäneet kyvyn oppia pidemmän aikasarjan taustalla olevaa prosessia, kun aikasarjaa ei differentioitu.

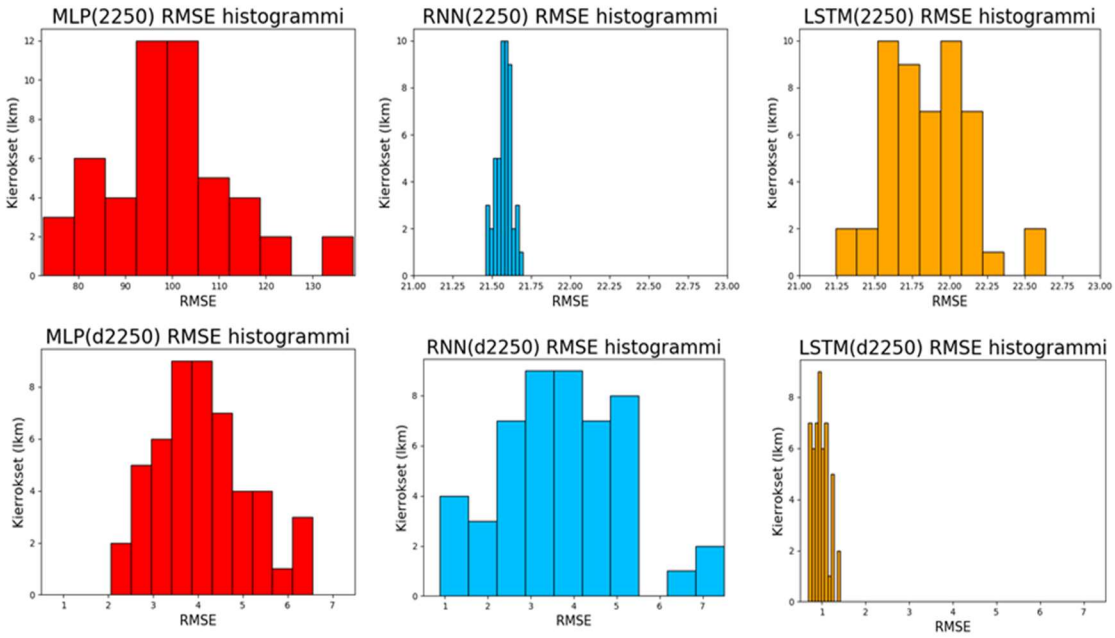
Neuroverkkojen ennustamiskyvyn luotettavuutta on havainnollistettu histogrammien avulla kuviossa 15.



Kuvio 15 Neuroverkkojen RMSE histogrammit (aikasarjat: 750 ja d750).

Kuviossa 15 vasemmalla ovat MLP-verkon histogrammit, joista ylempi vastaa 750 havainnon aikasarjaa ja alempi sen differentioitua versiota. Vastaavasti RNN-verkon histogrammit ovat kuvion 15 keskellä ja LSTM-verkon oikealla. Histogrammien y-akseleilla ovat kierrosten lukumäärät ja x-akseleilla RMSE-tulokset. Histogrammien perusteella RNN ja LSTM tarjoavat luotettavimmat verkot tehdä ennusteita.

Kuviossa 16 ovat verkkojen RMSE-histogrammit pidemmille aikasarjoille.



Kuvio 16 Neuroverkkojen RMSE histogrammit (aikasarjat: 2250 ja d2250).

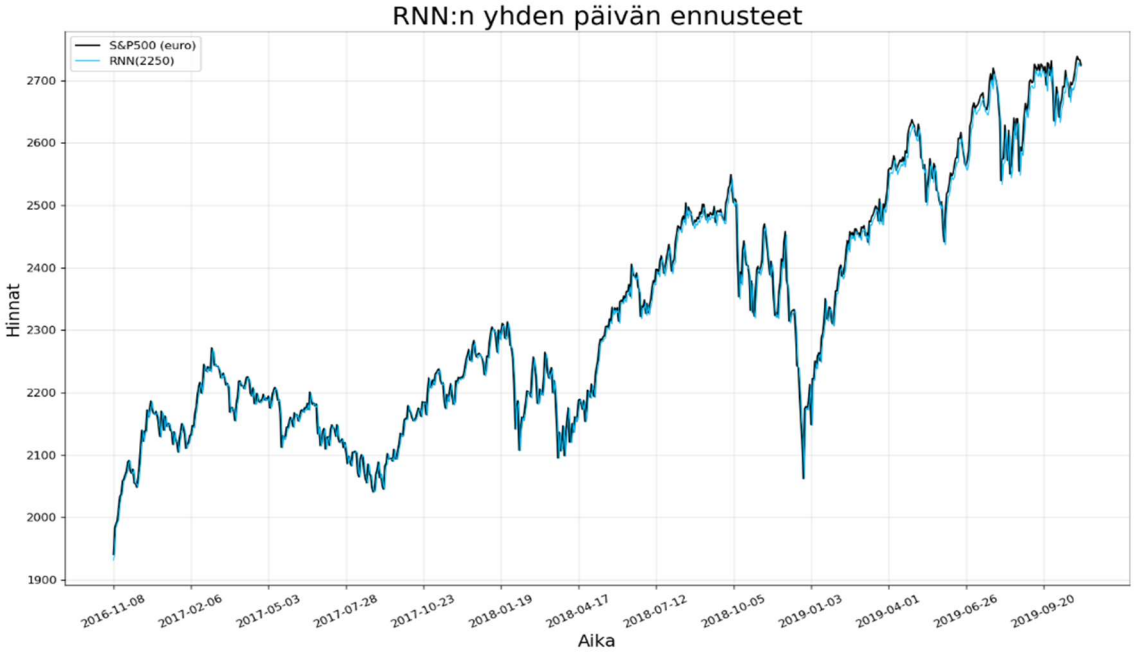
Huomioitavaa kuviossa 16 on MLP-verkon ylempi histogrammi, joka on skaalattu x-akselin suhteen eri tavalla kuin muilla verkoilla, koska verkko ei kyennyt ennustamaan tarkemmin. Kuvion 16 RNN- sekä LSTM-verkkojen histogrammit havainnollistavat näiden verkkojen ennustamistarkkuutta ja niiden luotettavuutta, kun kyseessä ovat pidempien aikasarjojen molemmat versiot.

Kuviossa 17 on MLP-verkon yhden päivän ennusteet sekä S&P 500 -indeksin vastaavat arvot, kun verkkoa on koulutettu 2 250 havainnon aikasarjalla, jota ei ole differentioitu. MLP-verkko kykenee osittain kuvaamaan indeksin muotoa, mutta ei kykene huomioimaan indeksin nousevaa trendiä. Kuviossa 17 MLP-verkon kuvaaja on 28. kerta 50:stä, jolloin RMSE ≈ 99,32.

Kuvio 17 MLP-verkon ennusteet (aikasarja: 2250).

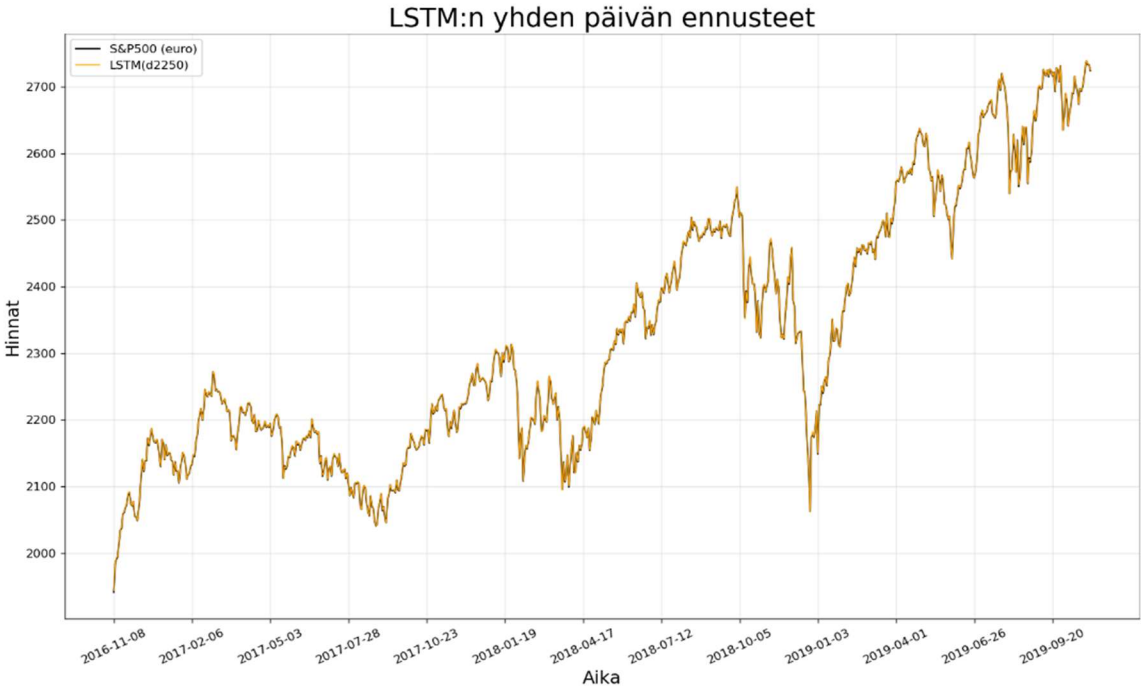
Kuvion 17 y-akselilla ovat indeksin arvot ja x-akselilla aikaväli johon arvot sijoittuvat.

Kuviossa 18 on RNN-verkon yhden päivän ennusteet sekä S&P 500 -indeksin vastaavat arvot, kun verkkoa on koulutettu 2 250 havainnon aikasarjalla, jota ei ole differentioitu. RNN-verkon sininen käyrä näyttää seuraavan hyvin indeksin mustaa käyrää. Kuvion 18 RNN-verkon ennusteet ovat 6. kerta 50:stä, jolloin RMSE ≈ 21,58.



Kuvio 18 RNN-verkon ennusteet (aikasarja: 2250).

Kuviossa 19 on LSTM-verkon yhden päivän ennusteet sekä S&P 500 -indeksin vastaavat arvot, kun verkkoa on koulutettu 2 250 havainnon aikasarjalla, joka on differentioitu. LSTM-verkon keltainen kuvaaja peittää lähes täydellisesti indeksin mustan kuvaajan. Kuviossa 19 LSTM-verkon ennusteet ovat 11. kerta 50:stä, jolloin RMSE ≈ 0,97.



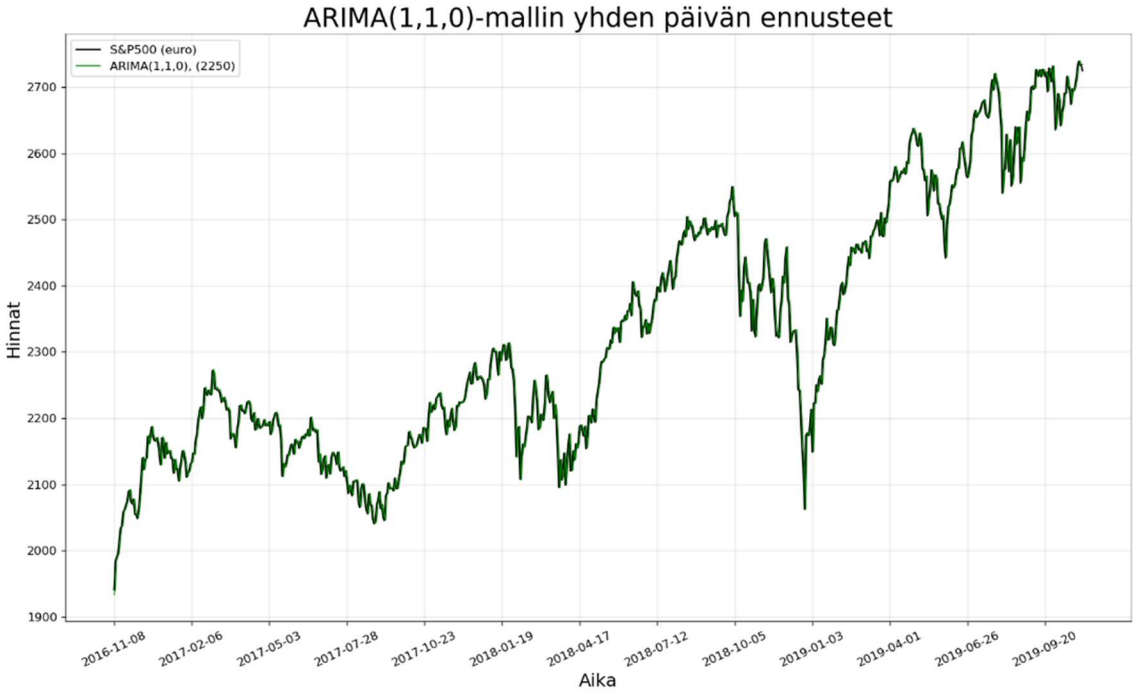
Kuvio 19 LSTM-verkon ennusteet (aikasarja: d2250).

Kuviosta 19 on nähtävissä LSTM-verkon kyvykkyys S&P 500 -indeksin ennustamisessa, kun verkko on koulutettu pidemmällä aikasarjalla ja se on differentioitu. Liitteissä on esitetty loput kuviot neuroverkkojen yhden päivän ennusteista eri aikasarjoilla.

RNN- ja LSTM-verkot näyttävät tuloksien valossa kykenevän oppimaan pidemmän aikavälin riippuvuuksia, joka on seurausta verkkojen muistiominaisuuksista. Lisäksi muistiominaisuudet parantavat verkkojen luotettavuutta sekä tarkkuutta aikasarjan pituuden kasvaessa. Kaikki neuroverkot kykenivät kaappaamaan aikasarjan epästationaarisuuden, kun aikasarjan havaintoja oli 750. Vain RNN- ja LSTM-verkot kykenivät kaappaamaan aikasarjan epästationaarisuuden, kun aikasarjan havaintoja oli 2 250. Tuloksien perusteella voidaan vastata toiseen tutkimuskysymykseen: Soveltuvatko neuroverkot taloudellisen aikasarjan mallintamiseen ja ennustamiseen? Kaikki tutkielmassa käytetyt neuroverkot soveltuvat taloudellisen aikasarjan mallintamiseen ja ennustamiseen lyhyemmillä aikasarjoilla. Vain RNN- ja LSTM-verkot soveltuvat myös pidemmille aikasarjoille. Työssä käytettiin indeksin kuvaamisessa euroja, koska vaihtamalla eurot

dollareiksi muuttuu indeksin kuvaajan muoto. Tämä edellyttää, että kokeile ja erehdy -taktiikka suoritetaan uudelleen kaikille verkoille.

ARIMA(1,1,0)-mallilla saavutettiin lähtötaso neuroverkkojen ennustamiskyvyn tarkastelulle. Tutkielman ARIMA-mallien ennustamisessa jokaisen päivittäisen arvon ennustamisen jälkeen mallit mallinnettiin uudelleen. Tässä harjoitusaineistoon lisättiin testiaineistosta aina sen päivän todellinen arvo, jota oli ennustettu. Kuviossa 20 on vihreällä ARIMA-mallin ennustekäyrä ja mustalla indeksi, kun aikasarja sisälsi 2 250 havaintoa.



Kuvio 20 ARIMA$(1, 1, 0)$-mallin ennusteet (aikasarja: 2250 ja RMSE ≈ 21,04).

Kuvio 20 havainnollistaa, että ARIMA(1,1,0)-mallin ennustekäyrä kykeni seuraamaan S&P 500 -indeksiä koko tarkastelujakson ajan, vaikkakin mallinnuksen kohteena oli taloudellinen aikasarja. Liitteissä on kuvion 20 vastaava kuvio 750 havainnon aikasarjalle.

5.4 Menetelmien robustisuus

Neuroverkkojen ristihakualgoritmin suorittamisen jälkeen kokeile ja erehdy -taktiikka suoritettiin niille hyperparametri kombinaatioille, joita neuroverkko ristihakualgoritmi ehdotti. Taktiikassa vaihdeltiin neuroverkkojen hyperparametreja ja seurattiin niiden vaikutusta verkkojen kouluttamis- ja ennustamisvaiheisiin. Usean yrityksen jälkeen selvisi, että neuroverkkojen vertailun kannalta oli hyvä yksilöidä hyperparametrit vastaamaan aikasarjaa, mutta pitäen ne samoina eri neuroverkkojen välillä. Makridakis ym. (2018) ovat valinneet MLP-verkon oppimisnopeuden väliltä 0,1 ja 1. RNN- ja LSTM-verkkojen

oppimisnopeutena he käyttävät 0,001. Tutkijoiden MLP-verkon optimointialgoritmi on skaalattu konjugaatio gradientti menetelmä (engl. Scaled Conjugate Gradient method, lyh SCG), kun taas RNN- ja LSTM-verkon tapauksessa he käyttävät RMSProp optimointialgoritmia. Lisäksi heidän verkkojen muut hyperparametrit vaihtelevat eri verkkojen välillä. Tämä on eräs syy, miksi vertailtavuus eri tutkimusten välillä on vaikeaa.

Täten ristihakualgoritmi sekä kokeile ja erehdy -taktiikka koettiin yhdessä parhaaksi menetelmäksi valita hyperparametrit. Huolimatta siitä, että ristihakualgoritmi vaatii huomattavasti tietokoneen laskentatehoa. Virhefunktion valinnassa kokeiltiin käyttää MSE:n tilalla keskimääräistä absoluuttista virhettä (engl. Mean Absolute Error, lyh. MAE). Tällä virhefunktion valinnalla verkkojen kouluttamiskäyrät muuttuivat satunnaisemmiksi, joka viittasi alisovittamisesta johtuviin ongelmiin. Esimerkiksi viimeisen sadan kierroksen aikana tappioarvot saattoivat heitellä jonkin keskiarvon ympärillä, tehden käyrien tulkinnan vaikeaksi. Kierrosten lukumäärän kasvattamisella oli osalla verkkoista kouluttamis- ja ennustamisvaihetta parantava vaikutus, mutta ainoastaan tiettyyn rajaan asti. Vaihtoehtoiset optimointialgoritmit, kuten Makridakis ym. (2018) käyttämät, rajattiin tutkielman ulkopuolelle, koska valinta oikeiden hyperparametrien kannalta olisi ollut haastavampaa.

Liukuvan ikkunan ja verkon rakenteen hyperparametreilla oli suurin vaikutus tutkielman neuroverkkojen oppimis- ja ennustamiskykyyn. Lisäksi liukuvan ikkunan ja verkon rakenteen hyperparametrit olivat vahvasti yhteydessä toisiinsa. Esimerkiksi tapauksessa, jossa ikkunan leveys oli suurempi kuin kaksi ei ollut mahdollista löytää verkon rakenteen hyperparametreja, jotka olisivat toimineet kaikilla neuroverkoilla samankaltaisesti. Sekoituspuskurin laittaminen ykköseksi toimi paremmin, kun aikasarjoja ei differentioitu. Kaikilla verkoilla piiloyksiköiden lukumäärän kasvattaminen vaikutti positiivisesti ennustamistarkkuuteen tiettyyn rajaan asti. Piiloyksiköiden aktivointifunktioina käytettiin niitä, jotka olivat valmiina Moroneyn (2019) koodeissa.

ARIMA-mallin käyttö antoi hyvän lähtötason neuroverkkojen ennustamiskyvyn arvioinnille. Lisäksi ARIMA-malli oli luotettavampi ennustamisessa kuin neuroverkot johtuen yhdestä ennustamiskerrasta, kun aikasarjoja ei differentioitu. Neuroverkot näyttävät kirjallisuuskatsauksen perusteella kykenevän tarkempiin ennustamistuloksiin, kun taloudellisen aikasarjan ennustamisessa käytetään useita sitä selittäviä aikasarjoja. Tutkielman tuloksien valossa ARIMA-mallit sekä neuroverkot soveltuvat molemmat ennustamaan yksittäistä taloudellista aikasarjaa. Aikasarjan käsittely näyttää tuloksien perusteella vaikuttavan eniten neuroverkkojen ennustamistarkkuuteen.

6 YHTEENVETO

Tutkielman tarkoituksena oli analysoida kolmen eri neuroverkon ennustamiskykyä. Tutkielman neuroverkoiksi valittiin MLP, RNN ja LSTM, koska ne ovat olleet rahoitustutkimuksien ”kuuma peruna”. Rahoitusteoriaa käsittelevässä luvussa esiteltiin keskeisemmät rahoitusteoriat tutkielman kannalta sekä havainnollistettiin niiden historiallista taustaa. Rahoitusteorian esittelyllä haettiin perusteluita taloudellisen aikasarjan ennustamiselle, koska teoriat kuten EMH-hypoteesi ja aikasarjan momentum-teoria ovat ristiriitaisia keskenään. Empiirisesti on havaittu, että markkinat eivät ole täydellisen tehokkaita, mikä mahdollisti ensimmäiseen tutkimuskysymykseen vastaamisen: Voidaanko neuroverkoilla ennustaa arvopapereiden hintoja käyttäen historiallista aikasarjaa? Empiirisesti havaittujen tuloksien perusteella neuroverkoilla voidaan ennustaa arvopapereiden hintoja käyttäen historiallista aikasarjaa.

Täten tutkielman luvussa 3 tehtiin katsaus tekoälyyn, koneoppimiseen ja syväoppimiseen sekä esiteltiin tutkielmassa käytettävät neuroverkot että niiden oppimisen taustalla oleva vastavirta-algoritmi. Luvun lopuksi tehtiin katsaus aiemmin tehtyihin tutkimuksiin, jotka vahvistivat näkemystä, että neuroverkot soveltuvat taloudellisten aikasarjojen ennustamiseen. Toisaalta tutkimuksissa ei ollut yhtenäistä lähestymistapaa sille, kuinka neuroverkot tulisi rakentaa, joten tutkielmassa käytettiin ristihakualgoritmia sekä kokeile ja erehdy -taktiikkaa. Luvussa 4 esiteltiin aineisto, sen käsittely, neuroverkkojen hyperparametrit, ARIMA-malli ja menetelmien ohjelmointi. Luvussa 5 esiteltiin tulokset, joiden perusteella neuroverkoista RNN ja LSTM ovat tarkimpia sekä luotettavimpia ennustamisessa. Tämä luotettavuus syntyy verkkojen muistiominaisuuksista ja niiden tarkkuus riippuu paljolti aineiston käsittelystä. Tuloksien perusteella voidaan vastata tutkielman toiseen tutkimuskysymykseen: Soveltuvatko neuroverkot taloudellisen aikasarjan mallintamiseen ja ennustamiseen? Kaikki tutkielmassa käytetyt neuroverkot soveltuvat taloudellisen aikasarjan mallintamiseen ja ennustamiseen lyhyemmillä aikasarjoilla. Vain RNN- sekä LSTM-verkot soveltuvat myös pidemmille aikasarjoille. ARIMA-mallilla ennustaminen havainnollisti, että neuroverkkojen ennustamiskyky sekä luotettavuus ovat riittäviä. Neuroverkkojen ennustamiskyky paranee aineiston differentioinnin myötä. ARIMA-malli oli tutkielman tarkin ja luotettavin, kun neuroverkoille syötettäviä aikasarjoja ei differentioitu.

Täten tutkielman tuloksilla on merkitystä, kun valitaan ennustamismenetelmiä. Neuroverkkoja ei kannata poissulkea, koska verkot saattavat edustaa tulevaisuudessa

uudenlaista laskutikkua, joka on kalibroitavissa tilanteen mukaan. Neuroverkkojen heikkous on, että ne vaativat jatkuvaa rakenteen uudelleen muokkaamista sekä niiden kouluttamista, jos niitä sovelletaan käytännössä osakemarkkinahintojen ennustamiseen, koska aikasarjojen käyttäytyminen muuttuu ajassa. Aineiston käsittely on tässä merkittävässä osassa. Tutkielman jatkotutkimusaihe liittyy neuroverkkojen mukautuvuuteen, jos esimerkiksi aineiston havainnot muutettaisiin dollareiksi eurojen sijaan, niin kuinka tarkasti neuroverkot ennustaisivat tutkielman hyperparametreilla?

LÄHTEET

Ahmed, N. K. – Atiya, A. F. – Gayar, N. – El-Shishiny, H. (2010) An Empirical comparison of machine learning models for time series forecasting. *Econometric Reviews.* Vol. 29 (5-6), 594–621.

Atsalakis, G. S. – Valavanis, K. P. (2009) Surveying stock market forecasting techniques – Part II: Soft computing methods. *Expert Systems with Applications.* Vol. 36, 5932–5941.

Bao W. – Yue, J. – Rao, Y. (2017) A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLoS ONE.* Vol. 12 (7).

Box, G. – Jenkins, G. – Reinsel, G. – Ljung, G. (2016) *Time series analysis: forecasting and control.* 9. p. John Wiley & Sons, New Jersey.

Brownlee, J. (2019). Machine learning mastery. <https://machinelearningmastery.com>, haettu 1.6.2019.

Colah's Blog (2015) Understanding LSTM networks. <http://colah.github.io/posts/2015-08-Understanding-LSTMs>, haettu 20.11.2019.

Coursera (2020) TensorFlow sequences time series and prediction. <https://pt.coursera.org/lecture/tensorflow-sequences-time-series-and-prediction/outputting-a-sequence-wNWnn>, haettu 25.2.2020

Elton, E. J. – Gruber, M. J. – Brown, S. J. – Goetzmann, W. N. (2014*) Modern portfolio theory and investment analysis.* 9. p. John Wiley & Sons, Danvers.

Fama, E. F. (1970) Efficient capital markets: A review of theory and empirical work. *The Journal of Finance.* Vol. 25 (2), 383–417.

Fama, E. F. (1991) Efficient capital markets: II. *The Journal of Finance.* Vol. 46 (5), 1575–1617.

Franses, P. H. – van Dijk, D. (2000) *Nonlinear time series models in empirical finance.* Cambridge University Press, Cambridge.

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks.* Springer.

Grossman, S. (1976) On the efficiency of competitive stock markets where trades have diverse information. *Journal of Finance.* Vol. 31, 573–585.

Grossman, S. – Stiglitz, J. (1980) On the impossibility of informationally efficient markets. *American Economic Review.* Vol. 70, 393–408.

Hamid, S. A – Iqbal Z. (2004) Using neural networks for forecasting volatility of S&P 500 index futures prices. *Journal of Business Research.* Vol. 57, 1116–1125.

Hawley, D. D. – Johnson, J. D. – Raina D. (1990) Artificial neural systems: a new tool for financial decision-making. *Financial Analysts Journal.* Vol. 46 (6), 63–72.

Haykin, S. (2009) *Neural networks and learning machines.* 3. p. Prentice Hall, New Jersey.

Hochreiter, S. – Schmidhuber, J. (1997) Long Short-Term Memory. *Neural Computation.* Vol. 9(8), 1735–1780.

Hochreiter, S. – Bengio, Y. – Frasconi, P. – Schmidhuber, J. (2001) Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Neural Networks.* IEEE Press, 237–243.

Huck, N. (2009) Pairs selection and outranking: An application to the S&P 100 index. *European Journal of Operational Research.* Vol. 196 (2), 819–825.

Huck, N. (2010). Pairs trading and outranking: The multi-step-ahead forecasting case. *European Journal of Operational Research.* Vol. 207 (3), 1702–1716.

Hull, J. C. (2012) *Options, futures and other derivatives.* 8. p. Prentice Hall, New York.

Jensen, M. C. (1978) Some anomalous evidence regarding market efficiency. *Journal of Financial Economics.* Vol. 6, 95–101.

Kahra, H. – Kanto, A. (1999) *Aikasarjaekonometria.* Opetusmoniste. Turun kauppakorkeakoulu, Turku.

López de Prado, M. (2018) *Advances in financial machine learning.* Wiley, New Jersey.

Makridakis, S. – Spiliotis, E. – Assimakopoulos, V. (2018) Statistical and Machine Learning forecasting methods: Concerns and ways forward. (Research Article). *PLoS ONE.* Vol. 13 (3).

Malkiel, B. G. (2003) The efficient market hypothesis and its critics. *Journal of economic perspectives.* Vol. 17 (1), 59–82.

Malliaris, M. – Malliaris, A. G. (2013) Neural network forecasting with the S&P 500 index across decades. *Proceedings of the International Conference on Data Mining (DMIN).* Vol. 1 (1).

McCulloch, W. S. – Pitts, W. (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics.* Vol. 5, 115–133.

Minsky, M. L. – Papert, S. (1969) *Perceptrons: An introduction to computational geometry.* MIT Press, Cambridge.

Moritz, B. – Zimmermann, T. (2014) Deep conditional portfolio sorts: The relation between past and future stock returns. *Working paper.* LMU Munich and Harvard University.

Moroney, L. (2019) TensorFlow in Practice. (GitHub). <https://github.com/lmoroney/dlaicourse/tree/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP>, haettu 20.11.2019.

Moskowitz, T. J. – Ooi, Y. H. – Pedersen, L. H. (2012) Time series momentum. *Journal of Financial Economics.* Vol. 104, 228–250.

Mostafa, F. – Dillon, T. – Chang, E. (2017) *Computational intelligence applications to option pricing, volatility forecasting and value at risk.* Vol. 697, Springer, Cham.

Niaki, S. – Hoseinzade, S. (2013) Forecasting S&P 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International.* Vol. 9 (1), 1–9.

Niskanen, J. – Niskanen, M. (2016) *Yritysrahoitus.* Edita Publishing Oy, Helsinki.

Nvidia (2019) Deeplearning. <https://developer.nvidia.com/deep-learning>, haettu 2.10.2019.

Rosenblatt, F. (1958). The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review.* Vol. 65, 386–408.

Rubinstein, M. (1994) Implied binomial trees. *Journal of Finance.* Vol. 49 (3), 771–818.

Russell, S.J. – Norvig, P. (2010) *Artificial intelligence: A modern approach.* 3. p. Prentice Hall, New Jersey.

Tuominen, H. (2020) Johdatus tekoälyn taustalla olevaan matematiikkaan. <https://tim.jyu.fi/view/kurssit/tie/tiep1000/tekoaly-mat/moniste>, haettu 22.1.2020.

LIITTEET

Liite 1 Versioiden tarkistus

```
#------------------------------------------------------------
# Import and checks libraries' versions
#------------------------------------------------------------
# scipy
import scipy
print('scipy: %s' % scipy.__version__)

# numpy
import numpy as np
print('numpy: %s' % np.__version__)

# matplotlib
import matplotlib
from matplotlib import pyplot
print('matplotlib: %s' % matplotlib.__version__)

# pandas
import pandas
from pandas import read_csv
print('pandas: %s' % pandas.__version__)

# statsmodels
import statsmodels
print('statsmodels: %s' % statsmodels.__version__)

# scikit-learn
import sklearn
print('sklearn: %s' % sklearn.__version__)

# tensorflow
import tensorflow as tf
print('tensorflow: %s' % tf.__version__)

scipy: 1.4.1
numpy: 1.18.1
matplotlib: 3.1.3
pandas: 1.0.1
statsmodels: 0.11.0
sklearn: 0.22.1
tensorflow: 2.1.0
```

Liite 2 Tilastollinen testaus (ADF-testi)

```
#-----------------------
# Downloading libraries
#-----------------------
import pandas as pd
from numpy import log
from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt
from numpy import mean
```

```
from numpy import var
from numpy import std

#-----------------------------
# Import data & Preparing data
# Data: “S&P500 index, euro”
#-----------------------------
data = pd.read_csv(r' file_path \S&P500_eur.csv', parse_dates=True,
squeeze=True)
prices = data['Close']
#A
nprices = prices
#B
lprices = log(prices)
#C
prices1 = prices.diff(periods=1)
dprices = prices1[1:750]

#-------------------------------------------------------------------------
#Calculate summary stats
#From: https://machinelearningmastery.com/statistics-for-machine-learning-
mini-course/
#-------------------------------------------------------------------------
# Change prices to #A, #B or #C
series = np.array(prices)
print('Mean: %.3f' % mean(series))
print('Variance: %.3f' % var(series))
print('Standard Deviation: %.3f' % std(series))
print('Minimum: %.3f' % min(series))
print('Maximum: %.3f' % max(series))

#-----------------------------------------------------------------
# ADF Statistic
#Brownlee(2019),
from: https://machinelearningmastery.com/time-series-data-stationary-python/
#-----------------------------------------------------------------
#Choose prices #A, #B or #C
result = adfuller(dprices)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
      print('\t%s: %.3f' % (key, value))
```

Liite 3 Neuroverkkojen koodit

```
#-----------------------------------------------------------
### Load libraries ###
#-----------------------------------------------------------
import timeit
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
from numpy import mean
```

```
from numpy import var
from numpy import std

#-------------------------------------------------------------
### Making needed functions ###
# (1) Orginal code from: Laurence Moroney
# (2) Orginal code from: Jason Brownlee
# (3) Modified from (1) & (2)
#-------------------------------------------------------------
# Create a differenced series (2)
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return diff

# Invert differenced forecast (2)
def inverse_difference(last_ob, value):
    return value + last_ob

# Root mean squared error or rmse (2)
def measure_rmse(actual, predicted):
    return sqrt(mean_squared_error(actual, predicted))

# Split a univariate dataset into train/test sets (2)
def train_test_split(data, split_time):
    return data[:split_time], data[split_time:]

# Windowed dataset (1)
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
    dataset = dataset.shuffle(shuffle_buffer).map(lambda window: (window[:-
1], window[-1]))
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset

# Fit a model (3)
# Choose model for fitting
def model_fit(train, config, i):
    # clear session
    tf.keras.backend.clear_session()
    # unpack config
    window_size, batch_size, shuffle_buffer_size, n_nodes, n_epochs = config
    # windowed dataset
    train_set = windowed_dataset(train, window_size, batch_size=batch_size,
shuffle_buffer=shuffle_buffer_size)
    # define MLP model
    #model = tf.keras.models.Sequential([
        #tf.keras.layers.Dense(n_nodes, input_shape=[window_size], activa-
tion="relu"),
        #tf.keras.layers.Dense(1)])
    # define RNN model
    #model = tf.keras.models.Sequential([
        #tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1), in-
put_shape=[None]),
        #tf.keras.layers.SimpleRNN(n_nodes),
```

```
        #tf.keras.layers.Dense(1)])
    # define LSTM model
    model = tf.keras.models.Sequential([
        tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1), in-
put_shape=[None]),
        tf.keras.layers.LSTM(n_nodes),
        tf.keras.layers.Dense(1)])
    # compile and fit the model
    optimizer = tf.keras.optimizers.SGD(lr=1e-5, momentum=0.4)
    model.compile(loss="mse", optimizer=optimizer)
    history = model.fit(train_set, epochs=n_epochs, verbose=0)
    # plot loss per epochs
    loss = history.history['loss']
    epochs = range(len(loss))
    plot_loss_all(loss, epochs, i)
    # plot all but the first 150 loss per epochs
    epochs = range(70, len(loss))
    plot_loss = loss[70:]
    plot_loss_last(plot_loss, epochs, i)
    return model

# Forecast with the fitted model (3)
def model_predict(model, series, config, split_time):
    # unpack config
    window_size, _, _, _, _ = config
    forecast = []
    for time in range(len(series) - window_size):
        forecast.append(model.predict(series[time:time + window_size][np.ne-
waxis]))
    forecast = forecast[split_time - window_size:]
    results = np.array(forecast)[:, 0, 0]
    return results

# Walk-forward validation for univariate data (3)
def walk_forward_validation(data1, data2, split_time, cfg, i):
    i = i + 1
    # splitting data to valid sets
    _, x_valid = train_test_split(data1, split_time)
    _, time_valid = train_test_split(data2, split_time)
    # difference(Option 1) and scale(Option 2) the data
    # NOTICE: if Option 1 is used then delete "#" at the beginning of the Op-
tion 1 lines
    differenced = difference(data1) #Option 1
    diff = np.array(differenced) #Option 1
    values = diff.reshape((len(diff), 1)) #Choose this if Option 1 is used!
    #values = data1.reshape((len(data1), 1))  #Choose this if Option 1 is NOT
used!
    scaler = MinMaxScaler(feature_range=(0, 1))  #Choose scaling interval!
    scaler = scaler.fit(values)
    normalized = scaler.transform(values)
    pseries = normalized.reshape(len(normalized))
    split_time = split_time - 1 #Option 1
    # splitting series to train set
    x_train, _ = train_test_split(pseries, split_time)
    # fit model
    model = model_fit(x_train, cfg, i)
    # saving models architecture
    model.save(r'file_path\LSTM(d2250)_Arc_{0}.h5'.format(i)) #optional
    # forecast with the fitted model
```

```
    results = model_predict(model, pseries, cfg, split_time)
    yhat = results.reshape((len(results), 1))
    # inverse scaling transform
    inversed = scaler.inverse_transform(yhat)  #Option 1
    # invert the differenced data
    inverted = [inverse_difference(x_valid[i], inversed[i]) for i in
range(len(inversed))] #Option 1
    # save to csv file
    np.savetxt(' file_path/LSTM(d2250)_tulokset_{0}.csv'.format(i), inverted,
               delimiter=',')  #optional: choose INVERSED if data is only
scaled else choose INVERTED
    # estimate prediction error
    error = measure_rmse(x_valid, inverted)  #choose INVERSED if data is sca-
led else choose INVERTED
    #print(' > %.3f' % error) #optional
    # plot forecast
    plot_forecast(time_valid, x_valid,
                  inverted, i)  #optional: choose INVERSED if data is scaled
else choose INVERTED
    return error

# Score a model, return None on failure (3)
def repeat_evaluate(data1, data2, config, split_time, n_repeats):
    # convert config to a key
    key = str(config)
    # fit and evaluate the model n times
    scores = [walk_forward_validation(data1, data2, split_time, config, i)
for i in range(n_repeats)]
    np.savetxt(' file_path /LSTM(d2250)_RMSE.csv', scores, delimiter=',')
#optional
    # summarize score
    result = mean(scores)
    # print('> Model[%s] %.3f' % (key, result)) #optional
    return (key, result)

# Grid search configs (2)
def grid_search(data1, data2, cfg_list, split_time, n_repeats):
    # evaluate configs
    scores = [repeat_evaluate(data1, data2, cfg, split_time, n_repeats) for
cfg in cfg_list]
    # sort configs by error, asc
    scores.sort(key=lambda tup: tup[1])
    return scores

# Plot loss per epochs
def plot_loss_all(loss, epochs, i):
    fig = plt.figure(figsize=(10, 7), dpi=80)
    plt.plot(epochs, loss, color='orange')
    plt.title("LSTM(d2250) kouluttaminen", fontsize=18)
    plt.xlabel('Kierros', size=13)
    plt.ylabel('Tappio', size=13)
    #plt.show() #optional
    fig.savefig(' r' file_path /LSTM(d2250)loss_all_{0}.png'.format(i))
    plt.close(fig)
    return

# Plot all but the first 150 loss per epochs
def plot_loss_last(plot_loss, epochs, i):
    fig = plt.figure(figsize=(10, 7), dpi=80)
```

```
    plt.plot(epochs, plot_loss, color='orange')
    plt.title("Viimeiset 30 kierrosta", fontsize=18)
    plt.xlabel('Kierros', size=13)
    plt.ylabel('Tappio', size=13)
    #plt.show() #optional
    fig.savefig(r' file_path /LSTM(d2250)loss_last_{0}.png'.format(i))
    plt.close(fig)
    return

# Plot forecast
def plot_forecast(time_v, x_v, inv, i):
    fig, ax = plt.subplots(figsize=(16, 9), dpi=120)
    ax.plot(time_v, x_v, color='black', label="S&P500 (euro)", linewidth=
1.3)
    ax.plot(time_v, inv, color='orange', label="LSTM(d2250)", linewidth= 0.9)
    plt.legend(loc="upper left")
    plt.title("LSTM:n yhden päivän ennusteet", fontsize=22)
    ax.set_xlabel('Aika', size=15)
    ax.set_ylabel('Hinnat', size=15)
    ax.set_xticks(np.arange(0, len(time_v), 60))
    ax.set_xticklabels(time_v[::60], rotation=25, fontdict={'fontsize': 10})
    ax.spines["top"].set_alpha(.3)
    ax.spines["bottom"].set_alpha(1.)
    ax.spines["right"].set_alpha(.3)
    ax.spines["left"].set_alpha(1.0)
    plt.grid(axis='both', alpha=.3)
    #plt.show() #optional
    fig.savefig(r' file_path /LSTM(d2250)_{0}.png'.format(i))
    plt.close(fig)
    return

# Create a list of configs (2)
def model_configs():
    # define scope of configs
    window_size = [2]
    batch_size = [1]
    shuffle_buffer_size = [1]
    n_nodes = [250]
    n_epochs = [100]
    # create configs
    configs = list()
    for i in window_size:
        for j in batch_size:
            for k in shuffle_buffer_size:
                for l in n_nodes:
                    for m in n_epochs:
                        cfg = [i, j, k, l, m]
                        configs.append(cfg)
    #print('Total configs: %d' % len(configs))
    return configs

#-------------------------------------------------------------
### Loading data ###
#-------------------------------------------------------------
# define dataset
data_d = pd.read_csv(r' file_path \S&P500L_eur.csv', parse_dates=True,
                     squeeze=True)
data_p = pd.read_csv(r' file_path \S&P500L_eur.csv', header=0,
                     index_col=0)
```

```
dates = data_d['Date']
prices = data_p['Close']
pseries = np.array(prices)

#-------------------------------------------------------------------
### LSTM training, forecasting and plotting graphs for n times ###
#-------------------------------------------------------------------
# model configs
config = model_configs()
print(config)
# grid search
split_time = 1500
n_repeats = 50
start = timeit.default_timer()
scores = [repeat_evaluate(pseries, dates, cfg, split_time, n_repeats) for cfg
in config]
stop = timeit.default_timer()
print('Time: ', stop - start)  # gives time in seconds
print('done')

#-------------------------------------------------------------------------------
-------------
### Grid search ###
# NOTICE: look and put "#" at the beginning of each function's plotting or
savetxt lines
#-------------------------------------------------------------------------------
-------------
# model configs
cfg_list = model_configs()
print(model_configs())
# grid search
split_time = 500
n_repeats = 1
start = timeit.default_timer()
scores = grid_search(pseries, dates, cfg_list, split_time, n_repeats)
print(scores)
print('done')
stop = timeit.default_timer()
print('Time: ', stop - start)
# list top 10 configs
for cfg, error in scores[:3]:
    print(cfg, error)
print(scores)

#-------------------------------------------------------------
### Statistical analysis for RMSEs ###
#-------------------------------------------------------------
df = pd.read_csv(r' file_path \LSTM(d2250)_RMSE.csv', header=None)
rmses = np.array(df)
#print(rmses)

# calculate statistics
print('Mean: %.3f' % mean(rmses))
print('Variance: %.3f' % var(rmses))
print('Standard Deviation: %.3f' % std(rmses))
print('Max: %.3f' % max(rmses))
print('Min: %.3f' % min(rmses))

# print histograms
```

```
plt.hist(rmses, color='orange', edgecolor='black')
plt.title("LSTM(d2250) RMSE histogrammi", fontsize=22)
plt.xlabel('RMSE', size=15)
plt.ylabel('Kierrokset (lkm)', size=15)
plt.xlim(xmin=0.0, xmax=2.0)
plt.show()
```

Liite 3 ARIMA-koodi

```
#------------------------------------------------------------
### Load libraries ###
#------------------------------------------------------------
from math import sqrt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from pandas import read_csv
from pandas import DataFrame
from pandas.plotting import autocorrelation_plot
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import warnings
from numpy import mean
from numpy import var
from numpy import std
from scipy.stats import kurtosis
from scipy.stats import skew

#------------------------------------------------------------
### Making needed functions ###
# Orginal code from: Jason Brownlee
# From: https://machinelearningmastery.com/grid-search-arima-hyperparameters-
with-python/
#------------------------------------------------------------

# Root Mean Squared Error (RMSE)
def measure_rmse(actual, predicted):
    return sqrt(mean_squared_error(actual, predicted))

# Evaluate an ARIMA model for a given order (p,d,q)
def evaluate_arima_model(X, arima_order):
    # prepare training dataset
    train_size = 500
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    # make predictions
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit(disp=0)
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    # calculate out of sample error
    error = measure_rmse(test, predictions)
    return error
```

```
# Evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p, d, q)
                try:
                    rmse = evaluate_arima_model(dataset, order)
                    if rmse < best_score:
                        best_score, best_cfg = rmse, order
                    print('ARIMA%s RMSE=%.3f' % (order, rmse))
                except:
                    continue
    print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))

#-------------------------------------------------------------
### Reading and preparing data###
#-------------------------------------------------------------
#Reading file
series = read_csv(r' file_path \S&P500_eur.csv', header=0, index_col=0,
squeeze=True)
df = pd.read_csv(r' file_path \S&P500_eur.csv', parse_dates=True,
squeeze=True)
dates = df['Date']
split_time = 500
time_train = dates[1:split_time]
time_valid = dates[split_time:]

#-------------------------------------------------------------
### Evaluate ARIMA parameters###
# Orginal code from: Jason Brownlee
#-------------------------------------------------------------
# Decide combinations of parameters
p_values = [0, 1, 2, 3, 4, 5, 6]
d_values = range(0, 3)
q_values = range(0, 3)

warnings.filterwarnings("ignore")
evaluate_models(series.values, p_values, d_values, q_values)

#-------------------------------------------------------------
### Fit ARIMA and model diagnostics###
# Orginal code from: Jason Brownlee
#-------------------------------------------------------------
#Decide ARIMA parameters
order=(1,1,0)

#Splitting series to train and test sets
train_size = 500
train, test = series[0:train_size], series[train_size:]
history = [x for x in train]

# Fit model
model = ARIMA(train, order=order)
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

```
# Print residuals summary
residuals = DataFrame(model_fit.resid)
print(len(residuals))
print(residuals.describe())
res = np.array(residuals)
print('Mean: %.3f' % mean(res))
print('Variance: %.3f' % var(res))
print('Standard Deviation: %.3f' % std(res))
print('Minimum: %.3f' % min(res))
print('Maximum: %.3f' % max(res))
print('Kurtosis: %.3f' % kurtosis(res))
print('Skewness: %.3f' % skew(res))

print(kurtosis(res))
print(skew(res))

# Plot residuals
fig, ax = plt.subplots(figsize=(8, 5), dpi=80)
ax.plot(time_train, residuals, color='green', label=False)
plt.axhline(linewidth=1, color='red')
plt.title("ARIMA(2,1,0)-mallin residuaalit ", fontsize=18)
plt.xlabel('Aika', size=12)
plt.ylabel('Arvo', size=12)
ax.set_xticks(np.arange(0, len(time_train), 90))
ax.set_xticklabels(time_train[::30], rotation=25, fontdict={'fontsize': 10})
plt.show()
fig.savefig(' file_path /ARIMA_residuaalit.png')
plt.close(fig)

# Plot residuals density function
residuals.plot(kind='kde', color="green", legend=False)
plt.title("Residuaalien tiheysfunktio", fontsize=18)
plt.xlabel('Arvo', size=12)
plt.ylabel('Tiheys', size=12)
plt.show()
fig.savefig(' file_path /ARIMA_tiheys.png')
plt.close(fig)

# Plot residuals histogram
residuals.plot(kind='hist', color="green", legend=False)
plt.title("Residuaalien histogrammi", fontsize=18)
plt.xlabel('Arvo', size=12)
plt.ylabel('Havainnot', size=12)
plt.show()

# Plot residuals ACF
plot_acf(residuals, color="green", lags=50)
plt.title("Autokorrelaatiofunktio(ACF)", fontsize=18)
plt.xlabel('Viive', size=12)
plt.show()
fig.savefig(' file_path /ARIMA_ACF.png')
plt.close(fig)

# Plot residuals PACF
plot_pacf(residuals, color="green", lags=50)
plt.title("Osittaisautokorrelaatiofunktio(PACF)", fontsize=18)
plt.xlabel('Viive', size=12)
plt.show()
fig.savefig(' file_path /ARIMA_PACF.png')
```

```
plt.close(fig)

# Plot residuals ACF
autocorrelation_plot(residuals, color="green")
plt.title("Autokorrelaatiofunktio", fontsize=18)
plt.xlabel('Viive', size=12)
plt.ylabel('ACF', size=12)
plt.show()

#------------------------------------------------------------
### ARIMA fit & forecast ###
# Orginal code from: Jason Brownlee
#------------------------------------------------------------
#Decide ARIMA parameters
order=(2,1,0)

#Splitting series to train and test sets
train_size = 500
train, test = series[0:train_size], series[train_size:]
history = [x for x in train]
predictions = list()

for t in range(len(test)):
    model = ARIMA(history, order=order)
    model_fit = model.fit(disp=0)
    yhat = model_fit.forecast()[0]

    predictions.append(yhat)
    history.append(test[t])
error = measure_rmse(test, predictions)
print('ARIMA%s RMSE=%.3f' % (order, error))

#Plot forecast
fig, ax = plt.subplots(figsize=(16, 9), dpi=80)
ax.plot(time_valid, test, color='black', label="S&P500 (euro)")
ax.plot(time_valid, predictions, color='green', label="ARIMA(2,1,0)-malli")
plt.legend(loc="upper left")
plt.title("ARIMA(2,1,0)-mallin yhden päivän ennusteet", fontsize=22)
ax.set_xlabel('Aika', size=15)
ax.set_ylabel('Hinnat', size=15)
ax.set_xticks(np.arange(0, len(time_valid), 30))
ax.set_xticklabels(time_valid[::30], rotation=25, fontdict={'fontsize': 10})
ax.spines["top"].set_alpha(.3)
ax.spines["bottom"].set_alpha(1.)
ax.spines["right"].set_alpha(.3)
ax.spines["left"].set_alpha(1.0)
plt.grid(axis='both', alpha=.3)
plt.show()
fig.savefig(' file_path /ARIMA_ennusteet.png')
plt.close(fig)
```

Liite 4 Ristihakualgoritmissa käytetyt kombinaatiot sekä niiden ajallinen kesto

GPU			Kombinaatiot = [window_size, batch_size, shuffle_buffer_size, n_nodes, n_epochs]			
Menetelmä	**Verkko**	**Kerrokset**	window_size	[1,2]	n_nodes	[15,25,50,75]
Grid search	LSTM	1	batch_size	[8,10,25]	n_epochs	[250]
Havainnot	**Jako (train/test)**		shuffle_buffer_size	[1,10,25,100,1500]	lr	1,00E-03
2 250	1 500/750			learning_rate	momentum	0,9
Diff	**Skaalaus**	**Komb.lkm**	**Kerrat per komb.**	**Parhaat kombinaatiot**	**RMSE**	**Aika**
Ei	[0,1]			1. [1,10,1,75,250]	25,??	
		120	5	2. [2,8,100,75,250]	25,??	**24h 7,5min**
				3. [2,25,1,75,250	25,??	
CPU			Kombinaatiot = [window_size, batch_size, shuffle_buffer_size, n_nodes, n_epochs]			
Menetelmä	**Verkko**	**Kerrokset**	window_size	[1,2]	n_nodes	[15,25,50,75]
Grid search	LSTM	1	batch_size	[8,10,25]	n_epochs	[250]
Havainnot	**Jako (train/test)**		shuffle_buffer_size	[1,10,25,100,1500]	lr	1,00E-03
2 250	1 500/750				momentum	0,9
Diff	**Skaalaus**	**Komb.lkm**	**Kerrat per komb.**	**Parhaat kombinaatiot**	**RMSE**	**Aika**
Ei	[0,1]			1. [1,10,100,75,250]	24,599	
		120	5	2. [1,8,1500,75,250]	25,03	**18h 7,5min**
				3. [1,8,1,75,250]	25,35	

Grid search kompojen kokeilu

MLP CPU 50 krt

Aineisto	Aika	Hyperparametrit							Tulokset(RMSE)					Histogrammin
		window_size	batch_size	shuffle_buffer_size	n_nodes	n_epochs	lr	momentum	Keskiarvo	Varianssi	Keskihajonta	Maksimi	Minimi	Skaalaus
d750	19 min	1	25	1	75	170	1e-4	0,65	5,413	3,574	1,8891	9,821	2,679	[0,10]
750	67 min	2	1	165	75	250	1e-3	0,9	28,791	10,609	3,257	42,842	25,483	[20,45]
d2250	100 min	1	1	1	75	100	1e-5	0,1	5,289	4,425	2,103	12,13	1,432	[0,13]
2250	61 min	2	32	1500	75	250	1e-3	0,1	100,956	1935,485	43,994	208,04	28,412	[25,210]

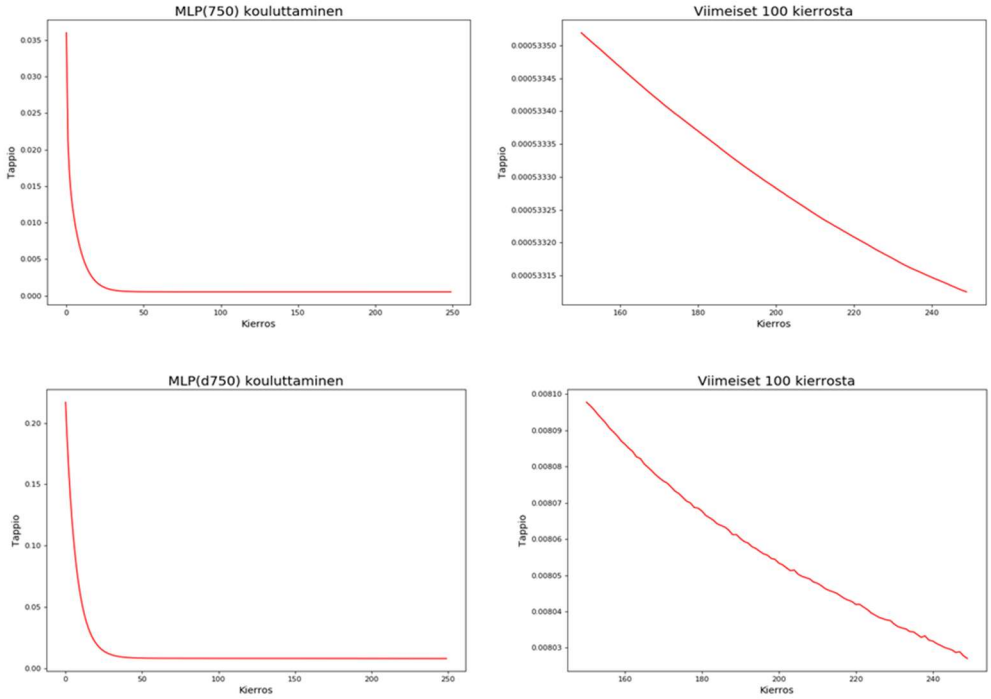
RNN CPU 50 krt

Aineisto	Aika	Hyperparametrit							Tulokset(RMSE)					Histogrammin
		window_size	batch_size	shuffle_buffer_size	n_nodes	n_epochs	lr	momentum	Keskiarvo	Varianssi	Keskihajonta	Maksimi	Minimi	Skaalaus
d750	19 min	1	25	250	250	150	1e-4	0,45	6,128	6,337	2,517	12,15	1,192	[0,13]
750	29 min	1	6	1	250	250	1e-3	0,9	25,417	0	0,009	25,437	25,396	[25,3, 25,5]
d2250	133 min	1	1	1	250	100	1e-5	0,1	4,503	3,187	1,785	7,841	0,706	[0,8]
2250	168 min	1	1	1000	250	140	1e-4	0,1	21,877	0,013	0,116	22,202	21,614	[21,5, 22,4]

LSTM CPU 50 krt

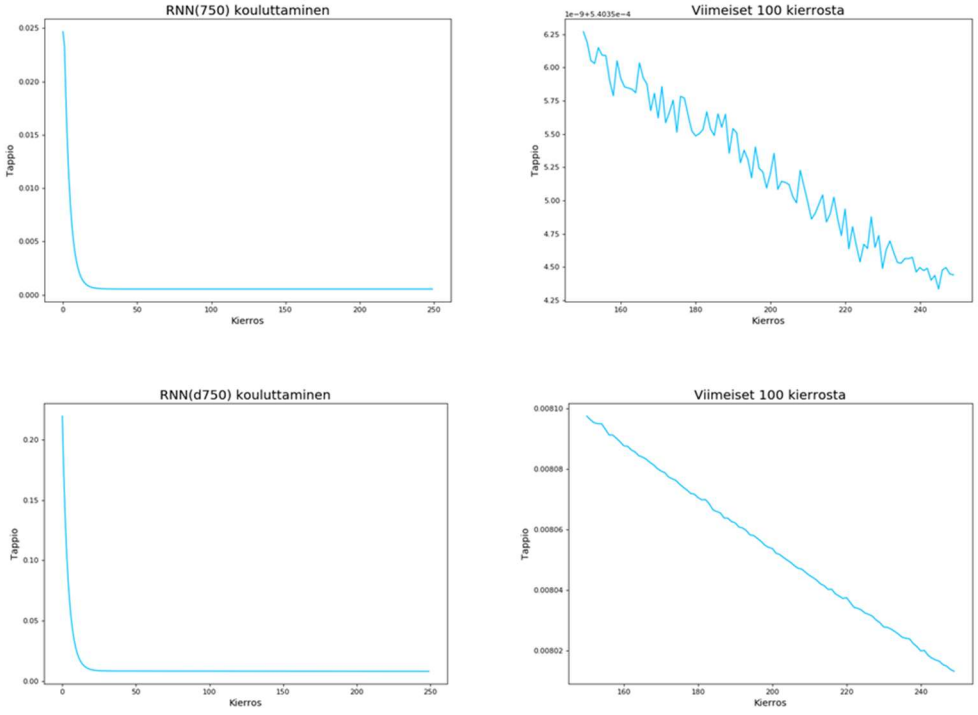
Aineisto	Aika	Hyperparametrit							Tulokset(RMSE)					Histogrammin
		window_size	batch_size	shuffle_buffer_size	n_nodes	n_epochs	lr	momentum	Keskiarvo	Varianssi	Keskihajonta	Maksimi	Minimi	Skaalaus
d750	30 min	1	6	250	75	180	1e-4	0,5	1,155	0,173	0,416	2,376	0,723	[0,3]
750	59 min	1	1	165	250	80	1e-3	0,9	26,141	0,416	0,645	28,257	25,373	[25, 28,5]
d2250	240 min	2	1	1	250	100	1e-5	0,4	1,23	0,108	0,329	1,992	0,661	[0,2]
2250	70 min	1	6	1500	250	90	1e-3	0,9	22,049	0,174	0,417	23,171	21,353	[21,23]

Liite 5 Neuroverkkojen kuvaajat (oppiminen)



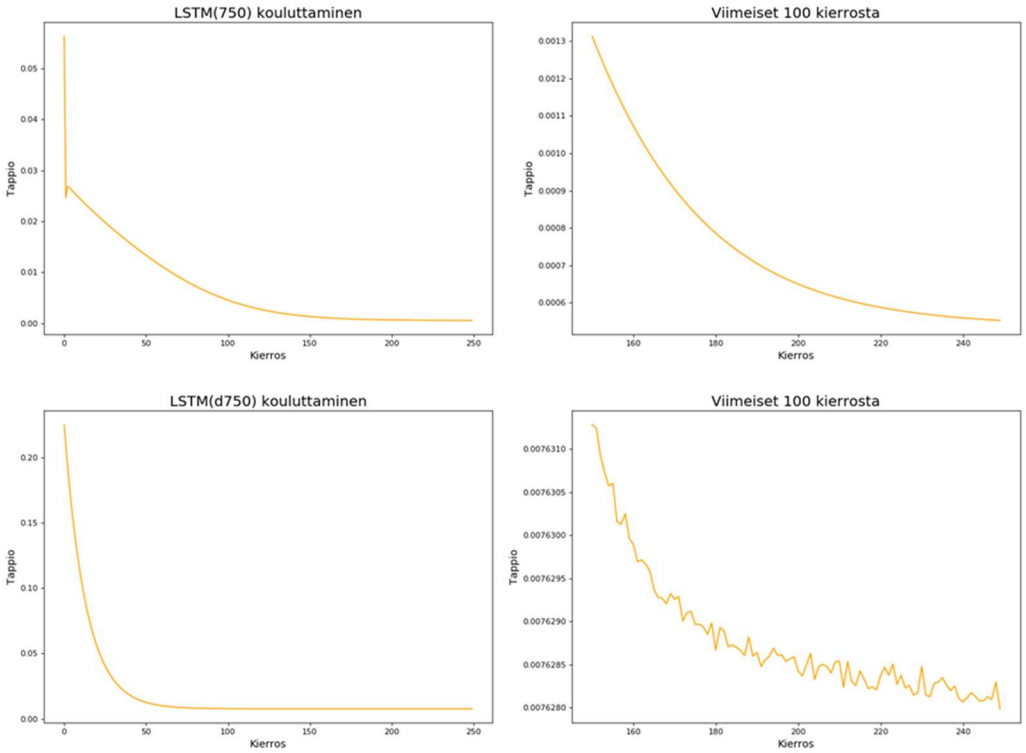
MLP(750): 29/50 kerta, RMSE ≈ 29,61

MLP(d750): 33/50 kerta, RMSE ≈ 4,84



RNN(750): 18/50 kerta, RMSE ≈ 24,42

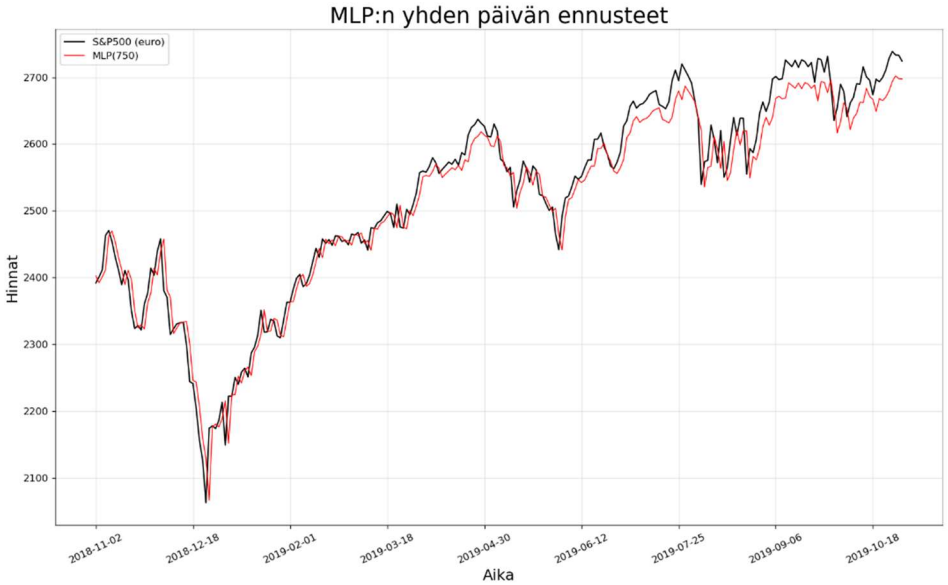
RNN(d750): 18/50, RMSE ≈ 4,36



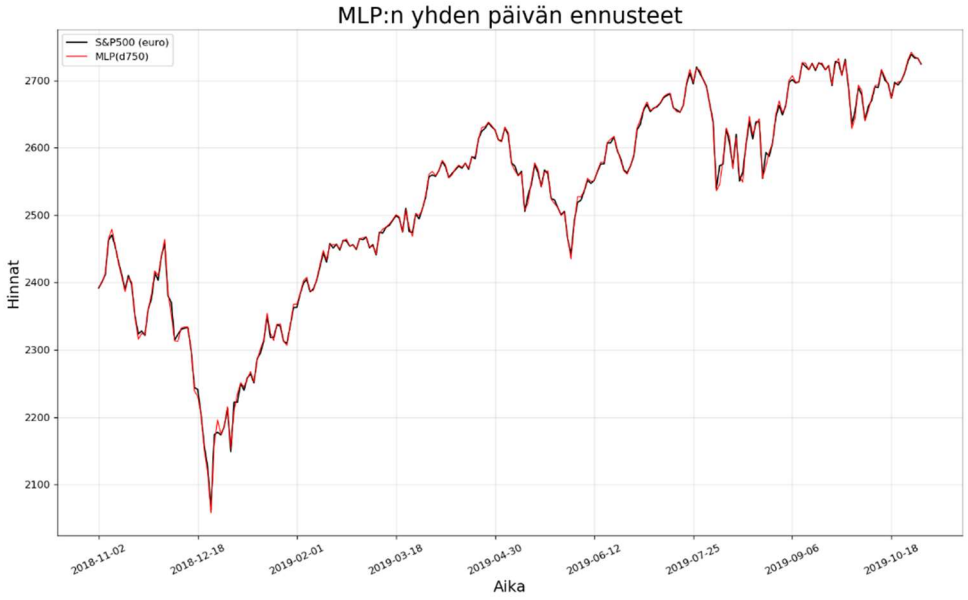
LSTM(750): 6/50 kerta, RMSE ≈ 27,43

LSTM(d750): 5/50 kerta, RMSE ≈ 1,164

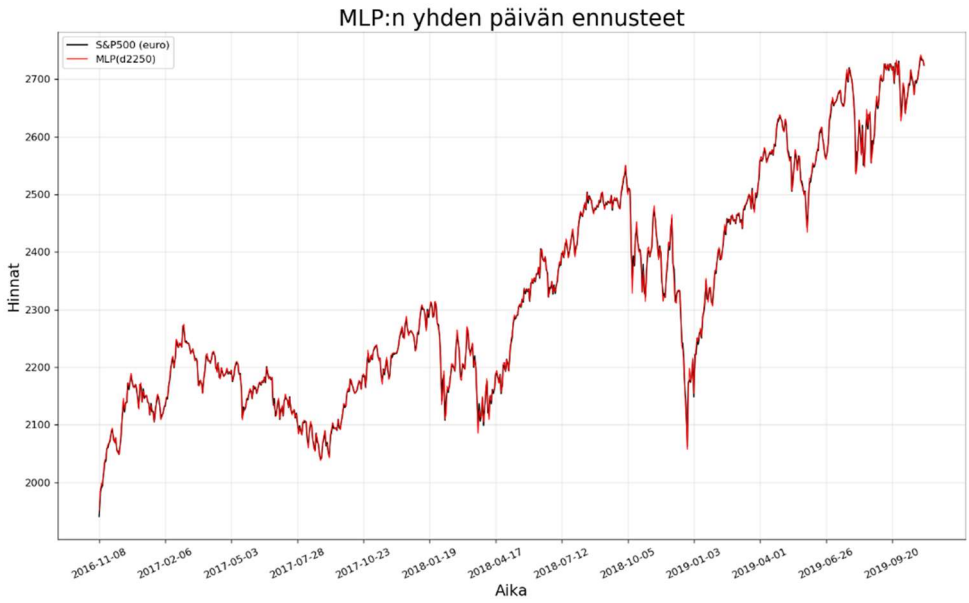
Liite 6 Neuroverkkojen kuvaajat (ennustaminen)



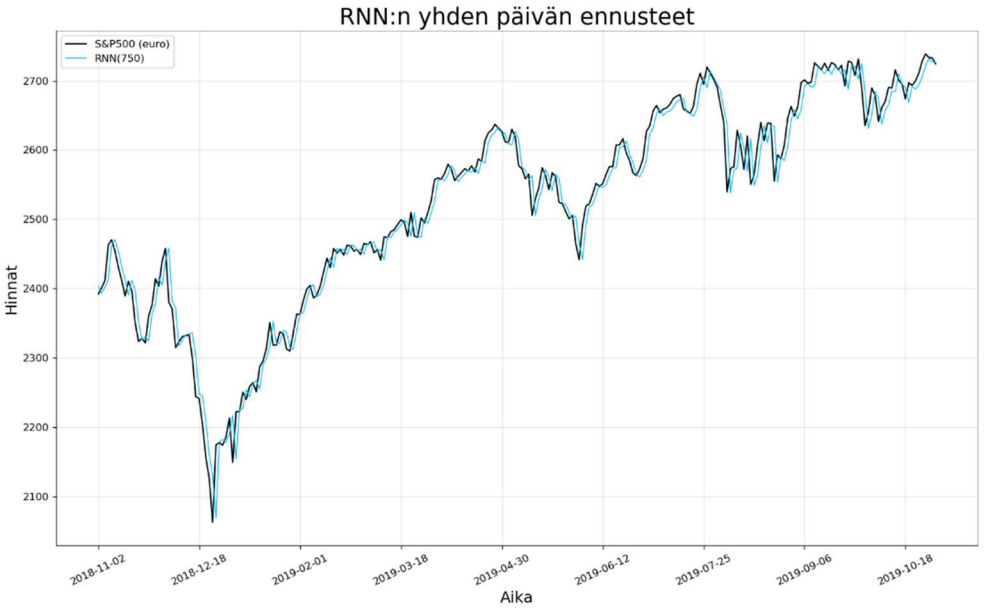
MLP(750): 29/50 kerta, RMSE ≈ 29,61



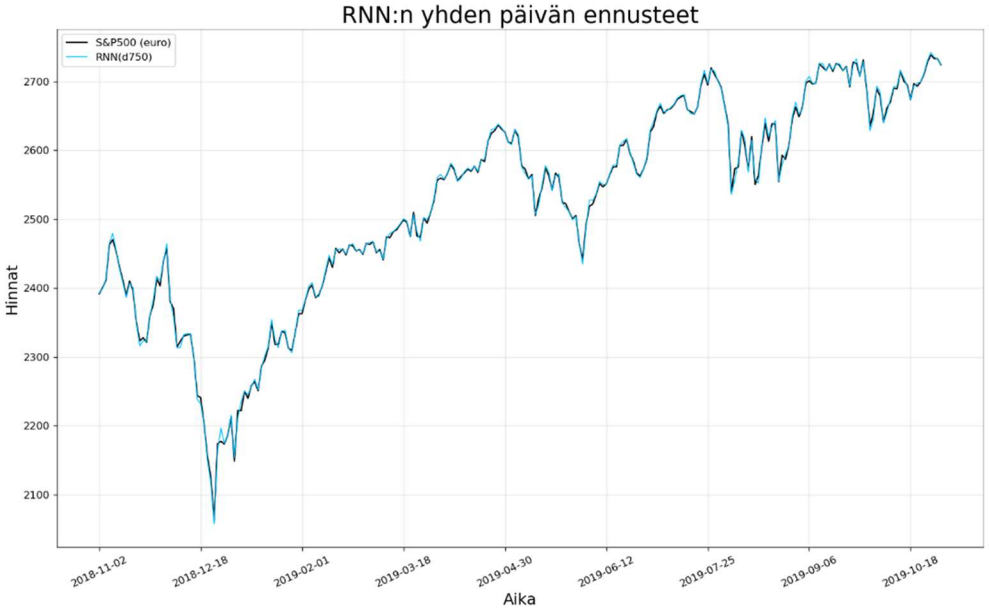
MLP(d750): 33/50 kerta, RMSE ≈ 4,84



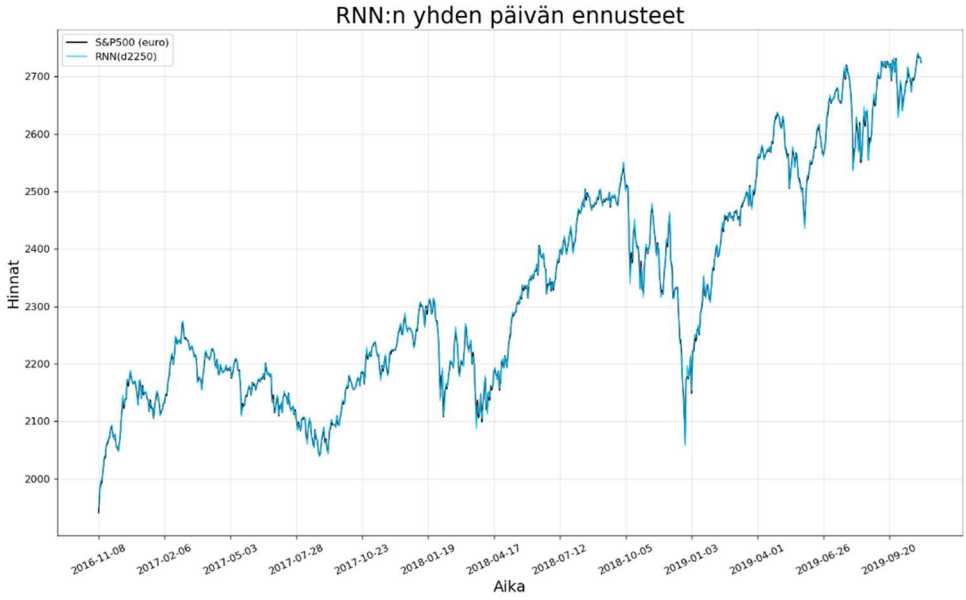
MLP(d2250): 16/50 kerta, RMSE ≈ 4,07



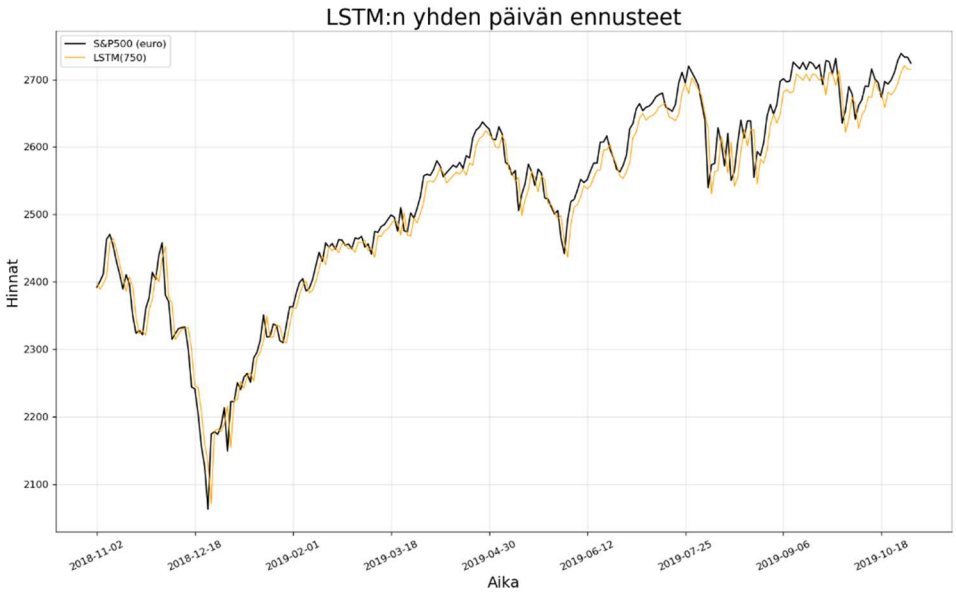
RNN(750): 18/50 kerta, RMSE ≈ 24,42



RNN(d750): 18/50, RMSE ≈ 4,36



RNN(d2250): 33/50 kerta, RMSE ≈ 21,576

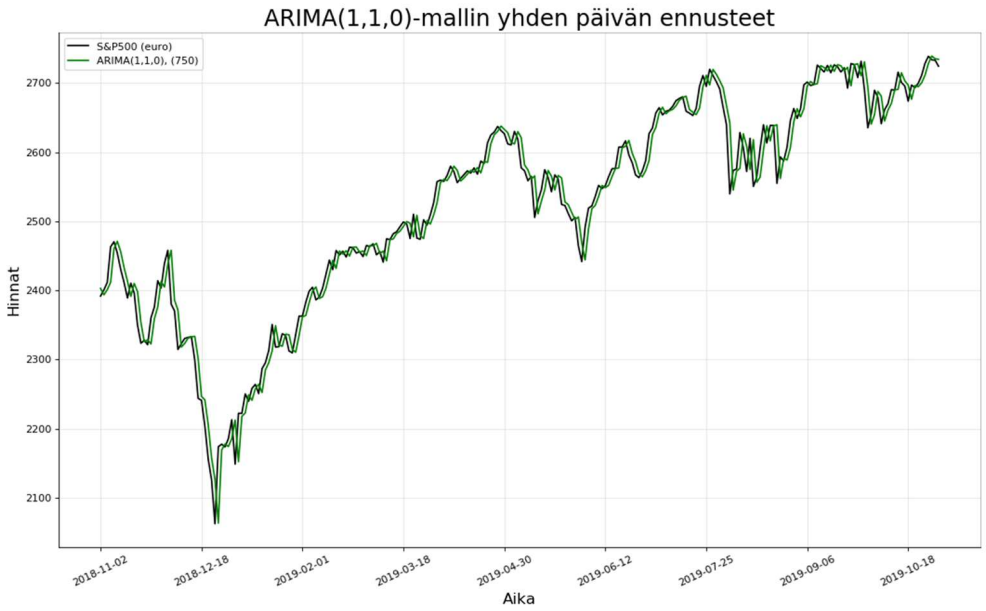


LSTM(750): 6/50 kerta, RMSE ≈ 27,43

LSTM(d750): 5/50 kerta, RMSE ≈ 1,164

LSTM(2250): 12/50 kerta, RMSE ≈ 21,849

Liite 7 ARIMA-mallin kuvaaja (ennustaminen)



ARIMA(1,1,0), (750), RMSE ≈ 25,309

Liite 8 ARIMA-mallien estimoinnin tulokset

ARIMA Model Results

Dep. Variable:	D.Close	No. Observations:	499
Model:	ARIMA(1, 1, 0)	Log Likelihood	-2162.820
Method:	css-mle	S.D. of innovations	18.456
Date:	Fri, 21 Feb 2020	AIC	4331.641
Time:	09:19:51	BIC	4344.279
Sample:	1	HQIC	4336.600

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.9181	0.767	1.197	0.232	-0.585	2.421
ar.L1.D.Close	-0.0776	0.045	-1.731	0.084	-0.165	0.010

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-12.8927	+0.0000j	12.8927	0.5000

ARIMA Model Results

Dep. Variable:	D.Close	No. Observations:	499
Model:	ARIMA(0, 1, 1)	Log Likelihood	-2162.537
Method:	css-mle	S.D. of innovations	18.445
Date:	Fri, 21 Feb 2020	AIC	4331.073
Time:	09:36:51	BIC	4343.711
Sample:	1	HQIC	4336.033

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.9156	0.750	1.220	0.223	-0.555	2.386
ma.L1.D.Close	-0.0915	0.048	-1.913	0.056	-0.185	0.002

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	10.9340	+0.0000j	10.9340	0.0000

ARIMA Model Results

Dep. Variable:	D.Close	No. Observations:	499
Model:	ARIMA(2, 1, 0)	Log Likelihood	-2160.790
Method:	css-mle	S.D. of innovations	18.380
Date:	Fri, 21 Feb 2020	AIC	4329.580
Time:	09:48:09	BIC	4346.431
Sample:	1	HQIC	4336.193

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.9052	0.700	1.292	0.197	-0.467	2.278
ar.L1.D.Close	-0.0848	0.045	-1.894	0.059	-0.173	0.003
ar.L2.D.Close	-0.0906	0.045	-2.019	0.044	-0.179	-0.003

Roots

	Real	Imaginary	Modulus	Frequency

AR.1	-0.4682	-3.2891j	3.3222	-0.2725
AR.2	-0.4682	+3.2891j	3.3222	0.2725

ARIMA Model Results

Dep. Variable:	D.Close	No. Observations:	1499
Model:	ARIMA(1, 1, 0)	Log Likelihood	-6167.413
Method:	css-mle	S.D. of innovations	14.812
Date:	Wed, 18 Mar 2020	AIC	12340.826
Time:	18:30:28	BIC	12356.764
Sample:	1	HQIC	12346.763

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.7006	0.394	1.779	0.075	-0.071	1.472
ar.L1.D.Close	0.0284	0.026	1.096	0.273	-0.022	0.079

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	35.1506	+0.0000j	35.1506	0.0000

ARIMA Model Results

Dep. Variable:	D.Close	No. Observations:	1499
Model:	ARIMA(0, 1, 1)	Log Likelihood	-6167.380
Method:	css-mle	S.D. of innovations	14.811
Date:	Wed, 18 Mar 2020	AIC	12340.760
Time:	18:45:54	BIC	12356.698
Sample:	1	HQIC	12346.698

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.7006	0.394	1.778	0.075	-0.072	1.473
ma.L1.D.Close	0.0300	0.027	1.128	0.259	-0.022	0.082

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	-33.3677	+0.0000j	33.3677	0.5000

ARIMA Model Results

Dep. Variable:	D.Close	No. Observations:	1499
Model:	ARIMA(0, 1, 2)	Log Likelihood	-6166.913
Method:	css-mle	S.D. of innovations	14.807
Date:	Wed, 18 Mar 2020	AIC	12341.827
Time:	18:51:17	BIC	12363.077
Sample:	1	HQIC	12349.743

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.6994	0.383	1.827	0.068	-0.051	1.450
ma.L1.D.Close	0.0268	0.026	1.026	0.305	-0.024	0.078
ma.L2.D.Close	-0.0260	0.027	-0.966	0.334	-0.079	0.027

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	-5.7125	+0.0000j	5.7125	0.5000
MA.2	6.7455	+0.0000j	6.7455	0.0000