Rony Franca Leppänen

Anomalioiden havaitseminen langattomissa sensoriverkoissa syväoppimisen avulla

Tietotekniikan
pro gradu -tutkielma
4. joulukuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Kokkolan yliopistokeskus Chydenius

Tekijä: Rony Franca Leppänen
Yhteystiedot: leppanen.rony@gmail.com
Puhelinnumero: +5583991752220
Ohjaaja: Ismo Hakala ja Risto Honkanen
Työn nimi: Anomalioiden havaitseminen langattomissa sensoriverkoissa syväoppimisen avulla
Title in English: Anomaly detection in wireless sensor networks with deep learning
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 103+3
Tiivistelmä: Globaalista IP-verkkoliikenteestä yhä suuremmasta osuudesta vastuussa olevat uuden sukupolven langattomat verkot ja mobiili- sekä IoT-sovellukset ovat jalkautumassa aina kriittisen infrastruktuurin järjestelmiin asti. Fyysisen ja digitaalisen maailman rajapinnassa osana IoT-sovelluksia toimivat langattomat sensoriverkot ovat alttiita laajalle kirjolle erilaisia tietoturvauhkia niiden avoimen luonteen, IoT-sovellusten teknologisen kypsymättömyyden ja alati kehittyvän kyberrikollisuuden vuoksi. Langattomien sensoriverkkojen suojaaminen kyberhyökkäyksiltä ja muulta niiden luotettavaa toimintakykyä uhkaavalta ja vahingoittavalta toiminnalta on tärkeä tutkimusaihe. Tässä työssä tutkittiin hiljattain julkaistun esineiden internetin sovellusympäristöä jäljittelevän Bot-IoT -datajoukon avulla verkkohyökkäyksien tunnistamista anomalioiden havaitsemisen keinoin käyttämällä moderneja syväoppimismenetelmiä. Työssä implementoidaan ja vertaillaan neljää autoenkooderiarkkitehtuuriin perustuvaa yksinkertaista ja laskennallisesti kevyttä syväoppimismallia. Suorituskykyisin toistuvaan neuroverkkoon perustuva LSTM-autoenkooderi kykeni tunnistamaan yli 3,6 miljoonaa hyökkäystä jättäen vain 101 hyökkäystä tunnistamatta. Työssä tehdyn kaltaista tutkimusta Bot-IoT -datajoukkoon ei ole tiedeyhteisössä aiemmin toteutettu eikä vastaavia tuloksia ole ennen saatu. Lisäksi työssä annetaan kattava teoreettinen tausta tunnetuimmista syväoppimismenetelmistä ja niiden soveltamisesta anomalioiden havaitsemiseen.
Avainsanat: tietoturvallisuus, anomalian havaitseminen, langaton sensoriverkko, syväoppiminen, autoenkooderi
Abstract: The next-generation wireless and mobile networking as well as IoT applications accounting for an ever-increasing share of the global IP network traffic are being widely deployed reaching critical infrastructures. Acting as an interface between the physical and the digital world in IoT applications, wireless sensor networks are exposed to a wide range of information security threats due to their open nature

of communications, the technological immaturity of IoT solutions and the accelerating growth of cybercrime. Protecting wireless sensor networks from cyberattacks and other factors that may impair the continuity of their secure and reliable operations is an important area of research. In this thesis, the ability of detecting network attacks with methods based on deep learning using principles from anomaly detection was investigated by a recently published dataset called Bot-IoT that incorporates flow-based network traffic from an IoT environment. Four different lightweight deep learning based autoencoders were implemented for evaluation and comparison purposes. The results demonstrated the superiority of the recurrent LSTM-autoencoder model by detecting over 3.6 million attacks while leaving only 101 attacks undetected. The empirical study conducted in this thesis with the Bot-IoT -dataset is the first of its kind in the scientific community and similar results have not yet been published. In addition, a comprehensive theoretical background of the most common deep learning methods and their applicability to anomaly detection is given.
Keywords: information security, anomaly detection, wireless sensor network, deep learning, autoencoder

Sanasto

6LoWPAN	IPv6 over Low Power Wireless Personal Area Networks
Adam	Adaptive Moment Estimation
AE	Autoencoder
AI	Artificial Intelligence
APT	Advanced Persistent Threat
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
DDoS	Distributed DoS
DoS	Denial of Service
ELU	Exponential Linear Unit
EMA	Exponential Moving Average
ERM	Empirical Risk Minimization
FFNN	Feed-Forward Neural Network
GMM	Gaussian Mixture Model
GRU	Gated Recurrent Unit
HTTP	HyperText Transfer Protocol
IDS	Intrusion Detection System
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoT	Internet of Things
IP	Internet Protocol
LOF	Local Outlier Factor
LSTM	Long Short-Term Memory
M2M	Machine-to-Machine
MITM	Man-in-the-Middle
MLP	Multi-Layer Perceptron
MQTT	Message Queuing Telemetry Transport
NLP	Natural Language Processing
OS	Operating System

OSI	Open Systems Interconnection Reference Model
PCA	Principal Component Analysis
PReLU	Parametric ReLU
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC AUC	Area Under the Receiving Operating Characteristic Curve
Seq2Seq	Sequence-to-Sequence
SOM	Self-Organizing Map
SVM	Support Vector Machine
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network

Sisältö

1 Johdanto

Eräiden arvioiden mukaan elokuussa 2019 oli maailmanlaajuisesti yli 26 miljardia esineiden internetin sovelluksissa (*eng. internet of things, IoT*) toimivaa laitetta ja lukumäärän odotetaan kasvavan 75 miljardiin vuoteen 2025 mennessä [10]. Vuoteen 2022 mennessä globaalin vuosittaisen IP-verkkoliikenteen määrän ennustetaan kasvavan 4.8 tsettatavuun (10^{15} MB), josta älypuhelinten luoman verkkoliikenteen osuus tulee olemaan yli puolet. Lisäksi joidenkin ennustusten mukaan vuoden 2022 aikana kaikista Internetiin yhdistetyistä laitteista yli puolet on IoT-laitteita ja 71 prosenttia globaalista IP-verkkoliikenteestä on peräisin mobiililaitteilta ja langattomista verkoista. Globaalista IP-verkkoliikenteestä 22 prosenttia katsotaan myös tulevan mobiili- eli solukkoverkoista (*eng. cellular network*), josta 12 prosenttia edelleen uuden sukupolven 5G-solukkoverkoista [31]. Yhdessä IoT:n kanssa, 5G:n odotetaan jalkautuvan kuluttajille suunnatuista palveluista aina kriittisen infrastruktuurin järjestelmiin asti [50]. Tällä vuosikymmenellä koettu ja tulevaisuudessakin jatkuva IoT-alan kasvu on toisaalta aiheuttanut suurta huolta IoT:n tietoturvallisuuden tasosta, kun IoT-järjestelmiin kohdistuneet tai IoT-laitteita hyödyntäneet verkkohyökkäykset ovat yleistyneet räjähdysmäisesti ja niiden seuraukset muuttuneet yhä vakavimmiksi [44, 43, 13].

Erilaisten järjestelmien etämonitorointi sensorein ja sensoreilta saatavan datavirran analysointi sekä hyödyntäminen päätöksenteossa on yksi IoT:n houkuttelevimmista ja nopeimmin kasvavista käyttötapauksista eri liiketoiminta-alueilla [35]. Langattomat sensoriverkot (*eng. wireless sensor network, WSN*) toimivat olennaisena osana IoT-sovelluksia fyysisen ja digitaalisen maailman rajapinnassa ja siten myös kriittisenä osana IoT-sovellusten tietoturvaa. IoT-sovellusten teknologinen kypsymättömyys ja langattomien sensoriverkkojen langaton luonne tekevät niistä alttiimpia laajemmalle kirjolle erilaisia hyökkäyksiä verrattuna langallisiin verkkoihin. Kyberhyökkäyksiltä ja muulta niiden luotettavaa toimintakykyä uhkaavalta ja vahingoittavalta toiminnalta suojaaminen on eräs lupaavimmista ja ajankohtaisimmista tutkimuksen kohteista langattomissa sensoriverkoissa [85].

Uuden sukupolven langattomien verkkojen ja mobiili- sekä IoT-sovellusten tuoma verkkojen monimuotoisuus, laitteiden heterogeenisyys ja valtava datan määrä

tekee verkkojen ja sen eri osien monitoroinnista ja hallinnasta lähes mahdotonta. Täten erilaisten älykkäiden menetelmien sisällyttäminen osaksi näitä tulevaisuuden verkkoja on herättänut suurta kiinnostusta tiedeyhteisössä. Tämä trendi on tuonut mukanaan koneoppimisen (*eng. machine learning*) soveltamisen verkottuneiden järjestelmien kehittämiseen. Koneoppimismenetelmillä voidaan muun muassa automatisoidusti analysoida ja irrottaa hyödyllistä tietoa kompleksisesta verkkodataliikenteestä ilman asiantuntija-apua pienin esioletuksin. Nykyään koneoppimista toteutetaan yhä kasvavissa määrin ennennäkemätöntä menestystä erilaisissa tekoälysovelluksissa viime vuosikymmenen aikana keränneellä syväoppimisella (*eng. deep learning*), jolla on monia potentiaalisia sovelluskohteita langattomien verkkojen ja mobiilisovellusten lisäksi langattomissa sensoriverkoissa. Syväoppimismenetelmien potentiaalisiin sovellusalueisiin langattomissa sensoriverkoissa lukeutuu niin paikannus, synkronointi, energiansäästö ja reititys kuin laiterikkojen ja verkkohyökkäysten havaitseminen [151, 85].

Tämän työn tavoitteena on tutkia modernien syväoppimismenetelmien soveltuvuutta verkkohyökkäysten havaitsemiseen nykyaikaisessa esineiden internetin sovellusympäristössä hiljattain julkaistun Bot-IoT -datajoukon avulla. Verkkohyökkäyksien tunnistaminen tehdään analysoimalla Bot-IoT -datajoukon sisältämää verkkoliikennettä kuvailevaa metatietoa anomalioiden havaitsemisen keinoin. Työn empiirisessä osuudessa implementoidaan ja vertaillaan neljää autoenkooderiarkkitehtuuria noudattavaa puoliohjattua syväoppimismenetelmää. Työn ensisijaisiin tuloksiin lukeutuu yksinkertaisen ja laskennallisesti kevyen syväoppimismallin kehittäminen, joka tunnistaa yli 3,6 miljoonaa hyökkäystä Bot-IoT -datajoukosta jättäen vain 101 hyökkäystä tunnistamatta. Lisäksi työssä annetaan kattava teoreettinen tausta käytetyistä syväoppimismenetelmistä ja niiden soveltamisesta anomalioiden havaitsemiseen.

Luvussa 2 määritellään langattomien sensoriverkkojen käsite ja pohditaan niiden ominaisuuksia. Lisäksi pureudutaan langattomia sensoriverkkoja koskeviin uhkiin ja haavoittuvuuksiin tietoturvallisuuden kannalta sekä käsitellään anomalioiden havaitsemista. Kolmannessa luvussa käydään läpi työssä käytettyjen syväoppimismenetelmien teoria yleisellä tasolla, jonka jälkeen luvussa 4 syvennytään menetelmien soveltamiseen anomalioiden havaitsemisessa. Luku 5 esittelee työssä käytetyn datajoukon ja implementoituihin menetelmiin liittyvät yksityiskohdat. Luvussa 6 analysoidaan työn tulokset, pohditaan jatkokehitysideoita ja tekoälytutkimuksen tilaa nyt ja tulevaisuudessa. Luku 7 päättää tutkielman yhteenvetoon työn sisällöstä.

2 Tietoturvallisuus ja anomalioiden havaitseminen langattomissa sensoriverkoissa

Keräämällä tietoa havainnoimalla tai aistimalla (*eng. sensing*) fysikaalisista esineistä, ilmiöistä, prosesseista ja ympäristöistä voidaan suorittaa erilaisia valvontaan ja hallintaan liittyviä tehtäviä. Kerättyjen tietojen perusteella voidaan muun muassa päätellä, onko valvottavassa kohteessa tapahtunut lämpötilasta, paineesta, värinästä, virtauksesta tai muun yhden tai useamman fysikaalisen ominaisuuden muutoksesta aiheutuvia tilanmuutoksia. Esinettä tai laitetta, joka suorittaa itse tietojen keräämistä, kutsutaan mittalaitteeksi, anturiksi tai sensoriksi (*eng. sensor*). Jos tietoa kerätessään sensorin ei tarvitse olla kosketuksissa valvottavan kohteen kanssa, sitä kutsutaan etäsensoriksi (*eng. remote sensor*). Teknisessä mielessä sensori muuntaa fysikaalisen energian sähköiseen muotoon niin, että sitä pystytään käsittelemään esimerkiksi laskennallisesti tai hallintalaitteen toimesta. Toisin sanoen sensori on laite, joka kääntää todellisesta ympäristöstä suureita ja tapahtumia mitattaviksi ja analysoitaviksi signaaleiksi [37].

Yksi sensori harvoin kuitenkaan riittää tietojen keräämiseen valvottavasta kohteesta vaan monissa käytännön sovelluksissa satoja tai tuhansia sensorinoodeja (*eng. sensor node*) valjastetaan usein syrjäisiin ja luoksepääsemättömiin paikkoihin. Nämä sensorinoodit ovat usein langattomia ja sensoreiden lisäksi niissä on tyypillisesti komponentteja, jotka antavat sille kyvyn tehdä laskentaa, varastoida tietoa ja kommunikoida muiden laitteiden kanssa. Kun monet sensorinoodit keräävät tietoja yhteistyössä valvottavasta kohteesta, ne muodostavat langattoman sensoriverkon, missä sensorinoodit eivät kommunikoi langattomasti pelkästään toistensa vaan myös tukiaseman kanssa. Tällä tavoin sensorinoodit pystyvät jakamaan kerätyt tiedot edelleen etälaitteille, jotka analysoivat, visualisoivat ja varastoivat dataa [37].

Kuten mainittua, langattoman sensoriverkon sensorinoodeilla voi olla erilaisia ominaisuuksia. Yksinkertaisimmat ja minimalistisimmat sensorinoodit saattavat havainnoida ja kerätä tietoa yksittäisestä fysikaalisesta suureesta, kun taas monia signaaleja, kuten elektromagneettisia, kemiallisia, optisia ja akustisia, samanaikaisesti keräävät laitteet ovat monimutkaisempia ja vaativat enemmän resursseja. Sensorinoodien viestintäviiveet ja tiedonsiirtonopeudet saattavat tapauskohtaisesti myös

vaihdella sekä tiedonlähetys voi tapahtua käyttämällä eri aallonpituuksia ja taajuuksia, kuten radio-, infrapuna-, mikro- tai ultraääniaaltoja. Enemmän resursseja omaavilla laitteilla voi olla edelleen datan prosessointia, varastointia ja syntetisointia vaativia lisätehtäviä verkossa, kuten reitittimenä tai tiedonkeruupisteenä toiminen [37].

Pienien, edullisten ja älykkäiden sensorien saatavuus on tehnyt langattomista sensoriverkoista kustannustehokkaampia ja helpommin käyttöön otettavampia, jolloin niistä on tullut kasvavissa määrin houkutteleva ratkaisu lähes reaaliaikaisuutta vaativiin käytännön sovelluksiin. Ne ovat myöskin kiinteä osa esineiden internetiä toimiessaan rajapintana fysikaalisen ja digitaalisen maailman välillä tehden niistä erään lupaavimmista tulevaisuuden teknologioista [117]. Lähes reaaliaikaisia langattomien sensoriverkkojen käytännön sovelluksia löytyy ympäristöjen, putkiliikenteen, teollisten prosessien ja rakenteiden valvonnasta sekä tarkkuusviljelystä, logistiikasta, maanalainen louhinnasta, kotiautomaatiosta, terveydenhuollosta, liikenteenvalvonnasta ja monesta muusta [37, 144].

Vaikka langattomilla sensoriverkoilla on paljon samoja piirteitä muiden hajautettujen järjestelmien (*eng. distributed system*) kanssa, niillä on myös niille ominaisia langattoman sensoriverkon suunnitteluun liittyviä haasteita ja rajoitteita. Usein näiden rajoitteiden kanssa toimiminen johtaa sellaisten protokollien ja algoritmien kehittämiseen, joiden vastineet muissa hajautetuissa järjestelmissä ovat erilaisia. Taulukkoon 2.1 on kerätty perinteisten verkkojen ja langattomien sensoriverkkojen ominaisuuksia ja niiden eroja [37].

Taulukko 2.1: Perinteisten verkkojen ja langattomien sensoriverkkojen vertailua.

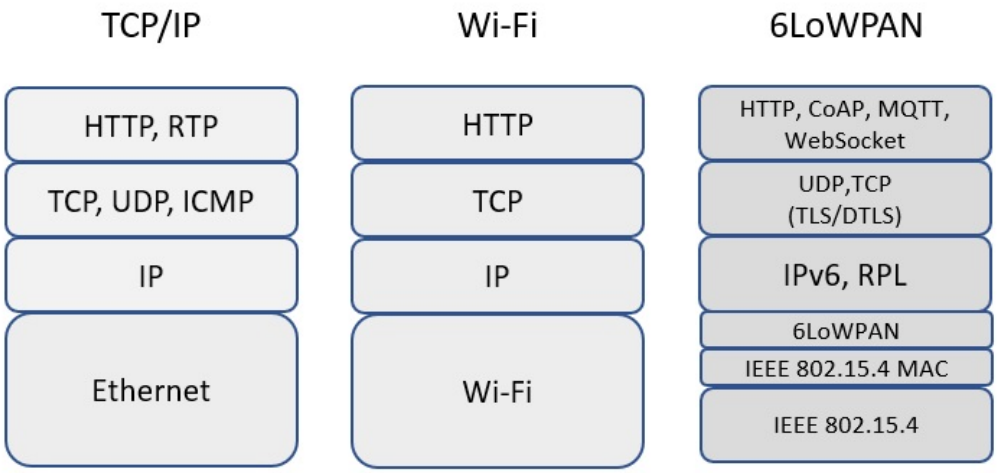
Perinteiset verkot	Langattomat sensoriverkot
Yleiskäyttöinen monia sovelluksia palveleva suunnittelu	Tapauskohtainen yhtä sovellusta palveleva suunnittelu
Keskeisiä haasteita verkon suorituskyky ja viive	Energiankulutus keskeisin haaste kaikissa noodeissa ja verkon osissa
Verkot suunnitellaan ja rakennetaan suunnitelmallisesti	Käyttöönotto, verkon rakenne ja resurssien käyttö usein *ad hoc*
Ylläpito ja huolto yleistä, fyysinen pääsy verkkoon ja sen eri osiin helppoa	Fyysinen pääsy verkkoon ja sen eri osiin usein vaikeaa tai jopa mahdotonta
Viat komponenteissa käsitellään ylläpidon ja huollon kautta	Komponenttivikoja odotetaan ja ne huomioidaan jo verkon suunnittelussa
Keskitetty hallinta ja globaali verkon tuntemus mahdollista	Monet päätökset tehdään lokaalisti ilman apuja

Eräs tärkeimmistä langattomien sensoriverkkojen kohtaamista haasteista energiankulutuksen, suunnitterelurajoitteiden ja muiden rinnalla on tietoturvallisuus. Langattomien järjestelmien avoin kommunikointiympäristö tekee niistä alttiimpia erilaisille hyökkäyksille verrattuna langallisiin järjestelmiin, missä laitteet ovat yhteydessä toisiinsa kaapeleita pitkin[1] [157]. Kun yhä useammat laitteet, järjestelmät ja tilat sekä erityisesti kriittinen infrastruktuuri ovat yhteydessä Internetiin langattomien järjestelmien kautta, vallitsevan kyberrikollisuuden jatkaessa kasvamistaan, muodostuvat tämän tyyppisiä verkkoja koskevat turvallisuus- ja yksityisyysongelmat yhä vakavimmiksi [136]. Langattomien sensoriverkkojen suojaaminen niin kyberhyökkäyksiltä kuin muilta tekijöiltä, jotka uhkaavat niiden turvallista ja luotettavaa toimintaa, on yksi tärkeimmistä langattomien sensoriverkkojen tutkimusongelmista [85].

[1]Tietoturvallisuudella tarkoitetaan kaikissa muodoissa esiintyvän tiedon suojaamista. Kyberturvallisuus on tietoturvallisuuden osa-alue, joka keskittyy digitaalisessa muodossa olevan tiedon suojaamiseen kyberavaruudessa. Huomaa, että langattomissa sensoriverkoissa digitaalisen dataan voidaan päästä käsiksi myös kyberavaruuden ulkopuolelta.

2.1 Uhat ja haavoittuvuudet

Niin langalliset kuin langattomat sensoriverkot seuraavat OSI-mallia[2] siirtäessään järjestyksessä eri otsikkotietoihin käärittyjä (*eng. encapsulate*) datapaketteja paikasta toiseen, jossa lopulta paketti avataan auki (*eng. decapsulate*) käänteisessä järjestyksessä alkuperäisen datan palauttamiseksi. Vaikka langattomien verkkojen sovellus-, kuljetus ja verkkokerrokset ovat monissa tapauksissa identtisiä langallisiin verkkoihin nähden, niiden keskeisimmät erot protokollapinon mielessä löytyvät fyysiseltä jaetulta tiedonsiirtotieltä ja laitteiden pääsynhallinnasta tiedonsiirtoa varten. Kuvassa 2.1 on esimerkki langattomien verkkojen protokollapinoista [110, 128]:



Kuva 2.1: Esimerkki langattomien verkkojen protokollapinoista.

Langallisilla ja langattomilla verkoilla on paljon yhteisiä haavoittuvuuksia, mutta niihin toisaalta kohdistuu myös monia verkkokohtaisia hyökkäyksiä. Lisäksi kullakin OSI-mallin kerroksella on omat turvallisuushaasteensa ja -ongelmansa kunkin kerroksen käyttäessä niille suunniteltuja protokollia [157]. Tämä tulee esille varsinkin esineiden internetin sovelluksissa, missä monet järjestelmät ovat heikosti suunniteltuja turvallisuusasioita laiminlyöden ja ne on otettu käyttöön teknologioilla, joita ei ole kehitetty ja testattu loppuun asti (*eng. immature technology*), kuten esimerkiksi uudet resurssiniukkiin ympäristöihin kehitetyt protokollat [127, 33]. Langattomien sensoriverkkojen puutteellinen fyysinen suojaus, rajoitetut resurssit ja avoin

[2]Tässä työssä viitataan viisikerroksiseen OSI-malliin, joka koostuu fyysisestä kerroksesta, linkki-, verkko-, kuljetus- ja sovelluskerroksesta, fyysisen kerroksen ollessa alimmalla tasolla ja sovelluskerroksen ylimmällä lähimpänä loppukäyttäjää.

kommunikointi tekevät niistä alttiita myös ulkoisille hyökkäyksille [136]. Seuraavassa on kategorisoitu lyhyesti eri tyyppisiä langattomien verkkojen kokemia yleisiä hyökkäyksiä protokollapinon suhteen [136, 157, 101]:

1. Tiedonsiirtoon liittyvät fyysiset ominaisuudet määritellään fyysisellä kerroksella, joka tekee siitä alttiin häirinnälle, salakuuntelulle ja peukaloinnille. Hyökkääjät saattavat myös tahallisesti kuluttaa tiedonsiirtotien kapasiteetin loppuun.

2. Linkkikerroksen tehtävänä on säädellä tiedonsiirtoa ja tarjota laitteille pääsy jaettuun tiedonsiirtokanavaan pääsynhallintamekanismien avulla. Häirintä ja törmäykset ovat tyypillisimpiä pääsynhallintaan liittyviä hyökkäyksiä.

3. Verkossa tapahtuvan kommunikoinnin reititystä hallinnoi verkkokerros. Laaja kirjo esimerkiksi erilaisia verkon sisäisiä harhauttamiseen ja pakettien uudelleenohjaamiseen liittyviä hyökkäyksiä voidaan laukaista kaapatun laitteen toimesta, kuten laitteiden imitointi ja replikointi sekä pakettien tahallinen toistaminen ja erilaiset aukkopohjaiset hyökkäykset (*eng. black-, grey-, sink-, wormhole*).

4. Kuljetuskerroksen tarkoituksena on taata luotettava tiedonsiirto. Laitteiden väliset yhteydet ja itse datan eheys voidaan vaarantaa muun muassa erilaisilla palvelunesto- ja energiankulutushyökkäyksillä sekä injektioilla ja datapakettien peukaloinnilla.

5. Sovelluskerroksella kommunikoidaan loppukäyttäjän kanssa ja monet uhat liittyvät kerroksella käytettävien web-palvelujen ja protokollien sekä ohjelmistojen haavoittuvuuksiin.

Hyökkäyksiä voidaan laukaista myös eri kerroksilla samanaikaisesti ja erilaisten hyökkäysten yhdistelmänä. Esimerkkejä ovat välistäveto-/välimieshyökkäys (*eng. man-in-the-middle, MITM*), hajautetut palvelunestohyökkäykset (*eng. distributed denial-of-service, DDoS*), injektiot ja sivukanavahyökkäykset (*eng. side-channel attack*) sekä edistyneemmät nollapäivähyökkäykset (*eng. zero-day*) ja APT-hyökkäykset (*eng. advanced persistent threat, APT*).

Langattomat sensoriverkot omaavat erilaisia ominaisuuksia, jotka edelleen muuttavat niiden luonnetta tietoliikenneverkkona [88]. Ensinnäkin, mittaukset eri sensorinoodeissa ja niiden välinen liikenne saattaa sisältää erilaisia esimerkiksi aikaan ja

paikkaan sidottuja keskinäisiä riippuvuuksia. Sensorinoodit voivat olla myös liikkeessä ja niillä voi olla itseorganisoituvia kykyjä, jolloin langattoman sensoriverkon topologia muuttuu ja kehittyy dynaamisesti ajan suhteen. Langattomat sensoriverkot ovat myös alttiita odottamattomille tapahtumille ja vaikutuksille esimerkiksi ympäristön ja kyberrikollisten alati kehittyvien työkalujen vuoksi. Lisäksi, kuten muissakin reaalimaailman järjestelmissä, syy-seuraussuhteet eivät ole yksinkertaisia eli pienellä muutoksella tai tapahtumalla voi olla suuri vaikutus muihin verkossa ilmeneviin signaaleihin erilaisten kerrannaisvaikutusten ja epälineaaristen suhteiden kautta. Näin ollen langattomassa sensoriverkossa esiintyvien dataa generoivien prosessien alla piilevää jakaumaa ei voida mallinnuksen kannalta olettaa staattiseksi tai tunnetuksi ennalta [93].

Yllä esitettyjen tekijöiden perusteella voidaan jo päätellä, että langattomat sensoriverkot ovat kompleksisia[3] ja epälineaarisia systeemejä, mikä osaltaan herättää kysymyksen siitä, miten tämän tyyppisiä järjestelmiä tulee lähestyä niitä hallinnoitaessa. Varsinkin kyberturvallisuuden alalla fokus ei tule olla niinkään kompleksisten systeemien yksinkertaistamisessa[4] vaan pikemminkin ymmärtämisessä ja kaikkien niiden tekijöiden huomioon ottamisessa, joita kompleksisten systeemien tarjoama data sisältää, jotta yksikään ja varsinkaan entuudestaan tuntematon hyökkäys ei jää havaitsematta. Näin ollen monesti käytettävät ennalta laadittuihin sääntöihin ja lineaarisiin mekanismeihin sekä perinteisiin tilastollisiin menetelmiin[5] perustuvat suojaustekniikat ovat riittämättömiä [93]. Tämä on tärkeää, koska viimeaikaisten lähteiden mukaan päivittäin tuotetaan lähes 230 000 uutta haittaohjelmaa. Lisäksi siitä hetkestä, kun IoT-laitteet liittyvät Internetiin, joita liittyy eräiden arvioiden mukaan 127 kappaletta sekunnissa, kestää noin 5 minuuttia kun ne löydetään ja niihin kohdistetaan hyökkäys [109, 114, 10].

[3]Jotkin systeemiin vaikuttavat tekijät ovat tuntemattomia ja eivät ole takaisin mallinnettavissa (*eng. reverse engineering*). Kompleksisuuden määrä voi myös vaihdella systeemissä, ympäristössä tai tiedonsiirrossa tapahtuvien muutosten myötä.

[4]Monilla tieteen aloilla tutkimus keskittyy epälineaaristen ilmiöiden mallintamiseen lineaaristen approksimaatioiden kautta, koska ne ovat helpommin ratkaistavissa [61].

[5]Erityisesti tiheysfunktioiden olettaminen ennalta ja muuttujien riippumattomuus sekä identtinen jakautuneisuus (*eng. independent and identically distributed, iid*) eivät ole realistisia.

2.2 Anomalioiden havaitseminen

Tiedonlouhinnan osa-aluetta, joka keskittyy harvinaisten esiintymien etsimiseen mahdollisesti suurista ja kompleksisista datajoukoista, kutsutaan anomalioiden havaitsemiseksi. Anomalioiden havaitsemisongelma itsessään on hyvin yleinen ja siitä on olemassa monia toisistaan poikkeavia määritelmiä eri sovelluksissa, kuten esimerkiksi poikkeamien, anomalioiden, puhkeamisten, tapahtumien, muutosten, petosten ja vikojen havaitseminen. Joissakin tapauksissa, kuten datan puhdistuksessa, anomalioita pidetään kohinana tai korruptoituneina havaintoina [1]. Kyberturvallisuuden sovelluksissa anomalioiden havaitsemisella on vakaa jalansija ja se tunnetaan yhtenä tunkeilijoiden havaitsemisjärjestelmien (*eng. intrusion detection system, IDS*) luokkana, väärinkäytöspohjaisten (*eng. signature-based/misuse detection*) järjestelmien rinnalla. Kyberhyökkäyksiä vastaan varautuminen ja suojautuminen jaetaan tyypillisesti puolustukseen ja ehkäisemiseen (*eng. prevention*), havaitsemiseen ja tunnistamiseen (*eng. detection*) sekä vastatoimenpiteisiin ja riskien minimointiin (*eng. mitigation*). IDS:t voidaan jakaa edelleen syötteeseen perustuen isäntä- ja verkkoliikennepohjaisiin (*eng. host-/network-based*) järjestelmiin [24]. Tässä työssä keskitytään anomalioiden havaitsemiseen verkkoliikenteestä.

Väärinkäytöspohjaisessa järjestelmässä haitallisen toiminnan tunnistamiseen käytetään verkkoliikenteeseen verrattavia tunnettujen hyökkäysten ja niistä johdettuja tunnusmerkkejä, kun taas anomaliapohjaisessa lähestymistavassa kaikki normaalia käyttäytymistä kuvaavasta verkkoliikenneprofiilista poikkeavat havainnot luokitellaan anomalioiksi, joiden joukkoon hyökkäykset oletetusti kuuluvat. Väärinkäytöspohjaisen järjestelmän ongelma on se, että vaikka se tunnistaa hyvin tunnettuja hyökkäyksiä, se ei kykene havaitsemaan hyökkäyksiä sen tunnusmerkkien ulkopuolelta eli sille uusia hyökkäyksiä. Anomaliapohjaiset järjestelmät sen sijaan pystyvät havaitsemaan ennalta tuntemattomia hyökkäyksiä, mutta eräs niiden ongelmista on korkea väärien hälytysten määrä erityisesti dynaamisissa ympäristöissä, joissa normaalin liikenteen profiili ei pysy vakiona [115].

Koneoppimiseen ja erityisesti syväoppimiseen perustuvien menetelmien vakuuttava suorituskyky monissa eri sovelluksissa on puhuttanut tieteellisessä yhteisössä viime vuosikymmenen aikana. Syväoppimismenetelmien menestys perustellaan tyypillisesti niiden poikkeuksellisena kykynä käsitellä moniulotteista ja esiprosessoimatonta eli raakaa dataa ja oppia automaattisesti monimuotoisia piirteitä joko pienin tai mitättömin esioletuksin [91]. Syväoppimisen historian eräs dramaattisimmista hetkistä lienee konvoluutioneuroverkkoon perustuvan menetelmän suveree-

ni menestys suurimmassa objektintunnistuskilpailussa[6] vuonna 2012, jonka jälkeen kyseisen kilpailun voittaneet ovat aina käyttäneet konvoluutioneuroverkkoihin perustuvia menetelmiä [75, 67, 149]. Syväoppimismenetelmät ovat keränneet menestystä myös puheentunnistuksessa ja luonnollisen kielen prosessoinnissa sekä monissa muissa sovelluksissa [91, 54].

Syväoppiminen itsessään ei ole uusi menetelmä, vaan sen 1940-luvulle ulottuvassa historiassa se on kantanut montaa eri nimeä. Nykyisellään se nousi uudestaan esille vuonna 2006 kehittyneempien optimointialgoritmien myötä, minkä seurauksena syvät neuroverkot alkoivat nousta perinteisten koneoppimismenetelmien suorituskyvyn ohi. Nykypäivän syväoppimisalgoritmit perustuvat pitkälti samoihin, jo 1980-luvullakin esiintyneisiin menetelmiin, mutta silloin ei ollut olemassa tänä päivänä saatavia olevia laskennallisia resursseja ja dataa eikä algoritmikehitys ollut yleisesti ottaen samalla tasolla [54]. Nykyään syväoppiminen on läsnä monissa eri sovelluksissa niin teollisuuden, kaupankäynnin kuin kuluttajien keskuudessa, ja sen menestyksen myötä syväoppiminen on löytänyt tiensä myös anomalioiden havaitsemismenetelmiin monilla eri tieteenaloilla [91, 25].

Tämä pitää paikkansa myös langattomiin sensoriverkkoihin liittyvässä anomaliapohjaisten tunkeilijan havaitsemisjärjestelmien tutkimuksessa, mutta monet hiljattain kehitetyt menetelmät keskittyvät muun muassa tunnistamaan vain tietyn tyyppisiä hyökkäyksiä, niiden testaamiseen on käytetty joko vanhentuneita tai muuta kuin langatonta sensoriverkkoa edustavaa liikennettä ja ne tekevät vahvoja oletuksia datasta sekä vaativat datan esiprosessointia. Näin ollen siirtyminen yleiskäyttöisempiin syväoppimismenetelmiin voi laajentaa menetelmien kykyä tunnistaa erilaisia ja varsinkin ennestään tuntemattomia hyökkäyksiä [93].

[6] ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [122]. Viimeinen kilpailu pidettiin vuonna 2017.

3 Syväoppiminen

Koneoppiminen[1] on kattotermi tietokoneille näennäisesti itsenäisen oppimis- ja reagoimiskyvyn mahdollistaville algoritmeille ja tekniikoille [95]. Tarkemmin sanottuna koneoppiminen voidaan ymmärtää[2] sellaisen mallin opettamiseksi saatavilla olevasta datasta, jonka päätöksenteon yleistämiskykyä mitataan jotakin suorituskykyä mittaavaa metriikkaa vastaan [20]. Koneoppimismenetelmät tarjoavat yleiskäyttöisiä ja joustavia ratkaisuja, jotka pystyvät parantamaan suorituskykyään kokemusperäisesti oppimalla [85]. Perinteisen ohjelmoinnin ja koneoppimisalgoritmien keskeinen ero on se, että perinteisessä ohjelmoinnissa tuotetaan vastauksia datan ja ennalta laadittujen sääntöjen pohjalta, kun taas koneoppiminen keskittyy sääntöjen etsintään datan ja mahdollisesti saatavilla olevien vastausten avulla [28].

Yleisesti ottaen koneoppimismenetelmät voidaan jakaa ohjattuihin ja ohjaamattomiin oppimisalgoritmeihin. Ohjatussa oppimisessa mallille annetaan opetusdatajoukkona mahdollisesti moniulotteisia syöte- ja tavoitepareja eli opetusesimerkkejä $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$. Luokittelutehtävissä tavoite on tyypillisesti jokin tietty joukko syötteitä ryhmitteleviä luokkia ($\mathbf{y}_n \in \{0, \cdots, k\}$) kun taas regressio-ongelmissa se voi saada arvonaan reaalilukuja ($\mathbf{y}_n \in \mathbb{R}^m$). Käytännössä ohjattu oppiminen käsittää sellaisen parametrijoukon θ etsimistä, joka opetusaineiston pohjalta muodostetun parametrisen funktion $\mathcal{F}_\theta$ avulla mahdollistaa virheettömimmät ennustukset tai luokittelut kustannusfunktion $\mathcal{L}(\hat{\mathbf{y}}_n, \mathbf{y}_n)$ perusteella, missä $\hat{\mathbf{y}}_n = \mathcal{F}_\theta(\mathbf{x}_n)$. Toisin sanoen oppiessaan malli approksimoi dataa (opetusaineiston) generoivaa prosessia parametrisen funktion $\mathcal{F}_\theta$ kautta, joka luo jatkuvan kuvauksen opetusdatasta tavoitteisiin[3] [94].

Kun syötteille ei ole niitä vastaavia tavoitteita saatavilla, eli opetusesimerkkejä ei ole mahdollista laatia, siirrytään ohjaamattoman oppimisen puolelle. Tällöin oppi-

[1] Koneoppimista pidetään yhtenä tekoälymenetelmien (*eng. artificial intelligence, AI*) osajoukkona.

[2] Eräs tunnettu koneoppimisen määritelmä Tom Mitchelliltä [100]: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, measured by P, improves with experience E"*.

[3] Ajatellaan, että menetelmä saa n-ulotteisen syötteen ja antaa m-ulotteisen ulostulon. Tällöin menetelmälle pätee jatkuva kuvaus n-ulotteisesta Euklidisesta avaruudesta m-ulotteiseen Euklidiseen avaruuteen, eli $\mathcal{F}_\theta : \mathbb{R}^n \mapsto \mathbb{R}^m$.

misprosessin tarkoituksena on löytää toistuvia piirteitä ja samankaltaisuuksia sekä suhteita eri syötteiden välille. Ohjaamattomaan oppimiseen kuuluu myös sellaisten esitysmuotojen löytäminen, jossa data voidaan esittää muutaman tärkeimmän muuttujan tai ulottuvuuden avulla. Eräitä tunnettuja ohjaamattomia koneoppimisalgoritmeja ovat pääkomponenttianalyysi (*eng. principal component analysis, PCA*), klusterointimenetelmät (*eng. clustering*) ja itseohjautuva kartta (*eng. self-organizing map, SOM*). Eräs käytännön esimerkki ohjaamattomasta oppimisprosessista on rekonstruoida mallille annettu syöte siitä johdetusta alemman ulottuvuuden tai kohinaisesta esityksestä, jolloin ennustusten virhettä mitataan rekonstruktiovirheen $\mathcal{L}(\hat{\mathbf{x}}_n, \mathbf{x}_n)$ avulla, missä $\hat{\mathbf{x}}_n = \mathcal{F}_\theta(\mathbf{x}_n)$ [94].

Ohjatun ja ohjaamattoman oppimisen lisäksi yleisesti tunnettu oppimiskategoria koneoppimismenetelmille on puoliohjattu oppiminen, joka sijoittuu ohjattujen ja ohjaamattomien menetelmien välimaastoon. Puoliohjatussa oppimisessa vain pienelle osalle syötteistä on saatavilla tavoitteet eli opetusesimerkkejä voidaan luoda pieni määrä, joiden avulla autetaan muuten ohjaamatonta oppimisprosessia. Puoliohjattu oppiminen on relevantti vaihtoehto silloin, kun tavoitteiden laatiminen koko datajoukolle on kallista tai jopa mahdotonta [81]. Eräs tärkeä oppimiskategoria on myös vahvistettu oppiminen, joka mahdollistaa koneen oppimisen ympäristöstä saamansa palautteen perusteella. Tällöin koneen on mahdollista tehdä omaa hyötyä kumulatiivisesti maksimoivia päätöksiä ympäristössä, jossa se toimii [47]. Vahvistettu oppiminen on muuten luonteeltaan ohjaamatonta.

Koneoppimismenetelmät voidaan myös jakaa pinnalliseen oppimiseen (*eng. shallow learning*) ja syväoppimiseen (*eng. deep learning*). Teknillisesti ero näiden kahden välillä on yksinkertainen: syväoppijoiksi kutsutaan erilaisia enemmän kuin yhden piilokerroksen omaavia keinotekoisia neuroverkkoarkkitehtuureja ja pinnallisiksi oppimismenetelmiksi lasketaan kaikki ne, jotka eivät sovi tuohon kuvaukseen syväoppimisesta [6]. Niiden ero käytännössä on kuitenkin ratkaiseva: pinnalliset oppimismenetelmät tekevät paljon esioletuksia käsittelyssä olevasta datasta, vaativat tyypillisesti käsin laadittuja sääntöjä ja ominaisuuksia sekä sovelluskohtaista asiantuntijuutta, kun taas syväoppimismenetelmillä on mahdollista oppia kompleksisia ja epälineaarisia eri abstraktiotason omaavia esitysmuotoja joko pienin tai mitättömin esioletuksin suoraan moniulotteisesta datajoukosta [6, 95, 91].

Syväoppiminen on siis eräs koneoppimisen osa-alue, jonka lähestymistapa dataa generoivan prosessin mallintamiseen perustuu kasvavissa määrin merkityksellisten esitysmuotojen (*eng. meaningful representation*) oppimiseen peräkkäin aseteltujen

toisiinsa vaikuttavien kerrosten kautta. Kuten mainittua, syväoppiminen tapahtuu käytännössä keinotekoisilla neuroverkoilla. Termi "neuroverkko" juontaa neurobiologiasta: monia syväoppimisessa esiintyviä keskeisiä konsepteja on inspiroinut ymmärrys ihmisen aivotoiminnasta, kuten aivojen yksittäistä neuronia mallintava Rosenblattin [14] perseptroni (*eng. perceptron*) ja näköaivokuoren toimintaa hahmotteleva Fukushiman [48] neokognitroni (*eng. neocognitron*). Todisteita ei kuitenkaan ole siitä, että ihmisen aivot toimivat ja käyttäytyvät moderneissa syväoppimismalleissa käytettyjen mekanismien lailla. Syväoppimisen sanotaankin olevan matemaattinen viitekehys erilaisten esitysmuotojen oppimiseen datasta [28].

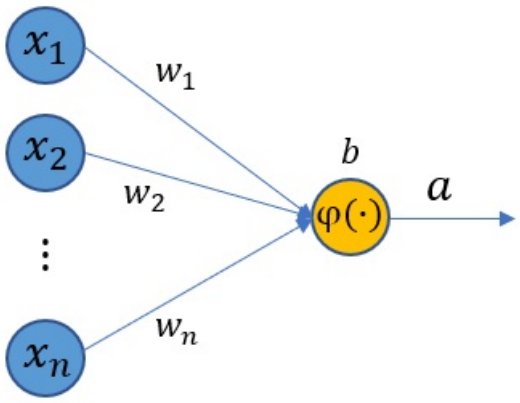
Erilaisten syväoppimismenetelmien kirjo on hyvin laaja: erilaisia neuroverkkoarkkitehtuureja on katselmoitu alkuperäisine viitteineen muun muassa lähteessä [140]. Seuraavissa aliluvuissa esitellään yleisimmät menestystä keränneet neuroverkkotyypit, joita ovat keinotekoiset neuroverkot, konvoluutioneuroverkot ja toistuvat neuroverkot.

3.1 Keinotekoiset neuroverkot

Keinotekoiset neuroverkot ovat koneoppimisalgoritmeja, jotka pohjustavat monia syväoppimismenetelmiä. Keinotekoinen neuroverkko (*eng. artificial neural network, ANN*) koostuu jonkin aktivaation a ja parametrit $\theta = \{\mathcal{W}, \mathcal{B}\}$ omaavista laskentayksiköistä tai neuroneista, missä $\mathcal{W}$ on joukko painokertoimia (*eng. weight*) ja $\mathcal{B}$ joukko vakiotermejä (*eng. bias*). Aktivaatio edustaa syötteen $\mathbf{x} \in \mathbb{R}^n$ ja neuronin parametrien lineaarikombinaatiota eli painotettua summaa, jota seuraa epälineaarinen aktivointifunktio $\varphi(\cdot)$ [94], jolloin

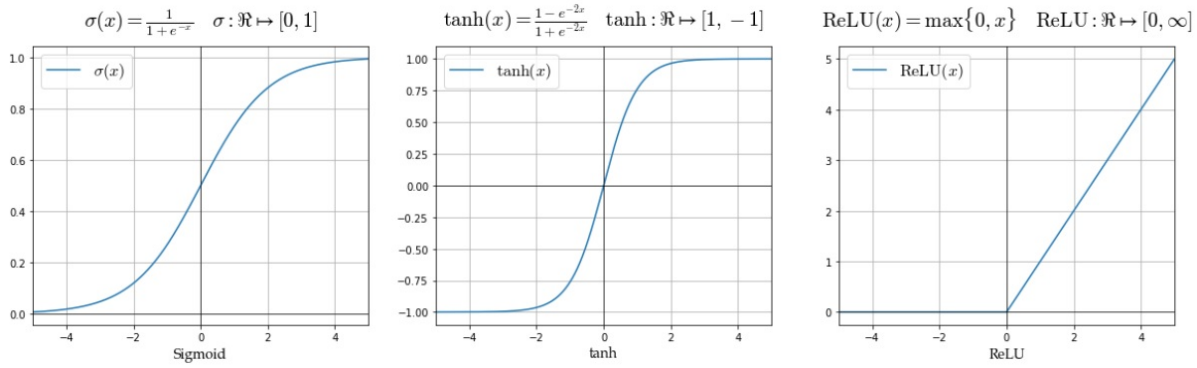
$$
\begin{aligned}
a &= \varphi(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b) && (3.1)\\
&= \varphi\Big(\sum_{n=1}^{N} w_n x_n + b \Big) && (3.2)\\
&= \varphi\Big(\begin{bmatrix} w_1 & w_2 & \dots & w_N \end{bmatrix}^\top \begin{bmatrix} x_1 & x_2 & \dots & x_N \end{bmatrix} + b \Big) && (3.3)\\
&= \varphi(\mathbf{w}^\top \mathbf{x} + b). && (3.4)
\end{aligned}
$$

Huomaa, että epälineaarisen aktivointifunktion syöte on affiini kuvaus kaavan 3.4 mukaisesti. Kuva 3.1 havainnollistaa yksittäisen neuronin toimintaa.



Kuva 3.1: Yksittäisen neuronin toiminta.

Tavallisesti aktivaatiossa käytettyjä epälineaarisia funktioita, tai aktivointifunktioita, ovat sigmoidina tunnettu logistinen funktio[4] ja hyperbolinen tangentti (tanh). Neuroverkkoja optimoitaessa niillä on kuitenkin havaittu ongelmallisia ominaisuuksia, joten kirjallisuudessa on esitetty myös muita aktivointifunktioita — eräs yleisimmistä ja suotuisimmista on ReLU-funktio [4]. Jos epälineaarisia aktivointifunktioita ei käytettäisi, rajoittuisi neuroverkon mallinnuskyky pelkästään lineaarisiin muunnoksiin. Toisin sanoen neuroverkon mallinnuskyky kasvaa ja laajenee huomattavasti epälineaaristen aktivointifunktioiden myötä [28]. Kuva 3.2 havainnollistaa yleisesti käytettyjen aktivointifunktioiden kuvauksia ja määrittelyjoukkoja [139].

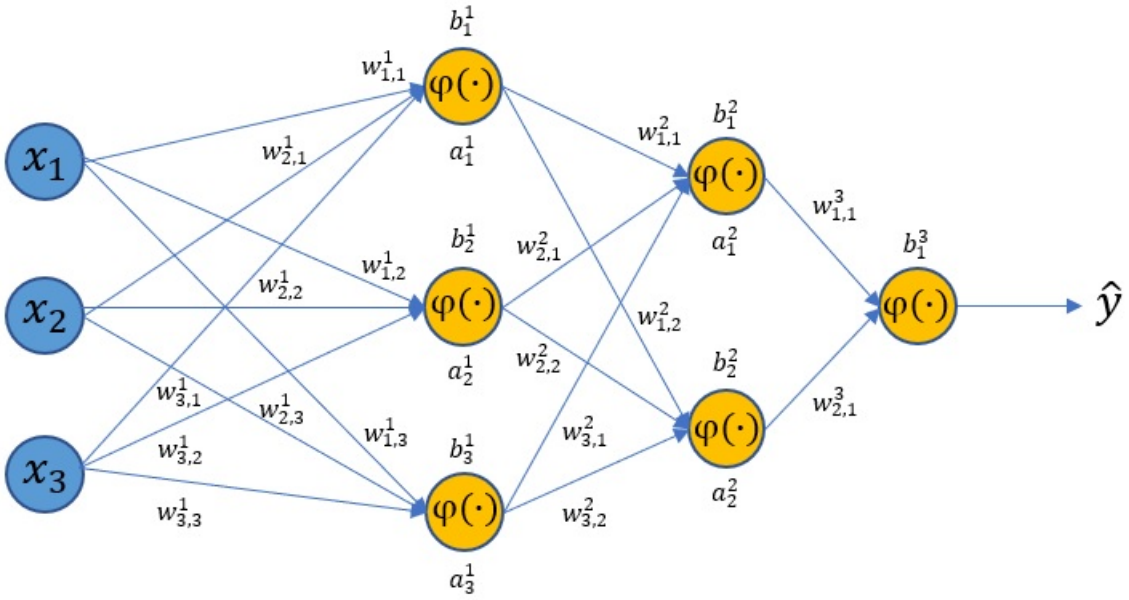


Kuva 3.2: Yleisesti käytetyt aktivointifunktiot.

Neuroverkoista perinteisesti tunnetuin on monikerroksinen perseptroni (*eng. mul-*

[4]Sigmoid-funktiolla varustettu yksi neuroni vastaa käytännössä logistista regressiomallia. Jos aktivointifunktiota ei käytetä, neuroni vastaa lineaarista regressiomallia.

tilayer perceptron, MLP), missä edellä esitettyjä syötteitä käsitteleviä neuroneita kootaan rinnakkain ja kerroksittain. Tarkastellaan kolme muuttujaa syötteenä ottavaa esimerkkineuroverkkoa, jossa on kaksi piilokerrosta ja ulostulokerros kuvan 3.3 mukaisesti[5].



Kuva 3.3: Kahden piilokerroksen omaava esimerkkineuroverkko.

Merkitsemällä syötettä $\{x_n\}_{n=1}^3 = \{a_n^0\}_{n=1}^3$ ja hyödyntämällä edellä esitettyä matriisinotaatiota, esimerkkineuroverkon ulostulokerroksen neuroni antaa tuloksen $\hat{y}$ välivaiheiden

$$
\begin{aligned}
a_1^1 &= \varphi\left(\begin{bmatrix} w_{1,1}^1 & w_{2,1}^1 & w_{3,1}^1 \end{bmatrix}^\top \begin{bmatrix} a_1^0 & a_2^0 & a_3^0 \end{bmatrix} + b_1^1 \right) \\
a_2^1 &= \varphi\left(\begin{bmatrix} w_{1,2}^1 & w_{2,2}^1 & w_{3,2}^1 \end{bmatrix}^\top \begin{bmatrix} a_1^0 & a_2^0 & a_3^0 \end{bmatrix} + b_2^1 \right) \\
a_3^1 &= \varphi\left(\begin{bmatrix} w_{1,3}^1 & w_{2,3}^1 & w_{3,3}^1 \end{bmatrix}^\top \begin{bmatrix} a_1^0 & a_2^0 & a_3^0 \end{bmatrix} + b_3^1 \right) \\
a_1^2 &= \varphi\left(\begin{bmatrix} w_{1,1}^2 & w_{2,1}^2 & w_{3,1}^2 \end{bmatrix}^\top \begin{bmatrix} a_1^1 & a_2^1 & a_3^1 \end{bmatrix} + b_1^2 \right) \\
a_2^2 &= \varphi\left(\begin{bmatrix} w_{1,2}^2 & w_{2,2}^2 & w_{3,2}^2 \end{bmatrix}^\top \begin{bmatrix} a_1^1 & a_2^1 & a_3^1 \end{bmatrix} + b_2^2 \right) \\
\hat{y} &= \varphi\left(\begin{bmatrix} w_{1,1}^3 & w_{2,1}^3 \end{bmatrix}^\top \begin{bmatrix} a_1^2 & a_2^2 \end{bmatrix} + b_1^3 \right)
\end{aligned}
$$

[5]Yhden neuronin ulostulokerros on tyypillinen yhden muuttujan regressio-ongelmissa, kuten esimerkiksi jonkin osake- tai valuuttakurssin ennustamisessa. Moniluokkaisissa luokittelu-ongelmissa sen sijaan kullekin luokalle on ulostulokerroksella yleensä oma neuroninsa.

kautta. Sievennetään niin, että kerätään muuttujat yhteen kerroksittain matriisimuodossa tarvittavine transpooseineen[6], jolloin

$$\begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{bmatrix} = \varphi\left(\begin{bmatrix} w_{1,1}^1 & w_{2,1}^1 & w_{3,1}^1 \\ w_{1,2}^1 & w_{2,2}^1 & w_{3,2}^1 \\ w_{1,3}^1 & w_{2,3}^1 & w_{3,3}^1 \end{bmatrix} \begin{bmatrix} a_1^0 \\ a_2^0 \\ a_3^0 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix} \right)$$

$$\begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix} = \varphi\left(\begin{bmatrix} w_{1,1}^2 & w_{2,1}^2 & w_{3,1}^2 \\ w_{1,2}^2 & w_{2,2}^2 & w_{3,2}^2 \end{bmatrix} \begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{bmatrix} + \begin{bmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \end{bmatrix} \right)$$

$$\hat{y} = \varphi\left(\begin{bmatrix} w_{1,1}^3 & w_{2,1}^3 \end{bmatrix} \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix} + \begin{bmatrix} b_1^3 \end{bmatrix} \right).$$

Edellä huomataan, miten painokerroinmatriisin kunkin rivin painot kohdistuvat tiettyyn neuroniin. Samalla nähdään, että painokerroinmatriisin koko $(N_l \times N_{l'})$[7] määräytyy nykyisen ja edellisen kerroksen neuronien lukumäärän perusteella. Merkitään kunkin kerroksen painokerroinmatriisia $\mathbf{W}$ ja aktivaatioiden sekä vakiotermien muodostamaa sarakevektoria $\mathbf{a}$ ja $\mathbf{b}$, vastaavasti. Tällöin saadaan kompaktit esitysmuodot käyttämällä kerroskohtaisia indeksejä, eli

$$\begin{aligned} \mathbf{a}^1 &= \varphi(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1) \\ \mathbf{a}^2 &= \varphi(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) \\ \hat{\mathbf{y}} &= \varphi(\mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3), \end{aligned}$$

tai vaihtoehtoisesti

$$\hat{\mathbf{y}} = \varphi(\mathbf{W}^3\varphi(\mathbf{W}^2\varphi(\mathbf{W}^1\mathbf{a^0} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3).$$

Yllä olevan esitysmuodon perusteella voidaan johtaa

$$\mathbf{y} \approx \hat{\mathbf{y}} = \mathcal{F}_\theta(\mathbf{x}) = \varphi(\mathbf{W}^L\varphi(\mathbf{W}^{L'} \cdots \varphi(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1)) + \mathbf{b}^{L'}) + \mathbf{b}^L), \tag{3.5}$$

[6]Käytetään sarakevektorimuotoista esitystä aktivoinneille, mistä seuraa muiden parametrien esitysmuodot. Tästä on hyötyä esimerkiksi Kerasilla implementoitaessa neuroverkkoja, kun syötteet tulee antaa sarakevektoreina. Huomaa, että $\phi \begin{bmatrix} x & y & z \end{bmatrix}^\top = \begin{bmatrix} \phi(x) & \phi(y) & \phi(z) \end{bmatrix}^\top$.

[7]Huomaa, $l' = l^{l-1}$. Tätä merkintätapaa käytetään paikoitellen läpi työn.

joka on yleinen muoto L-kerroksiselle perseptronille[8], missä painomatriisit $\{\mathbf{W}^l\}_{l=1}^{L}$ koostuvat riveistä painokertoimia $\mathbf{w}_k$, jotka ovat liittyneet tiettyyn aktivaatioon eli neuroniin k [94]. Syöte- ja ulostulokerroksen välisiä kerroksia kutsutaan usein piilokerroksiksi johtuen siitä, että neuroverkkoa optimoitaessa piilokerroksiin liittyvät automaattisesti määräytyvät parametrit eivät ole käyttäjälle samalla tavalla näkyvissä kuten syötteet ja ulostulot ovat. Keinotekoista neuroverkkoa voidaan ajatella esimerkiksi parametrisena funktiona, joka saa syötteenään n-ulotteisen vektorin, piilokerrokset vastaavat raskaasta laskennasta ja funktion m-ulotteinen arvo saadaan ulostulokerroksesta. Verkon käyttötarkoitus edelleen määrää, miten ulostulo tulkitaan; onko kyse esimerkiksi luokittelusta vai regressiosta. Huomionarvoista on myös se, että edellä esitettyä neuroverkkomallia kutsutaan myös eteenpäin syöttäväksi neuroverkoksi (*eng. feedforward neural network, FFNN*) syötekerrokselta ulostulokerrokselle asti tapahtuvan laskennan ollessa silmukaton (*eng. acyclic*), eli tietoa ei kierrätetä arkkitehtuurin eri osissa uudestaan.

Epälineaaristen aktivointifunktioiden käyttö mahdollistaa epälineaaristen ja ei-konveksisten funktioiden mallintamisen, mutta toisaalta neuroverkkojen optimointi on tällöin hankalaa. Tarkemmin sanottuna syntyvälle rajoittamattomalle ei-konveksiselle optimointiongelmalle ei ole olemassa globaalin optimin takaavaa suljettua ratkaisua [84]. Neuroverkkojen optimointi tapahtuu epäsuorasti minimoimalla sellaista kustannusfunktiota, jonka odotetaan parantavan neuroverkkomallin suorituskykyä mittaavaa metriikkaa. Tämä eroaa ratkaisevasti suorasta optimoinnista, jossa tavoitteena on optimoida kohdefunktiota itsessään [54]. Dataa generoivan prosessin approksimointi epäsuorasti optimoimalla havainnollistaa myös neuroverkkojen mustaa laatikkomaisuutta (*eng. black box*). Esimerkiksi ohjatussa oppimisessa neuroverkkojen optimointiongelma on tyypillisesti muotoa

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathcal{F}_\theta(\mathbf{x}_n), \mathbf{y}_n), \tag{3.6}$$

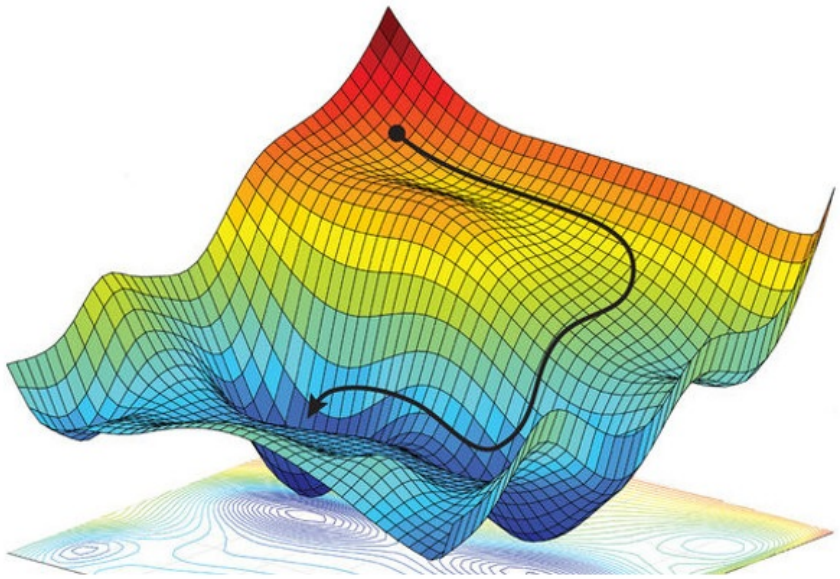
missä minimoidaan mallin antamien tulosten ja vastaavien tavoitteiden keskiarvoistettua kokonaisvirhettä annettujen syötteiden perusteella[9] [84, 30]. Edellä esi-

[8]Kerroksen l neuronin k aktivaatio a_k^l summamuodossa on $a_k^l = \varphi\Big(\sum_{i=1}^{L} w_{i,k}^l a_i^{l'} + b_k^l\Big)$. Matriisimuoto on kuitenkin huomattavasti kompaktimpi ja edullisempi implementoida matriisioperaatioita tukevilla ohjelmointikielillä.

[9]Kaavaa 3.6 voidaan ajatella empiirisen riskin minimointina (*eng. empirical risk minimization, ERM*), missä todellisen dataa generoivan prosessin jakaumaa approksimoidaan siitä kerätyn otoksen eli opetusdatajoukon avulla [118].

tettyä ongelmaa minimoimalla voidaan tarkastella esimerkiksi mallin ennustuskyvyn tarkkuuden kehittymistä. Neuroverkot ovat tyypillisesti moniulotteisia ja niissä voi olla sisäisiä parametreja lukematon määrä, mikä osaltaan rajoittaa valintaa käytettävästä optimointialgoritmista: sen tulee olla laskennallisesti kevyt, luotettava eli konvergoituu ja löytää mahdollisimman suotuisan (lokaalin) minimin. Ensimmäisen asteen gradienttimenetelmä on soveltunut tähän hyvin keveytensä ja suoraviivaisen käytettävyytensä vuoksi [80].

Gradienttimenetelmässä tarvitaan vain gradientteja, jotka koostuvat ensimmäisen asteen osittaisderivaatoista moniulotteisessa tapauksessa. Menetelmässä liikutaan iteratiivisesti negatiivisen gradientin suunnassa käyvien ratkaisujen joukkoa pitkin kohti minimiä[10]. Kuva 3.4 havainnollistaa gradienttimenetelmän toimintaa kolmiulotteisen ei-konveksisen optimointiongelman tapauksessa [3].



Kuva 3.4: Gradienttimenetelmän toimintaa havainnollistava esimerkki ei-konveksisen optimointiongelman tapauksessa.

Neuroverkkojen kustannusfunktion minimoinnissa tarvitaan mallin parametrikohtaisia osittaisderivaattoja, jotka yhtenäiseen verkkoon ja derivaatan ketjusääntöön nojaten vaikuttavat toistensa arvoihin - kokonaisvirheestä ulostuloon aina verkon syötekerrokselle asti. Tästä syystä neuroverkon opettamista gradienttien avulla kut-

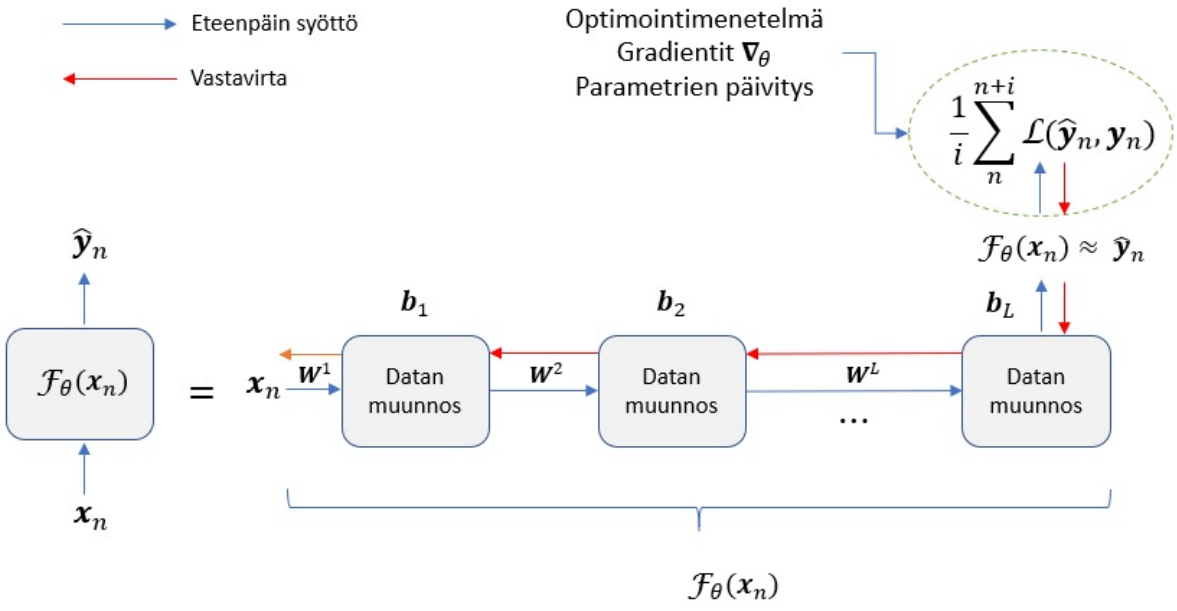
[10]Kustannusfunktiota minimoidaan usein navigoimalla hyvin moniulotteisessa parametriavaruudessa.

sutaan myös vastavirta-algoritmiksi (*eng. backpropagation*). Mallin antamiin tuloksiin vaikuttavia neuroverkon sisäisiä parametreja säädetään optimaalisesti [91]. Käytännössä gradienttimenetelmästä käytetään usein näennäisesti stokastista versiota[11], missä mallille annetaan kerrallaan pieni osa opetusesimerkkejä $\mathcal{D}_{n+i} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_n^{n+i}$ (*eng. mini-batch*), joiden perusteella lasketun virheen avulla mallin satunnaisesti alustetut parametrit päivitetään iteratiivisesti

$$\mathbf{w} \leftarrow \mathbf{w}' - \eta \frac{1}{i} \sum_{n}^{n+i} \nabla_{\mathbf{w}} \mathcal{L}(\mathcal{F}_\theta(\mathbf{x}_n), \mathbf{y}_n) \tag{3.7}$$

$$\mathbf{b} \leftarrow \mathbf{b}' - \eta \frac{1}{i} \sum_{n}^{n+i} \nabla_{\mathbf{b}} \mathcal{L}(\mathcal{F}_\theta(\mathbf{x}_n), \mathbf{y}_n), \tag{3.8}$$

missä η on käyttäjän antama kerroin tai askelkoko, jota kutsutaan neuroverkkokirjallisuudessa oppimisnopeudeksi (*eng. learning rate*) [118, 80, 120]. Parametrien päivittämistä jatketaan tyypillisesti opetusesimerkkejä alusta loppuun niin kauan läpi käyden, kunnes kokonaisvirhe saturoituu[12]. Kuva 3.5 havainnollistaa stokastisen vastavirta-algoritmin toimintaa eteenpäin syöttävän neuroverkon tapauksessa.



Kuva 3.5: Stokastinen vastavirta-algoritmi eteenpäin syöttävässä neuroverkossa.

[11] Pienempiin havaintomääriin perustuvat parametripäivitykset ovat kohinaisempia, eivätkä niin tarkkoja, mistä juontaa menetelmän stokastisuus. Päivitysten yhteydessä parametrien heilunta toisaalta vähenee, konvergioituminen on vakaampaa ja nopeampaa sekä vie vähemmän resursseja.

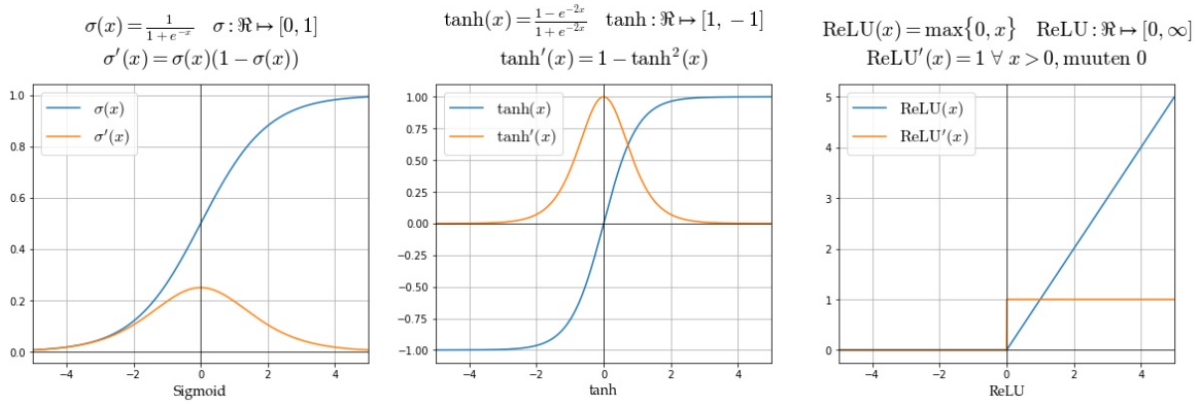
[12] Yhtä kierrosta, jossa kaikki opetusesimerkit käydään osissa läpi, kutsutaan epookiksi (*eng. epoch*).

Kaavoissa 3.7 ja 3.8 esitetyllä gradienttimenetelmällä on kuitenkin monia puutteita, mikä tekee siitä epäluotettavan oppimisprosessin kannalta. Näin ollen tiedeyhteisössä on kehitetty vuosien varrella monia erilaisia gradienttimenetelmän variantteja, jotka pyrkivät vastaamaan sen heikkouksiin. Lisätietoa näistä löytyy esimerkiksi Ruderin [121] kattavasta artikkelista sen sisältämine lähteineen[13].

Kaavoissa 3.7 ja 3.8 esitetyssä gradienttimenetelmässä oppimisnopeus ei yksin määritä, miten paljon parametreja päivitettäessä alkuperäinen arvo muuttuu, vaan gradientin arvolla on myös keskeinen merkitys. Kustannusfunktion gradienttia laskiessa tullaan ketjusäännön nojalla laskemaan myös aktivointifunktioiden osittaisderivaattoja. Jos esimerkiksi merkitään, että kerroksen l jokin painokerroin on $w^l_{i,k}$ ja aktivaatio vastaavasti $a^l_k = \varphi(z^l_k)$, missä aktivaation syöte on $z^l_k = w^l_{i,k} a^{l'}_i + b^l_k$, niin yhden opetusesimerkin kustannuksen osittaisderivaatta painon suhteen $w^l_{i,k}$ on $\frac{\partial \mathcal{L}(\hat{\mathbf{y}}_n, \mathbf{y}_n)}{\partial w^l_{i,k}} = \frac{\partial z^l_k}{\partial w^l_{i,k}} \frac{\partial a^l_k}{\partial z^l_k} \frac{\partial \mathcal{L}(\hat{\mathbf{y}}_n, \mathbf{y}_n)}{\partial a^l_k}$. Kuten nähdään, laskutoimitus vaatii niin kustannuskuin aktivaatiofunktion derivaatan käyttöä eli niiden valinnalla on vaikutusta neuroverkon oppimisprosessiin. Huomaa, että osittaisderivaatat $\frac{\partial \mathcal{L}}{\partial w^l_{i,k}}$ muodostavat gradientin $\nabla_{\mathbf{w}}$ komponentit ja analogisesti saadaan vakiotermien osittaisderivaatoista koostuva gradientti $\nabla_{\mathbf{b}}$. Huomaa myös, miten edellisen kerroksen aktivaatio $a^{l'}$ sisältyy aritmetiikkaan: neuroverkon painokertoimista riippuvien aktivaatioiden muutokset vaikuttavat rekursiivisesti muihin aktivaatioihin kaavan 3.5 sekä kuvien 3.3 ja 3.5 mukaisesti[14]. Vastavirta-algoritmissa laskettavia osittaisderivaattoja kutsutaankin toisinaan totaaleiksi derivaatoiksi (*eng. total derivative*), koska eri muuttujien suhteen laskettavat osittaisderivaatat riippuvat epäsuorasti muista muuttujista ketjusäännön nojalla. Kuva 3.6 havainnollistaa yleisesti käytettyjen aktivointifunktioiden derivaattoja.

[13] Ruderin artikkeli perustuu hänen laatimaan blogikirjoitukseensa [120], jossa on ajankohtaisempaa tietoa.

[14] Eteenpäin syöttävien neuroverkkojen ja vastavirta-algoritmin matematiikkaa on käyty läpi muun muassa lähteissä [12, 105, 89, 73, 139, 111]. Lisäksi vastavirta-algoritmista löytyy monia erilaisia ymmärtämistä helpottavia visualisointeja omaavia videoita, kuten lähteessä [124].

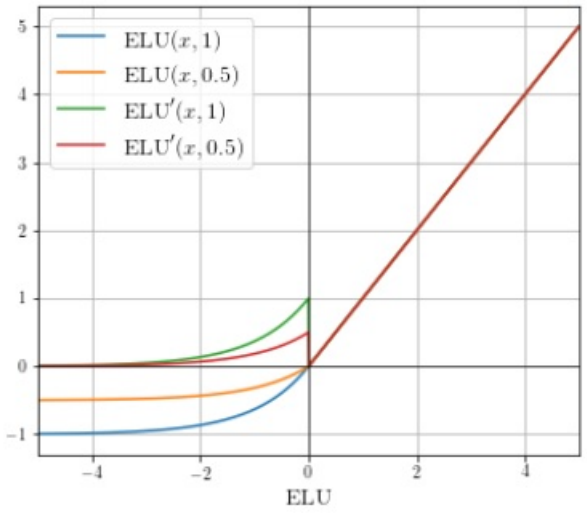


Kuva 3.6: Yleisesti käytetyt aktivointifunktiot ja niiden derivaatat.

Kuvasta 3.6 ja derivaattojen kaavoista nähdään, että sigmoid-funktioon saapuvan painokertoimista ja syötteestä riippuvan arvon saadessa suuria positiivisia tai negatiivisia arvoja, sen derivaatta pienenee nopeasti lähelle nollaa, josta seuraa kyseisen neuronin aktivaation pimentyminen vastavirta-algoritmin aikana. Tällöin neuronin läpi ei virtaa tietoa ja ketjusäännön nojalla myöskin sitä seuraavien neuronien osittaisderivaatat ovat pieniä. Pahimmassa tapauksessa neuroverkon oppiminen pysähtyy eli saturoituu liian aikaisin. Sigmoidin ei-symmetrisyys nollan suhteen voi aiheuttaa myös häiriöitä vastavirta-algoritmin askelten suunnissa. Hyperbolinen tangetti on symmetrinen nollan suhteen, mutta sen derivaatta pienenee suurilla syötteillä nopeasti sigmoidin tapaan. ReLU-funktio on yksinkertainen aktivointifunktio, jonka derivaatta on positiivisilla syötteillä aina 1 ja jonka on osoitettu nopeuttavan huomattavasti neuroverkkojen oppimista. Vaikka ReLU on eniten käytetty aktivointifunktio käytännön sovelluksissa, se ei ole täysin ongelmaton: negatiivisilla syötteillä ja nollassa sen derivaatta on nolla, joten neuroverkon ennenaikainen saturoituminen on tässäkin tapauksessa mahdollista. ReLU-funktiosta on edelleen kehitetty saturoitumista välttäviä muunnelmia kuten LeakyReLU $= \max\{ax, x\}$, missä a on manuaalisesti säädettävä parametri [76, 139, 89]. Parametrinen ReLU (*eng. parametric ReLU, PReLU*) määritellään kuten LeakyReLU, mutta a on neuroverkon mukana opittava parametri. Eräs muunnelma on ELU-funktio (*eng. exponential linear unit, ELU*), joka määritellään kuvan 3.7 esittämällä tavalla.

$$\mathrm{ELU}(x,\alpha) = \left\{ \begin{array}{ll} \alpha(e^x - 1) & x \leq 0 \\ x & x > 0 \end{array} \right\} \quad \mathrm{ELU} : \Re \mapsto [-\alpha, \infty]$$

$$\mathrm{ELU}'(x,\alpha) = \left\{ \begin{array}{ll} \mathrm{ELU}(x,\alpha) + \alpha & x \leq 0 \\ 1 & x > 0 \end{array} \right\}$$



Kuva 3.7: ELU-funktio ja sen derivaatta eri parametreilla.

ELU-funktio muistuttaa paljon ReLUa ja LeakyReLUa, mutta se sallii positiiviset gradientit negatiivisilla syötteillä. Suurilla negatiivisilla syötteillä sillä on parametrilla säädettävä raja-arvo, joka tekee ELU-funktiosta vakaamman neuroverkon oppimisen kannalta. Lisäksi ELU sallii annetun parametrin α suuruisen gradientin, kun syöte x menee nollaan eli $\mathrm{ELU}'(0,\alpha) = \alpha$, välttäen näin saturoitumista [51, 32]. Kuva 3.7 havainnollistaa ELU:n derivaattafunktion käyttäytymistä.

Kustannusfunktion valintaa rajaa moni tekijä — niin regressio- kuin luokittelutehtäviin on olemassa omat kustannusfunktionsa. Kustannusfunktio voi olla käytännössä mikä tahansa funktio, jolla mitataan joko kahden havainnon välistä etäisyyttä tai samankaltaisuutta/erilaisuutta. Syväoppimisessa se täyttää tyypillisesti metriikan tunnusmerkit eli etäisyys esimerkiksi kahden pisteen tai vektorin välillä on i) aina epänegatiivinen (positiivisesti definiitti), ii) yhtä suuri suunnasta riippumatta (symmetrinen) ja iii) se on kaikista mahdollisista etäisyyksistä lyhin (kolmioepäyhtälö pätee). Taulukkoon 3.1 on listattu syväoppimisessa yleisesti tunnettuja regressio-ongelmien kustannusfunktioita.

Taulukko 3.1: Erilaisia regressio-ongelmien kustannusfunktioita.

Kustannusfunktio	Määritelmä $\mathcal{L}_\theta(\hat{\mathbf{y}}, \mathbf{y})$
Neliösumma	$\frac{1}{n}\sum_{i=1}^{n}(\hat{\mathbf{y}}_i - \mathbf{y}_i)^2$
Absoluuttinen virhe	$\frac{1}{n}\sum_{i=1}^{n}\lvert\hat{\mathbf{y}}_i - \mathbf{y}_i\rvert$
Logaritminen neliösumma	$\frac{1}{n}\sum_{i=1}^{n}(\log(\hat{\mathbf{y}}_i + 1) - \log(\mathbf{y}_i + 1))^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\log(\hat{\mathbf{y}}_i+1)}{\log(\mathbf{y}_i+1)}\right)^2$
Logaritminen hyperbolinen kosini	$\frac{1}{n}\sum_{i=1}^{n}\log(\cosh(\hat{\mathbf{y}}_i - \mathbf{y}_i)) = \frac{1}{n}\sum_{i=1}^{n}\log\left(\frac{1}{2}(e^{\hat{\mathbf{y}}_i-\mathbf{y}_i} + e^{\hat{\mathbf{y}}_i-\mathbf{y}_i})\right)$
Prosentuaalinen virhe	$\frac{1}{n}\sum_{i=1}^{n}\frac{\lvert\mathbf{y}_i-\hat{\mathbf{y}}_i\rvert}{\mathbf{y}_i}100\%$
Kosinimitta	$\frac{\sum_{i=1}^{n}\hat{\mathbf{y}}_i\mathbf{y}_i}{\sqrt{\sum_i^n\hat{\mathbf{y}}_i^2}\sqrt{\sum_i^n\mathbf{y}^2}} = \frac{\hat{\mathbf{y}}\cdot\mathbf{y}}{\lvert\lvert\hat{\mathbf{y}}\rvert\rvert_2\lvert\lvert\mathbf{y}\rvert\rvert_2}$
Kosinietäisyys	$1-$Kosinimitta

Joillakin taulukossa 3.1 esitetyillä kustannusfunktioilla on olemassa neliöjuurellisia versioita, kuten esimerkiksi neliösummalla, jolloin siitä tulee perinteinen Euklidinen etäisyysmitta. Kustannusfunktioilla on erilaisia ominaisuuksia ja puutteita, jotka on syytä ottaa tapauskohtaisesti huomioon. Esimerkiksi absoluuttinen virhe[15] ja erityisesti neliösumma ovat herkkiä datassa esiintyvien muuttujien arvojen heilahteluille. Lisäksi neliösumma on harhaton vain datan ollessa normaalisti jakautunut. Prosentuaalinen virhe on toisaalta intuitiivinen ja helppolukuinen, mutta se on harhainen ennustuksille, jotka ovat arvoltaan pienempiä kuin tavoitearvo. Se ei myöskään ole symmetrinen[16] ja tavoitteen ollessa nolla sen nimittäjä menee nollaksi. Logaritminen neliösumma sen sijaan on symmetrinen ja soveltuu paremmin eri skaaloja sisältävälle raakadatalle [135]. Logaritminen hyperbolinen kosini on neliösummasta sileämpi versio, jolla pyritään välttämään neliösumman ja absoluuttisen virheen puutteita. Kosinimitta mittaa kahden havainnon välistä kulmaa tarkasteltavassa avaruudessa ja kosinietäisyys on siitä metriikan määritelmän täyttävä versio, jonka määrittelyjoukko on $[0, 2]$. Kaiken kaikkiaan ei ole yhtä ja oikeaa kustannus-

[15] Absoluuttinen virhe on käytännössä kahden pisteen välinen lyhin etäisyys, kun sallitaan liikkeet vain vaaka- ja pystysuunnassa. Se tunnetaan taksigeometrian (*eng. taxicab geometry*) metriikkana ja sitä kutsutaan myös Manhattanin etäisyydeksi sekä korttelietäisyydeksi (*eng. city-block*).

[16] Prosentuaalinen virhe ei anna samaa tulosta, vaikka määritelmässä ennustuksen ja tavoitearvon vaihtaisi päittäin. Esimerkissä $\frac{\lvert150-100\rvert}{150} \neq \frac{\lvert100-150\rvert}{100}$, vaikka ennustusten välinen ero on yhtä suuri eli 50 [97].

funktiota neuroverkkojen optimointiin ja mallien suorituskykyjen arviointiin, vaan käytännön sovelluksissa on suositeltavaa käyttää ja vertailla useampaa kuin yhtä.

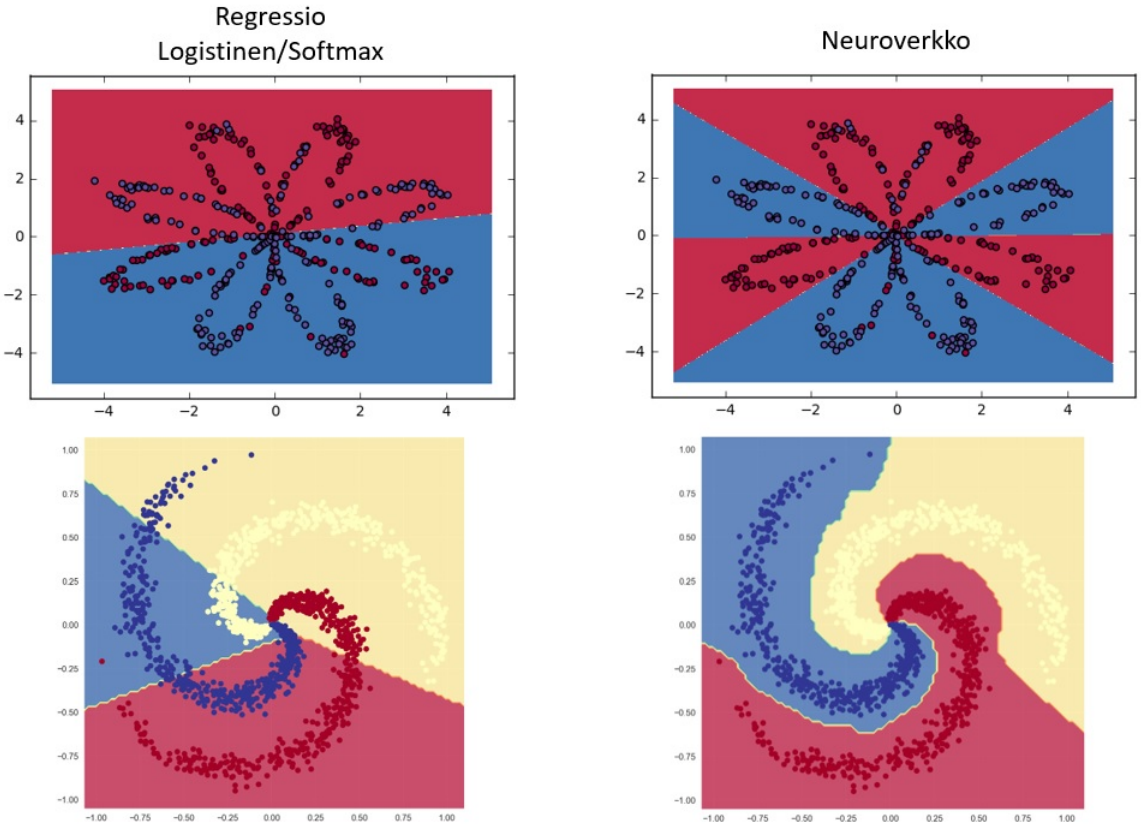
Yhteenvetona voidaan todeta, mitä oppimisella käytännössä tarkoitetaan: se on sellaisen parametriyhdistelmän etsimistä, joka minimoi kustannusfunktion saatavilla olevilla opetusesimerkeillä. Oppiminen tapahtuu prosessoimalla iteratiivisesti pieniä määriä opetusesimerkkejä kerrallaan ja tekemällä toistuvia parametripäivityksiä oppimisnopeuden ja gradienttien avulla, jotka lasketaan neuroverkon parametrien suhteen. Koska neuroverkon voidaan ajatella olevan yksi iso aktivoinneista koostuva yhdistetty funktio, siihen voidaan soveltaa ketjusääntöä, mikä mahdollistaa koko oppimisprosessin [28].

Neuroverkkojen eräs kiehtovimmista ominaisuuksista pohjautuu universaaliin approksimaatioteoriaan (*eng. universal approximation theorem*). Seuraavassa mukaillaan Funahashin [70] laatimaa teoreemaa. Olkoon $\mathbf{x} \in \mathbb{R}^n$ ja $|\mathbf{x}| = \sum_{i=1}^{n} (x_i^2)^{\frac{1}{2}} = ||\mathbf{x}||_2$. Olkoon $K \in \mathbb{R}^n$ kompakti[17] alijoukko ja $\mathcal{F}(\mathbf{x}) \in K$ reaalinen jatkuva funktio. Tällöin $\forall\, \epsilon > 0\ \exists\, N \in \mathbb{N}$ ja parametrit a_i, b_i $(i \in N)$ sekä w_{ij} $(i \in N, j \in n)$ siten, että

$$\max_{\mathbf{x} \in K} |\mathcal{F}(\mathbf{x}) - \hat{\mathcal{F}}(\mathbf{x})| < \epsilon \quad \text{missä} \quad \hat{\mathcal{F}}(\mathbf{x}) = \sum_{i=1}^{N} a_i \sigma\Big(\sum_{j=1}^{n} w_{ij} x_j - b_i \Big). \tag{3.9}$$

Edelliseen perustuen Funahashi yleistää, että mitä tahansa jatkuvaa funktiota ($\mathcal{F} : \mathbb{R}^n \mapsto \mathbb{R}^m$) voidaan approksimoida millä tahansa tarkkuudella (vähintään) yhden piilokerroksen omaavalla neuroverkolla, jossa piilokerroksen aktivointifunktio on jatkuva, rajoitettu ja monotonisesti kasvava sekä ei ole vakio. Neuroverkkojen mallinnuskykyä muutkin samoihin aikoihin tutkineet, esimerkiksi Cybenko [36] ja Hornik [66], ovat tehneet vastaavanlaisia havaintoja. Teorian ongelmana on kuitenkin sen keskittyminen jonkin arkkitehtuurin olemassaoloon eli se ei tarjoa vastauksia esimerkiksi siihen, että miten monta neuronia mahdollistaa tietyn tasoisen approksimaation ja millä parametrein [70, 36, 66]. Universaali approksimaatioteoria toisaalta mahdollistaa esimerkiksi monimutkaiset päätösrajat omaavien luokitteluongelmien ratkaisemisen vaivattomasti neuroverkoilla, mitä kuva 3.8 havainnollistaa [39, 40]. Tämä on pitkälti mahdollista juurikin epälineaaristen aktivointifunktioiden vuoksi.

[17] Kompaktiutta reaaliavaruuksissa käsitellään muun muassa lähteessä [23] ja Aleksi Karhun pro gradu -tutkielmassa [74].



Kuva 3.8: Epälineaaristen luokitteluongelmien päätösrajojen visualisointia.

Kuvan 3.8 ylärivillä on binääriluokitteluongelma, missä siniset ja punaiset datapisteet ovat kerääntyneet kukkamaiseen kahdeksan terälehden omaavaan muotoon. Vasemmalla puolella logistinen regressiomalli luokittelee alle puolet pisteistä oikein, kun taas oikean puolen yhdestä piilokerroksesta ja neljästä neuronista koostuva neuroverkko saa oikein 90 prosenttia. Koska datajoukko ei ole lineaarisesti jaettavissa (*eng. linearly separable*), logistinen regressio ei pärjää luokitteluongelmassa. Neuroverkko sen sijaan kykenee hahmottamaan terälehdet toisistaan ja määrittämään epälineaariset päätösrajat. Kuvan 3.8 alarivillä on eräs kolmen luokan luokitteluongelma, missä datapisteet ovat järjestyneet eräänlaisiksi spiraaleiksi värien kuvaamalla tavalla. Vasemman puoleisessa kuvaajassa logistisen regression moniluokkaisella versiolla softmax[18] regressiolla ei ole kykyä määrittää epälineaarisia

[18]Logistinen regressio muuttuu nyt siten, että yhden neuronin sijasta käytetään kolmea neuronia, joiden ulostulot saadaan softmax-funktion eli normalisoidun eksponentiaalisen funktion kautta. Softmax-funktio määritellään $\text{softmax}(a_i) = \frac{e^{a_i}}{\sum_{n=1}^{N} e^{a_i}}$. Toisin sanoen softmax-funktio normalisoi painotetut summat niin, että ne edustavat todennäköisyyksiä ja niiden summa on 1 [12].

päätösrajoja, jolloin mallilta menee puolet luokitteluista väärin. Oikean puoleisessa kuvassa sen sijaan hieman suurempi neuroverkko saa epälineaarisilla päätösrajoilla yli 99 prosenttia oikein.

Keinotekoisten neuroverkkojen vahvuus ja suorituskyky erilaisissa sovelluksissa perustellaan usein kuitenkin niiden syvyydellä. Kasvattamalla piilokerrosten määrää keinotekoisten neuroverkkojen mallinnuskyvyn katsotaan paranevan merkittävästi ja tällöin tarvitaan myös huomattavasti vähemmän parametreja verrattuna neuronien määrän kasvattamiseen yhden piilokerroksen neuroverkossa. Piilokerrosten sanotaan myös mahdollistavan hierarkkisen tavan prosessoida jotakin tehtävää: kukin kerros keskittyy johonkin tiettyyn osaan tehtävästä ja mitä syvemmälle mennään, niin sitä abstraktimpiin ja kokonaisvaltaisimpiin osiin neuroverkko keskittyy ja myöskin sitä kompleksisempia prosesseja voidaan mallintaa [65, 91, 113]. Empiiristen tutkimusten lisäksi myös teoreettiset tutkimukset puoltavat syvyyden edistävää vaikutusta neuroverkkojen mallintamiskykyyn: Bengio [8] esitti, että syvyyden $k - 1$ omaava neuroverkkoarkkitehtuuri saattaa vaatia eksponentiaalisen määrän laskentayksiköitä syvyyden k omaavaan arkkitehtuuriin verrattuna. Myös Cohen ym. [34] mukaan[19] piilokerrosten vähentämisestä syvästä neuroverkosta seuraa eksponentiaalinen kasvu neuronien määrässä mikäli alkuperäinen mallintamiskyky halutaan säilyttää. Lisäksi muun muassa Telgarsky [134] tekee vastaavanlaisia havaintoja.

3.2 Toistuvat neuroverkot

Edellä esitetyllä eteenpäin syöttävällä neuroverkolla on muutamia heikkouksia. Se prosessoi vain ennalta määrätyn kokoisia syötteitä tai havaintoja eli $\mathbf{x} \in \mathbb{R}^n$, missä n on vakio [84]. Sillä on myöskin vaikeuksia tunnistaa havaintojen välisiä yhteyksiä tai ylläpitää tietoa niiden järjestyksestä, ja verkon ollessa täysin yhdistetty siinä on paljon kerroskohtaisia parametreja. Kun eteenpäin syöttävään neuroverkkoon sallitaan silmukkarakenteet, tietoa voidaan siirtää ja käyttää uudelleen arkkitehtuurin eri osissa. Tällä pienellä muutoksella eteenpäin syöttävästä neuroverkosta saadaan luotua uusi luokka neuroverkkoarkkitehtuureja, joita kutsutaan toistuviksi tai takaisinkytkeytyviksi neuroverkoiksi (*eng. recurrent neural network, RNN*). Ne pystyvät

[19] Artikkelin liitteessä D on kattava katsaus teoreettisista tutkimuksista syvyyden vaikutuksesta neuroverkkojen mallinnuskykyyn.

käsittelemään vaihtelevan pituisia sekvenssimuotoisia[20] syötteitä eli $\mathbf{x} = \{\mathbf{x}_t\}_{t=1}^{T}$ ja $\mathbf{x}_t \in \mathbb{R}^m$, missä m on muuttuja. Ne pystyvät myös tuottamaan vaihtelevan pituisia ulostuloja eli $\hat{\mathbf{y}} = \{\hat{\mathbf{y}}_t\}_{t=1}^{T}$ ja $\hat{\mathbf{y}}_t \in \mathbb{R}^p$, missä p on muuttuja. Lisäksi eteenpäin syöttävän neuroverkon kyetessä luomaan kuvauksia vain vastaavien syötteiden ja ulostulojen välille, toistuvat neuroverkot pystyvät periaatteessa hyödyntämään koko syötehistorian kullekin ulostulolle[21]. Sanotaankin, että toistuvilla neuroverkoilla on eräänlainen ajan mittaan kehittyvä muisti, jonka avulla ne oppivat kontekstiriippuvaisia merkityksiä syötteille ja syötteistä opituille piirteille [55, 64]. Niillä voidaan mallintaa sellaisia piirteiden välisiä keskinäisiä suhteita, jotka ovat temporaalisia eli aikaan sidottuja.

Toistuvassa neuroverkossa kullakin syötteen saapumisajanhetkellä $\mathbf{x}_t$ lasketaan rekursiivisesti sisäiset tilat $\mathbf{h}_t$ (*eng. hidden state*), jotka riippuvat syötteestä $\mathbf{x}_t$, vakiotermeistä $\mathbf{b}_h$ ja edellisellä askeleella lasketuista tiloista $\mathbf{h}_{t-1}$

$$\mathbf{h}_t = \tanh(\mathbf{W}_{h,h}\mathbf{h}_{t-1} + \mathbf{W}_{h,x}\mathbf{x}_t + \mathbf{b}_h) \tag{3.10}$$

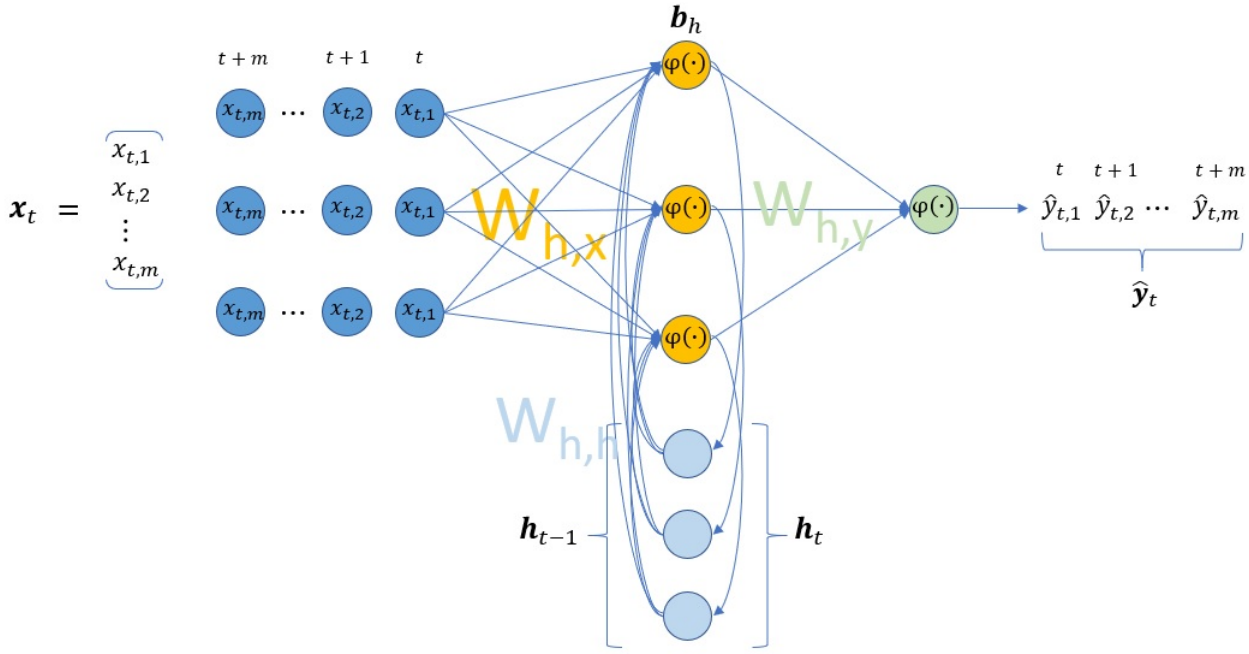
$$\hat{\mathbf{y}}_t = \mathbf{W}_{h,y}\mathbf{h}_t, \tag{3.11}$$

missä niin tiloja, syötettä kuin ulostuloja eli verkon antamaa ennustetta vastaa omat painokertoimet[22]. Sisäisten tilojen avulla verkko siis siirtää tietoa yhdeltä ajan hetkeltä toiselle. Oletetaan, että $\mathbf{x} = \{\mathbf{x}_t\}_{t=1}^{T}$ ja $\mathbf{x}_t \in \mathbb{R}^m$, missä m voi vaihdella. Kuva 3.9 havainnollistaa yhden sekvenssin $\mathbf{x}_t$ tapauksessa yksinkertaista yhden piilokerroksen omaavaa eteenpäin syöttävää neuroverkkoa, joka on muutettu toistuvaksi arkkitehtuuriksi.

[20] Sekvenssimuotoista dataa esiintyy muun muassa teksteissä, puheessa, videoissa, genetiikassa, aikasarjoissa ja signaaleissa, kuten bittijonoissa. Ei-sekvenssimuotoista dataa voidaan myös muuttaa sekvenssimuotoon, kuten vektorisoimalla esimerkiksi matriisimuotoon koottuja pikseliesityksiä digitaalisista kuvista.

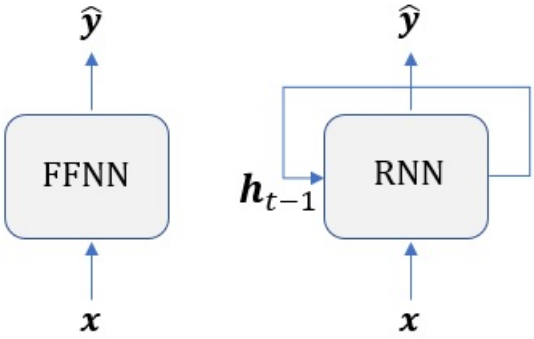
[21] Toistuville neuroverkoille voidaan myös johtaa universaali approksimaatioteoria, jonka mukaan riittävällä määrällä piilokerroksen laskentayksiköitä voidaan approksimoida mitä tahansa funktiota, joka on kuvaus sekvenssistä sekvenssiin, millä tahansa tarkkuudella [60, 55].

[22] Toistuvissa neuroverkoissa käytetään usein hyperbolista tangettia tilojen aktivointifunktiona, koska sen määrittelyväli [-1,1] mahdollistaa negatiiviset arvot ja se on symmetrinen nollan ympärillä. Näin tilojen arvot pystyvät laskemaan ja kasvamaan tarkoituksenmukaisesti.



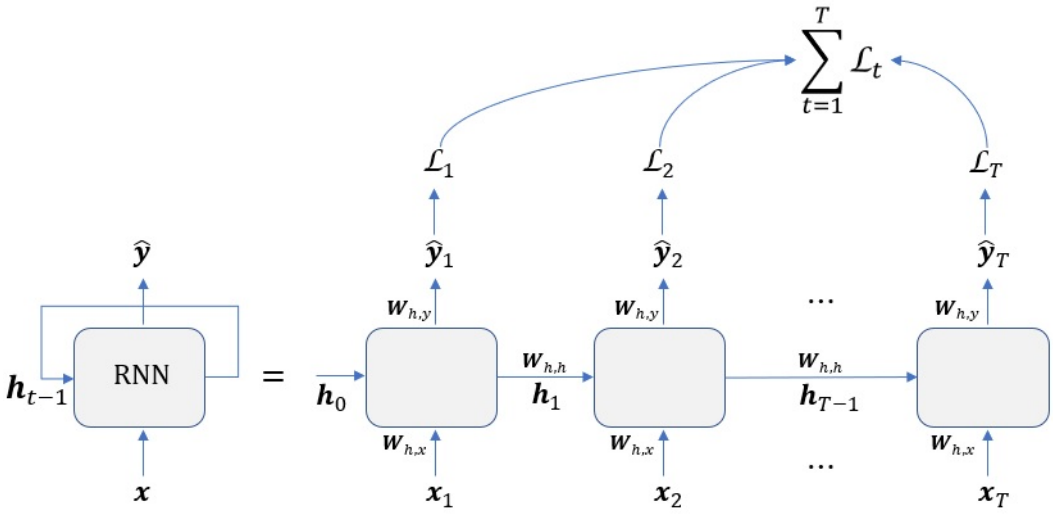
Kuva 3.9: Havainnollistava esimerkki yhden piilokerroksen omaavasta toistuvasta neuroverkosta.

Kuvan 3.9 arkkitehtuuri eroaa eteenpäin syöttävästä neuroverkosta siten, että kutakin piilokerroksen neuronia kohti tuodaan kuvaannollisesti oma apuyksikkönsä tai kontekstinoodinsa (*eng. context node/unit*). Näihin kontekstinoodeihin varastoidaan kullakin askeleella kussakin neuronissa lasketut aktivaatiot eli sisäiset tilat, jotka seuraavalla aika-askeleella tulevan syötteen yhteydessä otetaan mukaan neuroneihin. Eli nyt aktivaatioiden laskentaan vaikuttaa myös edellisellä aika-askeleella lasketut sisäiset tilat painokertoimineen, kun eteenpäin syöttävässä neuroverkossa käsiteltiin vain syötteitä ja niiden painokertoimia. Huomaa, että kukin piilokerroksen neuroni vastaanottaa saman syötteen sekvenssistä eli neuronit käsittelevät syötettä rinnakkain ja sekvenssistä otetaan vuorollaan uusi syöte prosessoitavaksi. Kullakin kontekstinoodilla on oma painokertoimensa kuhunkin piilokerroksen neuroniin eli ne ovat myöskin täysin yhdistettyjä [98, 51]. Ajatellaan, että kuvan 3.9 piilokerros esitellään yhtenä abstraktina kokonaisuutena kun tiedetään, että kullakin piilokerroksella on vakiotermien lisäksi syöte- ja tilakohtaiset painokertoimet. Tällöin kuva 3.10 havainnollistaa abstraktisti eteenpäin syöttävän neuroverkon ja toistuvan neuroverkon periaatteellista eroa yhden piilokerroksen tapauksessa, missä neuronien määrää ei ole rajattu.



Kuva 3.10: Abstraktiot eteenpäin syöttävästä (vas.) ja toistuvasta neuroverkosta (oik.).

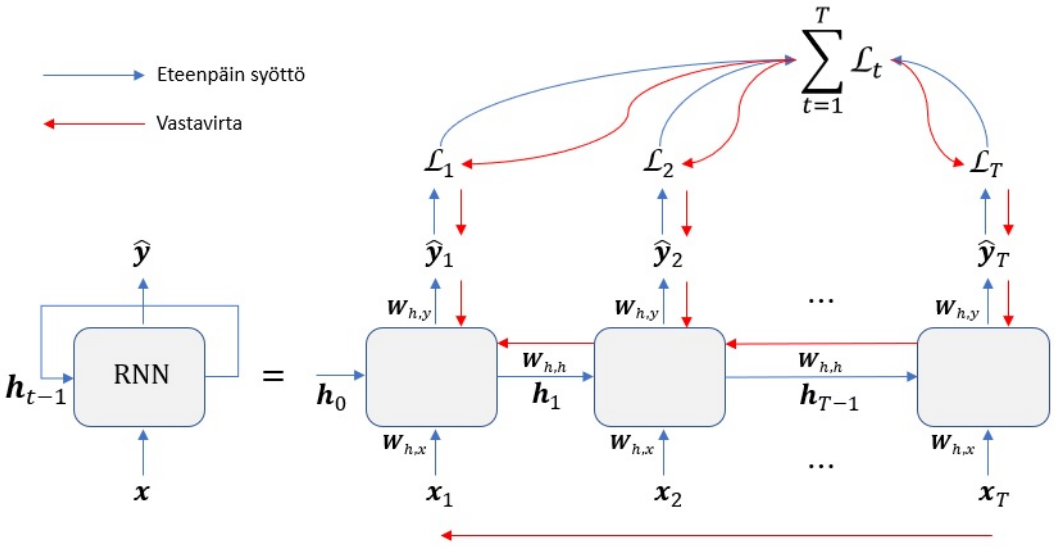
Toistuvan neuroverkon rekursiivista luonnetta voidaan hyödyntää verkon rakenteen ymmärtämisessä ja havainnollistamisessa siten, että rekursio avataan auki ajan suhteen, jolloin silmukat poistuvat. Näin ollen samasta neuroverkosta tehdään käytännössä useampi kopio, jossa informaatiota kulkee kopiolta toiselle, kuten kuvassa 3.11 [132].



Kuva 3.11: Toistuvan neuroverkon toimintatapa.

Nyt muodostunut rakenne muistuttaa paljon eteenpäin syöttävää neuroverkkoa, mutta eri ajan hetkellä saapuvat sekvenssit käyttävät samoja painokertoimia, eli parametrit ovat jaettuja verkon eri osissa. Kuvaan on myös lisätty kullakin ajan het-

kellä tehdystä ennusteesta laskettu virhe, joiden avulla kokonaisvirhe saadaan muodostettua[23]. Tämä toimii perustana verkon optimoinnille, joka on muunnelma aiemmin esitellystä vastavirta-algoritmista ja jota kutsutaan aikakeskeiseksi vastavirta-algoritmiksi (*eng. backpropagation through time, BPTT*) [64]. Tässä tapauksessa sen sijaan, että kokonaisvirhettä (ja siten gradienttia) lasketaan yhden eteenpäin syöttävän verkon suhteen yksittäisellä ajan hetkellä, niin virheet lasketaan joka ajan hetkellä kustakin kopiosta. Lisäksi, nyt toistuvien neuroverkkojen tapauksessa tulee huomioida yksi joukko parametreja lisää, eli sisäisiä tiloja vastaavat painokertoimet. Näin ollen lasketaan myös joukko osittaisderivaattoja yli aika-askelten[24]. Kuvassa 3.12 havainnollistetaan prosessia [132].



Kuva 3.12: Aikakeskeinen vastavirta-algoritmi.

On myös mahdollista, että toistuvassa neuroverkossa on useampi piilokerros. Tällöin esimerkiksi kuvan 3.9 arkkitehtuuriin voidaan lisätä omat painokertoimet ja kontekstinoodit omaava piilokerros, joiden neuroneiden syötteenä on ensimmäisen piilokerroksen sisäiset tilat $\mathbf{h}_t$. Lisäksi kuvassa 3.11 esiintyvien laskentayksiköiden

[23]Kaikkia ulostuloja $\{\hat{\mathbf{y}}_t\}_{t=1}^{T}$ ei välttämättä tarvitse huomioida, vaan niistä voidaan ottaa osajoukko tai vain viimeinen ulostulo. Näin onnistuu esimerkiksi yhden ennustuksen tekeminen useampaan sekvenssiin perustuen. Sama onnistuu myöskin sekvensseille: yhtä sekvenssiä (toistuvasti) prosessoimalla voidaan tehdä useampi ennustus [51].

[24]BPTT käyttää paljon muistia pitääkseen yllä kunkin ajanhetken sisäisiä tiloja osittaisderivaattojen laskemiseksi. Gruslys ym. [58] esittelevät tehokkaamman BPTT algoritmin, joka hyödyntää dynaamista optimointia määrittäessään varastoinnin ja uudelleenlaskennan tarvetta.

yläpuolelle muodostuu kutakin piilokerrosta kohti uusi rivi vastaavia laskentayksiköitä omine painokertoimineen ja sisäisine tiloineen, joihin edeltävän rivin sisäiset tilat $\{\mathbf{h}_t\}_{t=1}^{T}$ ohjataan ulostulojen sijaan. Viimeisen piilokerroksen jälkeen lasketaan ulostulot $\{\hat{\mathbf{y}}_t\}_{t=1}^{T}$ tuttuun tapaan. [51].

Ongelmia saattaa aiheuttaa kuitenkin se, että aika-askelten yli tehtävässä vastavirtauksessa eli laskettaessa kokonaisvirheen gradienttia verkon ensimmäisen sisäisen tilan $\mathbf{h}_0$ suhteen, käytetään ketjusäännön myötä toistuvasti matriisioperaatioita ja epälineaarista aktivointifunktiota. Nyt, jos laskennan aikana syntyy paljon arvoja, jotka ovat suurempia kuin 1, saattavat gradientit räjähtää eksponentiaalisesti käsiin (*eng. exploding gradients*) ja vastaavasti jos arvot saavat pienempiä arvoja kuin 1, puhutaan häviävistä gradienteista (*eng. vanishing gradients*). Nämä ongelmat[25] vaikeuttavat verkon optimointia ja siten oppimisprosessia. Häviävät gradientit ovat erityisen ongelmallisia siinä mielessä, että tällöin toistuva neuroverkko ei gradienttien pienentyessä pysty kaappaamaan havaintojen välisiä suhteita pitkällä aikavälillä [132]. Tätä voi kutsua lyhytkestoiseksi työmuistiksi. Näin ollen on kehitetty erilaisia "pitkäkestoisen työmuistin" omaavia toistuvia neuroverkkoarkkitehtuureja, jotka ovat kontrolloivimpia tiedonkulun suhteen ja sisältävät useampia aktivointifunktioita omaavia laskentayksiköitä[26]. Eräs tunnetuimmista on LSTM (*eng. long short-term memory*), johon tutustutaan seuraavaksi lähteitä [46, 108] mukaillen.

LSTM on toistuvien neuroverkkojen variantti, jonka toiminta perustuu muistilohkoihin (*eng. memory cell*). Niiden sisällä on porttimekanismit (*eng. gate*), joilla se pystyy määrittelemään, missä määrin tietoa pidetään mukana kullakin ajan hetkellä ja missä määrin tietoa vastaavasti hylätään tai unohdetaan. Nyt sisäisen tilan lisäksi muistilohkoissa on olemassa erillinen lohkotila (*eng. cell state*), joka on myös lohkon sisäinen. Ensimmäiseksi LSTM-lohko määrittää, että miten paljon tietoa tulisi poistaa sen edellisistä lohkotiloista $\mathbf{s}_{t-1}$ hyödyntämällä unohtamisportissa (*eng. forget gate*) $\mathbf{f}_t$ laskettuja nykyiseen syötteeseen $\mathbf{x}_t$ ja edellisiin tiloihin $\mathbf{h}_{t-1}$ perustuvia

[25] Räjähtäviä gradientteja voidaan ehkäistä yksinkertaisesti määrittelemällä maksimiarvo, jonka gradientit voivat saada (*eng. gradient clipping*). Häviäviä gradientteja voidaan välttää valitsemalla eri aktivointifunktioita, alustamalla painot eri tavalla tai muuttamalla neuroverkkoarkkitehtuuria itsessään.

[26] Kuten mainittua, auki avattu RNN muistuttaa syvää eteenpäin syöttävää neuroverkkoa. RNN ei kuitenkaan tee vastaavan tyyppistä hierarkista prosessointia, vaan historiallinen data prosessoidaan useampaan kertaan rekursion avulla. Lisäksi tiettyjä siirtymisiä kuten tilan päivitystä erikseen tarkastelemalla huomataan, että jotkin operaatiot eivät itsessään ole syviä vaan pinnallisia [65, 112].

aktivointeja

$$\mathbf{f}_t = \sigma(\mathbf{W}_{f,x}\mathbf{x}_t + \mathbf{W}_{f,h}\mathbf{h}_{t-1} + \mathbf{b}_f). \tag{3.12}$$

Lisäksi syöteportissa (*eng. input gate*) $\mathbf{i}_t$ lasketaan miten paljon tietoa tulisi sisällyttää uusiin lohkotiloihin $\mathbf{s}_t$ kandidaatti- eli ehdokasarvoista $\tilde{\mathbf{s}}_t$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{i,x}\mathbf{x}_t + \mathbf{W}_{i,h}\mathbf{h}_{t-1} + \mathbf{b}_i) \tag{3.13}$$

$$\widetilde{\mathbf{s}}_t = \tanh(\mathbf{W}_{\tilde{s},x}\mathbf{x}_t + \mathbf{W}_{\tilde{s},h}\mathbf{h}_{t-1} + \mathbf{b}_{\tilde{s}}). \tag{3.14}$$

Huomaa, että unohtamis- ja syöteporttien käyttämä aktivointifunktio (sigmoid) skaalaa tulokset välille [0,1], joka ilmaisee kertoimien muodossa miten paljosta tiedosta tulisi pitää kiinni. Näillä tiedoilla saadaan laskettua uudet lohkotilat $\mathbf{s}_t$

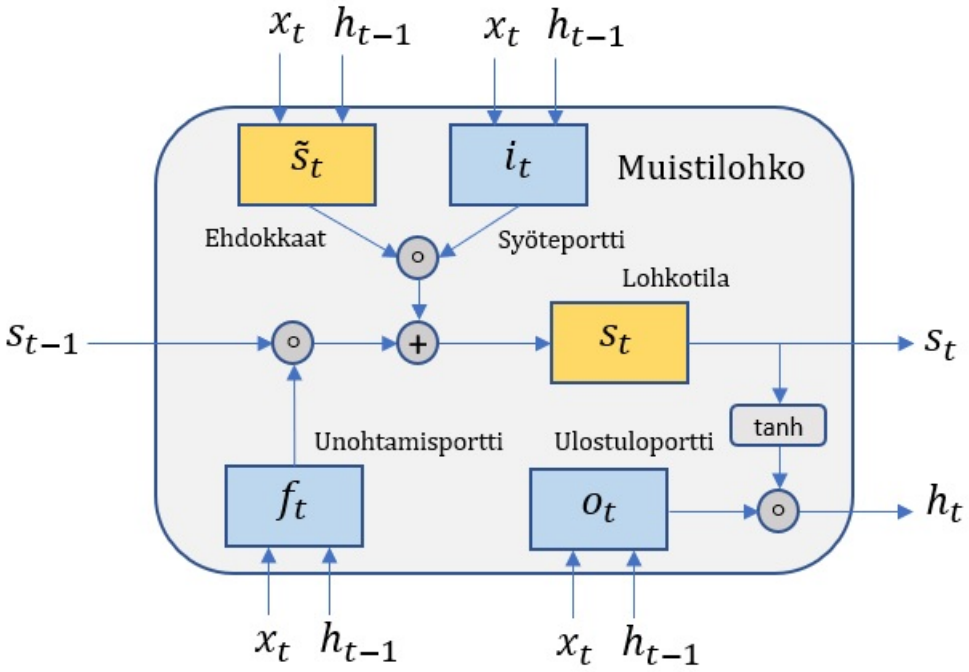
$$\mathbf{s}_t = \mathbf{f}_t \odot \mathbf{s}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{s}}_t, \tag{3.15}$$

missä $\odot$ edustaa Hadamardin eli alkioittaista tuloa. Lohkotilan selvittämisen jälkeen saadaan määriteltyä lohkon uudet sisäiset tilat $\mathbf{h}_t$, joiden laskemiseen tarvitaan lohkotilojen lisäksi nykyiseen syötteeseen ja edellisiin sisäisiin tiloihin perustuvan ulostuloportin (*eng. output gate*) $\mathbf{o}_t$ aktivointeja

$$\mathbf{o}_t = \sigma(\mathbf{W}_{o,x}\mathbf{x}_t + \mathbf{W}_{o,h}\mathbf{h}_{t-1} + \mathbf{b}_o) \tag{3.16}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{s}_t). \tag{3.17}$$

Huomaa, että kullekin portille ja ehdokasarvoille on omat optimoitavat painokertoimensa ja vakioterminsä. Edellä esitettyä dynaamista tiedonkulkua voidaan havainnollistaa kuvan 3.13 mukaisesti [46].



Kuva 3.13: LSTM-muistilohko.

Huomaa, että lohkotiloja $\mathbf{s}_t$ käytetään vain muistilohkon sisäisissä operaatioissa, kun taas lohkon sisäiset tilat $\mathbf{h}_t$ edustavat samalla lohkon ulostuloja $\tilde{\mathbf{y}}_t$. Lohkon sisäisistä tiloista erillään olevien lohkotilojen $\mathbf{s_t}$ laskennasta voidaan huomata, miten LSTM-lohko pyrkii välttämään häviäviä gradientteja. Tämä ilmenee erityisesti tilanteessa, jossa LSTM-lohkon rekursiivisuus avataan ajan suhteen: kun aikakeskeistä vastavirta-algoritmia sovelletaan, lohkotilan suhteen lasketun gradientin kohdalla ei tarvita matriisioperaatioita ja aktivointifunktioita [71]. LSTM-lohkoja voi yhdellä piilokerroksella olla useampia ja syviä arkkitehtuureja voidaan edelleen muodostaa lisäämällä piilokerrosten määrää. Syvät toistuvat arkkitehtuurit saattavat mahdollistaa temporaalisen hierarkkisen prosessoinnin, jolloin mallin sisäiset tilat voivat tunnistaa erilaisia riippuvuussuhteita eri aikaväleillä samanaikaisesti [113].

Kirjallisuudessa on eniten käytetty erästä muokattua versiota edellä esitetystä LSTM:stä [57]. Sen ero perinteiseen LSTM:ään on painotettujen "tirkistysyhteyksien" (*eng. peephole connections*) lisääminen, mikä tarkoittaa edellisten lohkotilojen lisäämistä syöte- ja unohtamisportin yhtälöihin sekä päivitettyjen lohkotilojen lisäämistä ulostuloportin yhtälöön. Näin tietoa lohkotiloista virtaa muistilohkon eri osissa tehden siitä tilannetietoisemman ja yhdistetymmän, mikä taas parantaa verkon suorituskykyä [52]. Täten edellä esitetyt kaavat saadaan Gravesin [56] notaatiota

seuraten muotoon

$$\mathbf{f}_t = \sigma(\mathbf{W}_{f,x}\mathbf{x}_t + \mathbf{W}_{f,h}\mathbf{h}_{t-1} + \mathbf{W}_{f,s}\mathbf{s}_{t-1} + \mathbf{b}_f) \tag{3.18}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{i,x}\mathbf{x}_t + \mathbf{W}_{i,h}\mathbf{h}_{t-1} + \mathbf{W}_{i,s}\mathbf{s}_{t-1} + \mathbf{b}_i) \tag{3.19}$$

$$\widetilde{\mathbf{s}}_t = \tanh(\mathbf{W}_{\tilde{s},x}\mathbf{x}_t + \mathbf{W}_{\tilde{s},h}\mathbf{h}_{t-1} + \mathbf{b}_{\tilde{s}}) \tag{3.20}$$

$$\mathbf{s}_t = \mathbf{f}_t \odot \mathbf{s}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{s}}_t \tag{3.21}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{o,x}\mathbf{x}_t + \mathbf{W}_{o,h}\mathbf{h}_{t-1} + \mathbf{W}_{o,s}\mathbf{s}_t + \mathbf{b}_o) \tag{3.22}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{s}_t). \tag{3.23}$$

Eräs toinen tunnettu muunnelma LSTM:stä on GRU (*eng. gated recurrent unit*), jossa on vähemmän porttimekanismeja ja siinä ei ole "tirkistysyhteyksiä". Tässä tapauksessa $\mathbf{f}_t$ ajatellaan nollaus- tai resetointiporttina (*eng. reset gate*) ja $\mathbf{i}_t$ päivitysporttina (*eng. update gate*). GRU:ssa ei ole myöskään erillisiä lohkotiloja ja lohkon sisäisiä tiloja, vaan ne yhdistyvät. Perinteinen LSTM saa nyt muodon

$$\mathbf{f}_t = \sigma(\mathbf{W}_{f,x}\mathbf{x}_t + \mathbf{W}_{f,h}\mathbf{h}_{t-1} + \mathbf{b}_f) \tag{3.24}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{i,x}\mathbf{x}_t + \mathbf{W}_{i,h}\mathbf{h}_{t-1} + \mathbf{b}_i) \tag{3.25}$$

$$\widetilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{\tilde{h},x}\mathbf{x}_t + \mathbf{W}_{\tilde{h},h}(\mathbf{f}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_{\tilde{h}}) \tag{3.26}$$

$$\mathbf{h}_t = \mathbf{i}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{i}_t) \odot \widetilde{\mathbf{h}}_t, \tag{3.27}$$

missä päivitysportti määrittää miten paljon edellisistä tiloista tuodaan tietoa nykyisiin tiloihin ja kun resetointiportti saa arvoja lähellä nollaa, nykyiset tilat päivittyvät enimmäkseen vain syötteen perusteella [27]. Kirjallisuudessa on esiintynyt monia LSTM:n variantteja, mutta hiljattain on tutkittu ja osoitettu, että ne eivät tuo merkittäviä suorituskykyparannuksia "tirkistysyhteyksien" omaavaan LSTM:ään verrattuna erilaisissa luonnollisen kielen prosessoinnin tehtävissä [57].

3.3 Konvoluutioneuroverkot

Edellä nähtiin, että toistuvilla neuroverkoilla on edullisia ominaisuuksia sellaisten ilmiöiden mallintamiseen, jotka tuottavat sekvenssimuotoista dataa ja missä temporaalisuudella voi olla tärkeä rooli piirteiden välisiä suhteita tarkastellessa. Monissa tapauksissa datassa voi kuitenkin esiintyä myös rakenteita, joissa esimerkiksi muuttujien ja arvojen sijainnilla on merkitystä koko dataan nähden. Tällöin piirteiden keskinäiset suhteet ovat paikkaan sidottuja eli spatiaalisia. Esimerkiksi puheessa ilmenevien äänteiden ja tekstissä esiintyvien kirjainten sijainti on tärkeää, koska

yhdessä ne muodostavat sanoja ja lauseita. Kuvamuotoisessa datassa tietyssä kohdassa sijaitsevat pikselit, jotka piirtävät jotakin tiettyä muotoa, ovat voimakkaammin korreloituneita [92, 91]. Lisäksi graafeista johdetuissa vierusmatriiseissa rivien ja sarakkeiden indeksoinnilla sekä alkioiden sijainnilla on perustavanlaatuinen merkitys.

Tarkastellaan seuraavaksi kuvamuotoisen datan mallintamista. Oletetaan mustavalkoinen kuva, joka voidaan esittää funktiona kaksiulotteisella tasolla ruudukossa, jonka kukin elementti edustaa kokonaislukuna värin voimakkuutta kussakin pikselissä. Tällöin kukin elementti on myös muuttuja. Kuvat ovat pikselikooltaan tyypillisesti suuria, jolloin muuttujien määrä kasvaa nopeasti[27]. Eteenpäin syöttävällä ja toistuvalla neuroverkolla mallinnettaessa tämän tyyppistä kuvaa, tulee kaksiulotteinen ruudukko tai matriisi ensin vektorisoida yksiulotteiseksi syötteeksi. Tällöin alkuperäisen konstruktion sisältämä spatiaalinen informaatio häviää [129]. Monissa tapauksissa, kuten kuvamuotoisessa datassa, sama piirre voi esiintyä monessa eri paikassa ja piirteet tulee myös pystyä tunnistamaan kohinan ja erilaisten vääristymien läsnä ollessa [91, 150]. Lisäksi edellä mainitut neuroverkkoarkkitehtuurit eivät skaalaudu parametrien puitteissa tehokkaasti muuttujien määrien kasvaessa suuriksi. Näin ollen tarvitaan menetelmä, joka soveltuu erityisesti moniulotteisen datan mallintamiseen ja pystyy hyödyntämään datassa esiintyviä luontaisia rakenteita sellaisenaan.

Konvoluutioneuroverkot (*eng. convolutional neural network, CCN*) ovat neuroverkkoarkkitehtuuri, jotka perustuvat eteenpäin syöttävään neuroverkkoon ja on suunniteltu vastaamaan yllä esitettyihin haasteisiin. Tässä tapauksessa piilokerrosten aktivoinnit eivät ota syötteenään kaikkia edellisen kerroksen aktivointeja, vaan tietyn osan niistä. Tällöin verkko ei ole täysin yhdistetty eteenpäin syöttävän neuroverkon tapaan vaan paikoitellen eli lokaalisti yhdistetty. Toinen merkittävä tekijä toistuvien neuroverkkojen lailla on painokertoimien jakaminen verkon eri osissa [90]. Näiden ominaisuuksien myötä voidaan tarkastella konvoluutionaalisen kerroksen (*eng. convolutional layer*) toimintaa, joka on konvoluutioneuroverkkojen perusyksikkö.

Konvoluutionaalisissa kerroksissa tapahtuva laskenta perustuu nimensä mukaisesti konvoluutioon (*eng. convolution*). Neuroverkkokirjallisuudessa konvoluutiota käytetään synonyyminä ristikorrelaatiolle (*eng. cross-correlation*), koska monissa ko-

[27]Esimerkiksi pikselikoon 1280x720 omaavan kuvan muuttujien määrä on 921600 ($\approx 10^6$). Värillisen kuvan tapauksessa kukin RGB-värimallin kanava tulee ottaa huomioon, jolloin muuttujien määrä edelleen kolminkertaistuu.

neoppimiskirjastoissa konvoluutio on implementoitu käytännössä ristikorrelaationa [54]. Vaikkakin määritelmiltään hyvin saman kaltaisia, teoreettisessa ja erityisesti käyttötarkoituksellisessa mielessä konvoluutio ja ristikorrelaatio eroavat kuitenkin hieman toisistaan: esimerkiksi signaalinkäsittelyssä ristikorrelaatiolla mitataan kahden signaalin samankaltaisuutta kun taas konvoluutiolla pyritään tuottamaan usein uusi suodatettu signaali kahdesta alkuperäissignaalista. Tässä mielessä konvoluutioneuroverkkojen "konvoluutio" on juurikin ristikorrelaatiota, missä intuitiona on iteratiivisesti etsiä sellaisia esiintymiä tutkittavasta kuvasta, jotka muistuttavat jotakin piirrettä (*eng. template matching*).

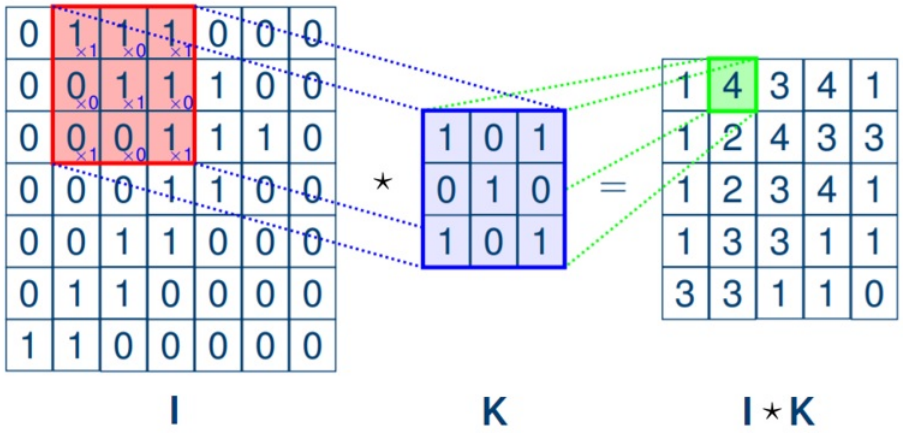
Kaksiulotteisessa tapauksessa konvoluutionaalisissa kerroksissa esiintyvä diskreetti ristikorrelaatio eli konvoluutio määritellään[28]

$$(\mathcal{I} \star \mathcal{K})(i,j) = \sum_m \sum_n \mathcal{I}(i+m, j+n)\mathcal{K}(m,n), \tag{3.28}$$

missä syöte $\mathcal{I}$ on kuvaa edustava kokonaislukuja sisältävä kaksiulotteinen matriisi ja $\mathcal{K}$ on kaksiulotteinen kokonaislukuja sisältävä suodin tai ydin (*eng. filter/kernel*), joka on kooltaan merkittävästi pienempi kuin $\mathcal{I}$. Yllä oleva lauseke kuvastaa ristikorrelaation toista nimeä eli liukuvaa pistetuloa: matriisin $\mathcal{I}$ vasemmasta ylälaidasta lähtien suodinta $\mathcal{K}$ liikutetaan askel kerrallaan matriisin oikeaan laitaan asti, jonka jälkeen siirrytään pykälä alaspäin ja aloitetaan vasemmasta reunasta uudestaan. Kullakin askeleella suotimen sisältämät arvot kerrotaan niillä vastaavilla matriisin arvoilla elementtikohtaisesti, joiden ylle suodin sinä hetkenä laskeutuu. Kunkin askeleen lopuksi saadut arvot lasketaan yhteen ja tulos otetaan talteen uuteen matriisiin. Konvoluution tuloksena syntynyttä matriisia kutsutaan piirrekartaksi (*eng. feature map*). Kuva 3.14 havainnollistaa[29] tätä prosessia kaksiulotteisella syötteellä ja suotimella sekä yhden askeleen liu'uilla: [141]:

[28] Yksiulotteisessa tapauksessa konvoluutio yksinkertaistuu muotoon $(\mathcal{I} \star \mathcal{K})(i) = \sum_m \mathcal{I}(i+m)\mathcal{K}(m)$.

[29] Yksiulotteisessa konvoluutiossa yksiulotteinen suodin liu'utetaan käytännössä vain ensimmäisen rivin läpi.



Kuva 3.14: Yksinkertainen esimerkki konvoluutiosta kaksiulotteisessa tapauksessa.

Muodostuvan piirrekartan alkion arvo muuttuu sen mukaan, mitä saman kaltaisempi suodin $\mathcal{K}$ on kyseessä olevalla askeleella kuvan $\mathcal{I}$ kanssa. Lisäksi konvoluution tuloksena syntyneen piirrekartan koko pienenee alkuperäiseen kuvaan verrattuna: piirrekartan koko riippuu muun muassa käytetyn suotimen koosta ja askelpituuksista[30]. Huomaa myös, että kullakin askeleella tehtävä laskenta on käytännössä sama kuin eteenpäin syöttävien neuroverkkojen yhteydessä esitetyssä neuronissa eli tarkastelun alla olevasta kuvan osasta tai osajoukosta (*eng. patch*) lasketaan painotettu summa suotimen arvojen avulla. Analogian viimeistelemiseksi saadut summat käytetään vielä epälineaarisen aktivointifunktion läpi. Käytännössä suotimien sisältämät arvot alustetaan satunnaisesti ja konvoluutioneuroverkon tulee ne oppia verkkoa optimoitaessa. Siten suotimen sisältämiä arvoja kutsutaan myös verkon painoiksi. Konvoluutioneuroverkoissa käytetään yleensä useampia toisistaan poikkeavia suotimia, jolloin syntyy keskenään erilaisia piirrekarttoja.

Konvoluutionaalisen kerroksen lisäksi CNN:issä käytetään vahvasti myös alinäytteistystä (*eng. downsampling*), joka mielletään omana kerroksena ja jota käytetään tyypillisesti konvoluutionaalisen kerroksen jälkeen. Alinäytteistys voidaan tehdä yksinkertaisen epälineaarisen operaation kautta, jolla kootaan (*eng. pooling*) piirrekartoissa esiintyvät piirteet yhä pienemmäksi, abstraktimmaksi piirrekartaksi. Käytännössä alinäytteistys tehdään samaan tapaan pienikokoista ikkunaa (*eng. pooling window*) siirrellen kuten konvoluution tapauksessa, mutta nyt kullakin askeleella vain piirrekartan sisältämille arvoille tehdään jokin epälineaarinen operaa-

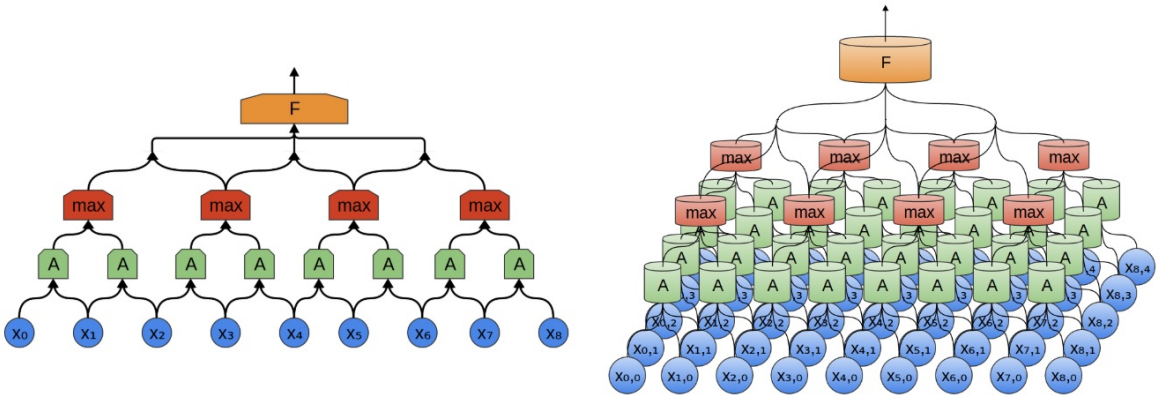
[30]Suodinta on mahdollista liikuttaa monia askelia (*eng. stride*) kerrallaan niin sivu- kuin pystysuunnassa. Tällöin muodostuvan piirrekartan koko pienenee entisestään.

tio. Tässä tapauksessa ikkuna on parametriton (vrt. suotimen painot) ja laskenta on deterministinen [150]. Nykyään laajalti käytetty operaatio on maksimialinäytteistys (*eng. max pooling*), jolla nimensä mukaisesti otetaan kustakin kartan osajoukosta tai naapurustosta maksimiarvo (L_∞-normi), joka siirretään uuteen piirrekarttaan [91]. Muita alinäytteistysfunktioita ovat muun muassa summan ja neliösumman ottaminen tarkasteltavasta naapurustosta (L_1- ja L_2-normit, vastaavasti) [54]. Huomaa, että alinäytteistysfunktio on luonteeltaan tyypillisesti permutaatioinvariantti (*eng. permutation-invariant*) [19]. Kuvassa 3.15 on havainnollistava esimerkki maksimialinäytteistyksestä [18].

Kuva 3.15: Yksinkertainen esimerkki alinäytteistyksestä.

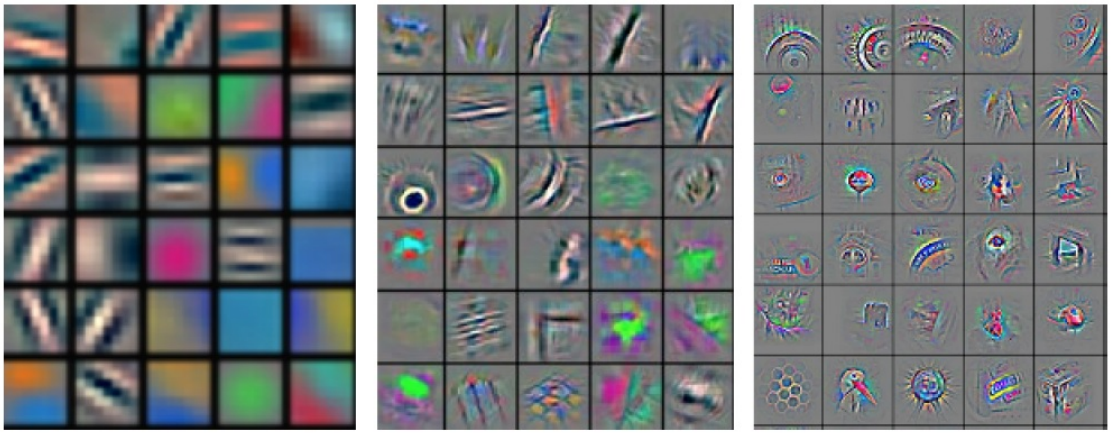
Kuvassa 3.15 on erään konvoluutio-operaation tuloksena syntynyt piirrekartta, jonka jälkeen maksimialinäytteistys tehdään. Tässä tapauksessa maksimialinäytteistys tehdään 2x2 kokoisella ikkunalla ja kahden askeleen siirtymillä värien osoittamalla tavalla.

Konvoluutioneuroverkko koostuu tyypillisesti kahdesta osasta: piirteiden irrotuksesta tai oppimisesta (*eng. feature learning*) ja luokittelijasta. Piirteiden oppiminen tapahtuu usein peräkkäisiä konvoluutionaalisia kerroksia ja alinäytteistyskerroksia putkittamalla. Piirteiden oppimisen tuloksena syntyneet lopulliset piirrekartat annetaan syötteenä luokittelijalle, joka on tavanomaisesti eteenpäin syöttävä neuroverkko. Kuva 3.16 havainnollistaa modulaarisesti konvoluutioneuroverkkoa yksi- ja kaksiulotteisen syötteen sekä yhden suotimen tapauksessa [106, 107].



Kuva 3.16: Yksinkertainen konvoluutioneuroverkko yksi- (vas.) ja kaksiulotteisella (oik.) syötteellä.

Yllä olevassa esityksessä A edustaa konvoluution tuottamaa piirrekartan yksittäistä elementtiä ja max vastaavasti maksimialinäytteistystä. Syötteen ja A:n väliset linkit ovat suotimen painoja. F on luokittelijana toimiva eteenpäin syöttävä neuroverkko. Esitys havainnollistaa miten eri tyyppisen datan luontaista rakennetta prosessoidaan konvoluutioneuroverkossa. Konvoluutioneuroverkon eräs tärkeimmistä ominaisuuksista on yhden suotimen painokertoimien rajoittaminen niin, että samoja painokertoimia käytetään kunkin kuvan osajoukon yhteydessä. Intuitiivisesti tämä oletus tarkoittaa sitä, että jos yksi piirre löytyy yhdestä syötteen osasta tietyillä painoilla, tulisi samoilla painoilla sama piirre löytyä myös syötteen jostakin toisesta osasta [75]. Toisin sanoen verkosta tulee invariantimpi erilaisille siirtymille (*eng. shift invariance*). Yllä oleva esitys havainnollistaa myös toista ominaisuutta, joka korostaa opittavien piirteiden koostumusta ja erityisesti niiden osittamista (*eng. compositionality*). Neuroverkon koon pienentyessä kohti luokittelijaa verkon eri osissa liikkuvat yksityiskohtaisista piirteistä tietoa kantavat osasyötteet yhdistyvät, jolloin muodostuu uusia, kokonaisvaltaisempia piirteitä. Kuva 3.17 havainnollistaa eräästä konvoluutioneuroverkkomallista satunnaisesti visualisoituja piirrekarttoja sen eri kerroksilta.



Kuva 3.17: Esimerkki neuroverkon syvyyden vaikutuksesta piirteiden oppimiseen.

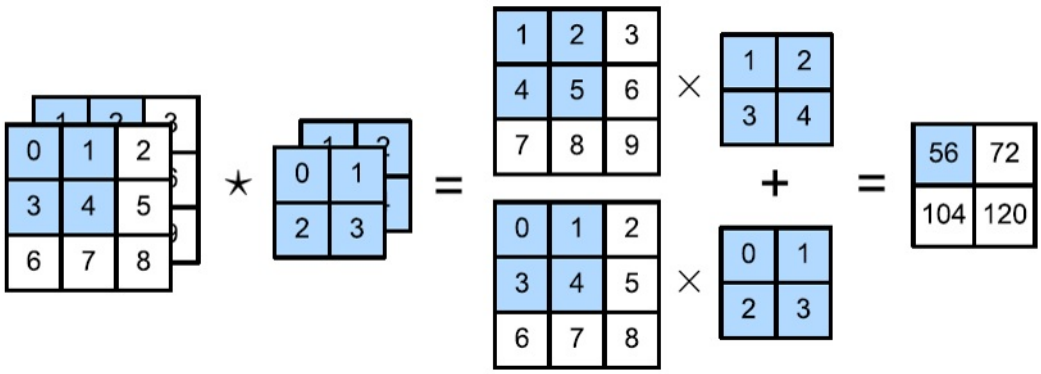
Olennaista kuvasta 3.17 on huomata, miten ensimmäisillä kerroksilla (vas.) konvoluutioneuroverkko oppii melko yksinkertaisia piirteitä, mutta syvemmillä kerroksilla niistä muotoutuu monimutkaisempia ja selkeämmin erottuvampia rakenteita. Voidaan esimerkiksi ajatella, että yksinkertaisista viivoista ja reunoista syntyy muotoja, joista edelleen voidaan rakentaa osia, jotka taas muodostavat erilaisia osakokonaisuuksia. Samanlaista hierarkisuutta löytyy myös puheesta ja tekstistä: kirjaimista syntyy sanoja, sanoista lauseita ja lauseista kokonaisia kappaleita [18, 148, 91].

Lisäksi alinäytteistyksen tarkoituksena on viime kädessä yhdistellä semanttisesti saman kaltaisia piirteitä. Tällä tavoin alinäytteistys vähentää verkon herkkyyttä niin erilaisille piirteiden vääristymille ja kohinalle eli tekee siitä invariantimman erilaisille muunnoksille (*eng. translation invariance*), kuin myös edelleen piirteiden sijainneille [91, 150]. Hiljattain on toisaalta osoitettu, että konvoluutioneuroverkoissa usein käytetyt alinäytteistystekniikat, mukaan lukien konvoluutio, eivät ota huomioon riittävät ehdot signaalin palauttamiselle takaavaa Nyquistin näytteenottoteoreema[31] (*eng. sampling theorem*), mikä lisää verkon herkkyyttä piirteiden siirtymille ja erilaisille häiriöille. Muokkaamalla alinäytteistystekniikoita sopivalla alipäästösuotimella (*eng. low-pass filter*) on saatu vakaampia sekä luotettavampia ja siten suorituskykyisempiä tuloksia [152].

Tarkastellaan seuraavaksi yksinkertaista konvoluutioneuroverkkoarkkitehtuuria. Käytännössä syöte voi olla monikanavainen eli kaksiulotteisia matriiseja voi olla pinottuna kolmanteen ulottuvuuteen muodostaen volumetrisen syötteen. Tämä on

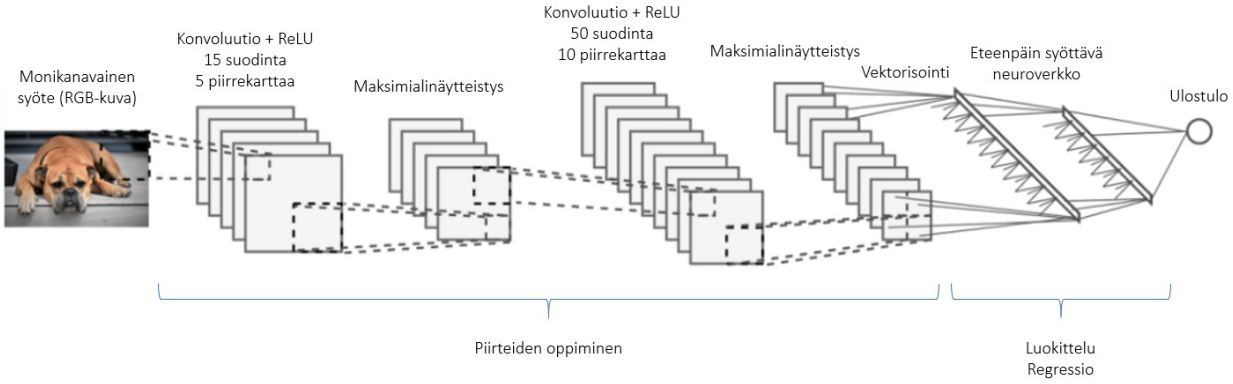
[31] Jatkuva-aikainen signaali voidaan muodostaa uudelleen näytearvoistaan, jos näytteenottotaajuus on vähintään kaksi kertaa niin suuri kuin signaalin sisältämä suurin taajuuskomponentti [68].

tyypillistä esimerkiksi värillisen kuvadatan tapauksessa, jossa RGB-värimallin mukaisesti kukin kanava edustaa yhden värin kaksiulotteisesta voimakkuusmatriisia. Tällöin konvoluution laskennassa kutakin kanavaa kohti on oma suodin ja kullekin suotimelle verkko oppii omat painonsa, mutta ulostulona on edelleen yksi piirrekartta. Kuvassa 3.18 on yksinkertainen havainnollistava esimerkki konvoluution laskennasta tässä tapauksessa [150].



Kuva 3.18: Esimerkki monikanavaisesta konvoluutiosta.

Eli konvoluutio-operaatio tehdään kullekin kanavalle erikseen, mutta laskut tehdään syvyyssuunnassa samaan aikaan samoilla askelilla. Esimerkiksi kuvan 3.18 oikeassa laidassa olevan piirrekartan luku 56 saadaan, kun pistetulot eri kanavissa lasketaan yhteen eli $(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 37 + 19 = 56$. Huomaa, että monilla erilaisilla suotimilla on mahdollista oppia erilaisia piirteitä. Yksinkertainen konvoluutioneuroverkkoarkkitehtuuri voi muodostua kuvan 3.19 mukaisesti.



Kuva 3.19: Yksinkertainen konvoluutioneuroverkkoarkkitehtuuri.

Piirteiden oppimisen jälkeen muodostuneet piirrekartat vektorisoidaan syötteeksi eteenpäin syöttävälle neuroverkolle, joka ohjatussa tapauksessa antaa ulostulona esimerkiksi todennäköisyysjakauman vektorina softmax-funktion kautta ennalta määritettyjä luokkia vastaan. Tässä tapauksessa koiran luokkaa edustava todennäköisyys tulee olla suurempi kuin muut, ideaalissa tilanteessa lähellä ykköstä.

Konvoluutioneuroverkolla on suotuisia laskennallisia ominaisuuksia: sen sisältämä parametrien määrä konvoluutionaalisissa kerroksissa on vakio eikä riipu syötekoosta. Kunkin kerroksen laskennallinen kompleksisuus on lineaarinen ja luokittelutehtävissä konvoluutioneuroverkon koko kasvaa logaritmisesti syötekokoon nähden [19]. Konvoluutioneuroverkon sisältämä kerrosten määrä voikin käytännön sovelluksissa vaihdella muutamista aina satoihin kerroksiin [63]. Huomaa, että konvoluutioneuroverkon loppuun lisättävä eteenpäin syöttävä neuroverkko lisää verkon parametrien määrää huomattavasti suhteessa muuhun verkkoon. Konvoluutioneuroverkon optimointi onnistuu myös vaivattomasti perinteisellä vastavirta-algoritmilla pienin muutoksin [90, 91].

4 Syväoppiminen ja anomalioiden havaitseminen

Ensimmäisessä alaluvussa tarkastellaan, miten edellä esitettyjä syväoppimismenetelmiä voidaan soveltaa autoenkooderiarkkitehtuuriin. Aliluvussa 4.2 käsitellään anomalioiden havaitsemismenetelmien vertailuun yleisesti käytettyjä ja tässäkin työssä hyödynnettyjä suorituskykymittareita. Korkeiden ulottuvuuksien kirouksesta, ali- ja ylisovittumisesta ja optimointimenetelmien erilaisista puutteista aiheutuvia haasteita syväoppimismenetelmien optimointiin pohditaan syvemmin aliluvussa 4.3.

4.1 Autoenkooderit

Autoenkooderi on neuroverkkomalli (*eng. autoencoder, AE*), jonka on osoitettu soveltuvan hyvin muun muassa anomalioiden havaitsemiseen ja jota tässä työssä sovelletaan [154, 26, 99, 146]. Autoenkooderi on parametrinen funktio, jonka perustana on tuottaa ulostulonaan sen saama syöte. Ideana ei ole kuitenkaan luoda jokaista syötettä itsekseen kuvaavaa identiteettifunktiota vaan autoenkooderin sisäistä rakennetta rajoitetaan siten, että malli rakentaa uudelleen eli rekonstruoi saamansa syötteen vaihtoehtoisesta, usein tiivistetystä esitysmuodosta. Tämä saavutetaan suunnittelemalla sen piilokerrokset pakottamaan syötteen esitysmuodon hetkellisesti pieniulotteisempaan avaruuteen, jolloin malli oppii mahdollisimman tiiviin, mutta kuvaavan, esityksen alkuperäisestä datasta. Jos opetettu malli ei testausvaiheessa onnistu riittävän hyvin rekonstruoimaan sille annettua syötettä, voidaan päätellä, että syöte on peräisin mallille vieraasta dataa generoivasta prosessista, joka taas voi olla merkki poikkeavasta toiminnasta tarkasteltavassa ilmiössä [138]. Autoenkooderi tunnetaan myös epälineaarisena ulottuvuuksien pienennystekniikkana (*eng. dimensionality reduction*), joka pyrkii irrottamaan ilmaisukykyisimmät muuttujat alkuperäisestä datasta suoraan [123].

Autoenkooderi koostuu yleisesti ottaen enkooderista (*eng. encoder*) $\mathcal{F}_\theta$ ja dekooderista (*eng. decoder*) $\mathcal{H}_\theta$, jotka oppivat syötteen $\mathbf{x}$ ja ulostulon $\mathbf{x}'$ välille kuvauksen

$$\mathcal{F}_\theta : \mathbf{x} \mapsto \mathbf{z} \tag{4.1}$$

$$\mathcal{H}_\theta : \mathbf{z} \mapsto \mathbf{x}', \tag{4.2}$$

missä $\mathbf{z}$ on alemman ulottuvuuden esitys. Edellä esitettyä mallia voidaan soveltaa eri syväoppimismenetelmiin. Esimerkiksi eteenpäin syöttävien neuroverkkojen tapauksessa yhden piilokerroksen autoenkooderi antaa ulostulon

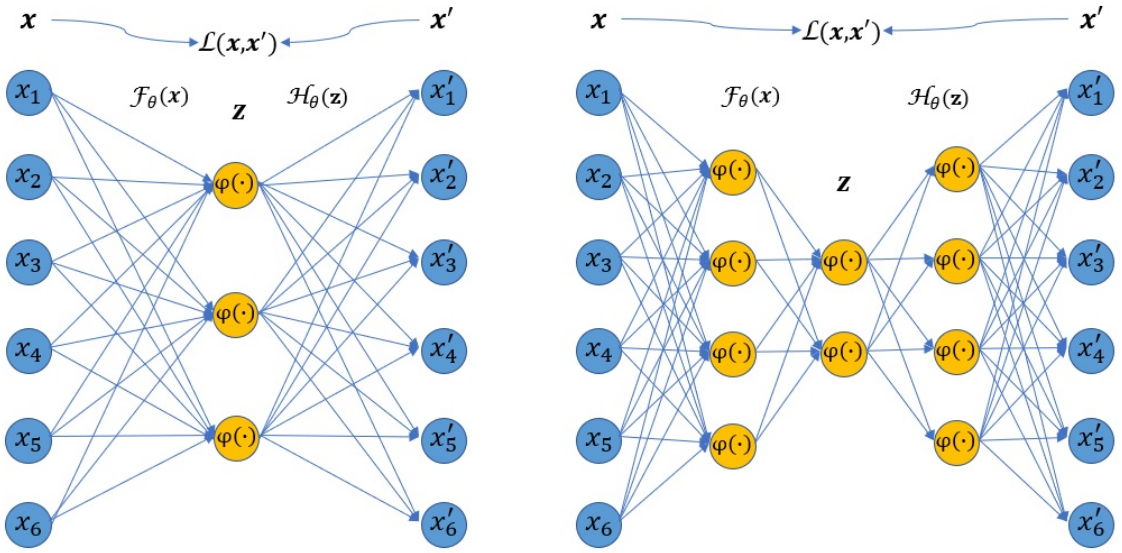
$$\mathbf{x}' \approx \mathcal{H}_\theta(\mathbf{z}) = \mathcal{H}_\theta(\mathcal{F}_\theta(\mathbf{x})) = \varphi(\mathbf{W}'(\varphi(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{b}')), \tag{4.3}$$

missä $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$, $\mathbf{z} \in \mathbb{R}^p$, $p << d$ ja $\theta = \{\mathbf{W}, \mathbf{b}, \mathbf{W}', \mathbf{b}'\}$ enkooderi- ja dekooderikohtaiset parametrit sekä φ tuttuun tapaan epälineaarinen aktivointifunktio. Autoenkooderin optimointiongelma opetusesimerkkien yli on tällöin

$$\min_\theta \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathcal{H}_\theta(\mathcal{F}_\theta(\mathbf{x}_n)), \mathbf{x}_n). \tag{4.4}$$

Optimointiongelma havainnollistaa syötteen uudelleen rakentamista: tässä tapauksessa mallin antaman tuloksen poikkeavuutta sille annetusta syötteestä mittaavaa kokonaisvirhettä kutsutaan usein rekonstruktiovirheeksi (*eng. reconstruction error*). Eteenpäinsyöttävän autoenkooderin tapauksessa verkon optimointi onnistuu perinteisellä vastavirta-algoritmilla. Useampaa kuin yhden piilokerroksen omaavaa autoenkooderia kutsutaan syväksi autoenkooderiksi, jolloin kutakin lisättyä piilokerrosta kohden tulee lisätä uusi enkooderi-dekooderi -pari [87, 154, 9]. Käytännössä syvä autoenkooderi muodostetaan usein niin, että dekooderi peilaa enkooderin arkkitehtuuria[1]. Kuva 4.1 havainnollistaa tilannetta eteenpäin syöttävän neuroverkon tapauksessa.

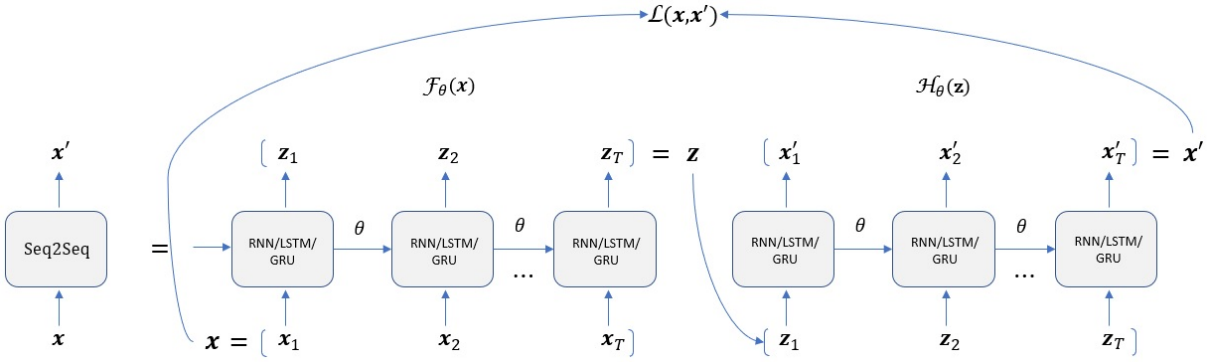
[1]Käytännön sovelluksissa käytetään toisinaan enkooderissa ja dekooderissa jaettuja painokertoimia ($\mathbf{W}' = \mathbf{W}^\top$), jolloin mallin sisältämien parametrien määrä saadaan puolitettua [9].



Kuva 4.1: Yhden piilokerroksen omaava autoenkooderi (vas.) ja syvä autoenkooderi (oik.).

Toistuvat neuroverkot taipuvat myös autoenkooderiksi, jonka optimointi onnistuu toistuville neuroverkoille ominaiseen tapaan aikakeskeisesti. Tällöin mallin enkooderi kuvaa sille syötteenä saapuvan sekvenssin ensin pienempään ulottuvuuteen, jonka jälkeen dekooderi palauttaa enkoodatun sekvenssin tavoitekokoon. Tämän tyyppisen mallin oppimisprosessia kutsutaan sekvenssioppimiseksi (*eng. sequence-to-sequence learning, Seq2Seq*). Lisäksi toistuvaan neuroverkkoon perustuva autoenkooderi ei rajoitu vakiokokoisiin syötteisiin ja tavoitteisiin, vaan ne voivat vaihdella. Tyypillisesti ohjatussa aikasarja-analyysissa alkuperäisestä datasta muodostetaan joukko jonkin tietyn ikkunan kokoisia syöte- ja tavoitepareja (*eng. sliding window/rolling forecast method*). Jos esimerkiksi syöte muodostuu sekvenssistä $[1, 2, 5, 6, 4, 3, 8, 9, 10, 12]$, niin kolmen aika-askeleen ikkunalla ensimmäinen opetusesimerkki on $([1, 2, 5], [6, 4, 3])$, toinen $([2, 5, 6], [4, 3, 8])$ ja niin edelleen. Toisaalta, jos malli opetetaan tämän tyyppisillä opetusesimerkeillä, malli vaatii aina kolme havaintoa tehdäkseen kolmen havainnon ennustuksen [22]. Anomalioiden havaitsemisen näkökulmasta tämä ei välttämättä ole ihanteellista, varsinkin jos menetelmän tarkoituksena on havaita poikkeamat mahdollisimman viiveettömästi. Näin ollen saapuvat havainnot tulee prosessoida yksi kerrallaan ja vastaavasti kullekin prosessoidulle havainnolle tulee saada tulos siitä, onko kyseessä poikkeama vai ei. Toistuvaan neuroverkkoon perustuva sekvenssioppija toimii tällä tavalla. Sekvenssioppija

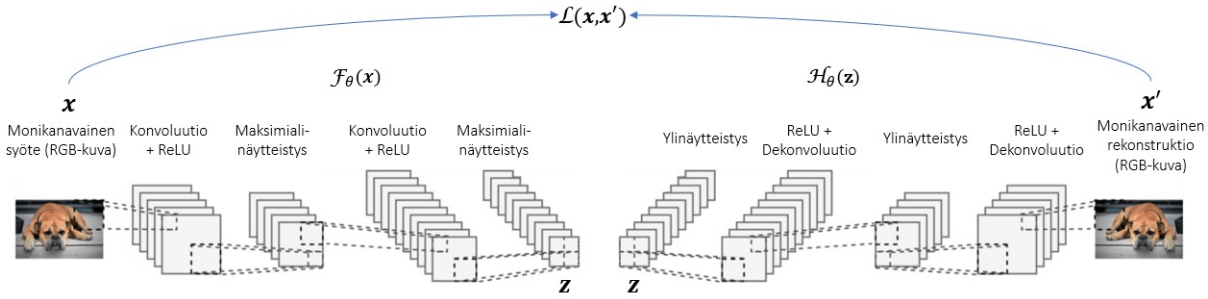
voidaan käytännössä rakentaa niin RNN kuin GRU, LSTM tai muilla toistuvilla lohkoilla [133, 132, 71]. Kuva 4.2 havainnollistaa sekvenssioppijan toimintaa. Huomaa, että sekvenssioppijan tapauksessa $\mathbf{x} = \{\mathbf{x}_t\}_{t=1}^{T}$ ja $\mathbf{x}_t, \mathbf{x}'_t \in \mathbb{R}^m, \mathbf{z}_n \in \mathbb{R}^p, p << m$ sekä θ sisältää enkooderi- ja dekooderikohtaiset parametrit.



Kuva 4.2: Sekvenssioppijan toimintaperiaate.

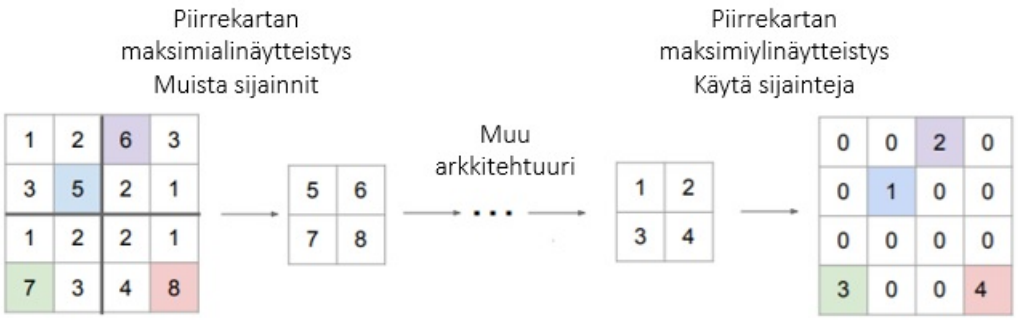
Toisin sanoen sekvenssioppijassa on kaksi toistuvaa neuroverkkoarkkitehtuuria, joista toinen toimii enkooderina $\mathcal{F}_\theta(\mathbf{x})$ ja toinen vastaavasti dekooderina $\mathcal{H}_\theta(\mathbf{z})$. Erona niissä on se, että enkooderi tuottaa alemman ulottuvuuden esitysmuodon $\mathbf{z}_t$ alkuperäisestä syötteestä $\mathbf{x}_t$, kun taas dekooderi tuottaa alkuperäistä syötettä approksimoivan rekonstruktion $\mathbf{x}'_t$ enkooderin luomasta alemman ulottuvuuden esityksestä $\mathbf{z}_t$.

Autoenkooderin voi rakentaa myös konvoluutionaalisista kerroksista. Tässä tapauksessa enkooderi koostuu neuroverkoille tutuista komponenteista, kuten konvoluutiosta ja alinäytteistyksestä. Dekooderi sisältää vastaavasti samat komponentit, mutta päinvaistaisessa järjestyksessä ja operaatiot ovat käänteisiä. Kuva 4.3 havainnollistaa konvoluutionaalista autoenkooderiarkkitehtuuria.



Kuva 4.3: Konvoluutionaalisen autoenkooderin toimintaperiaate.

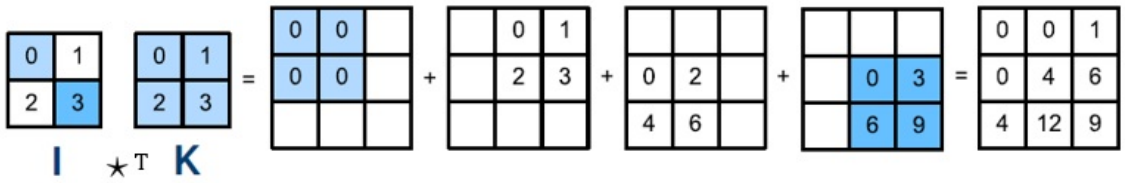
Konvoluutionaalinen autoenkooderi ei poikkea paljoa tavallisesta konvoluutioneuroverkosta: ottamalla kuvan 3.19 konvoluutioneuroverkon lopusta eteenpäin syöttävän neuroverkon kerrokset ja vektorisoinnin pois, ja jatkamalla jäljelle jäänyttä arkkitehtuuria sen peilikuvalla, saadaan kuvan 4.3 konvoluutionaalinen autoenkooderi. Huomaa, että nyt autoenkooderi on puhtaasti konvoluutionaalinen (*eng. fully convolutional*). Konvoluutionaalisissa autoenkoodereissa tyypillisesti kutakin enkooderin alinäytteistystä kohti on dekooderissa vastaava ylinäytteistys, kuten on kuvassa 4.3. Alinäytteistykselle ei ole olemassa suoraan käänteistä vastinetta ja siihen voidaan soveltaa erilaisia ylinäytteistyksen (*eng. unpooling/upsampling*) toteuttavia menetelmiä. Tällöin esimerkiksi maksimin tapauksessa uusi piirrekartta luodaan sijoittamalla enkooderin alinäytteistyksessä prosessoidun piirrekartan maksimiarvot dekooderin ylinäytteistyksen luomaan saman kokoisen piirrekartan samoihin positioihin ja asettamalla muut arvot nollaksi. Kuva 4.4 havainnollistaa tämän tyyppistä ylinäytteistystä.



Kuva 4.4: Maksimiylinäytteistys konvoluutionaalisessa autoenkooderissa.

Lisäksi kutakin enkooderin konvoluutiota kohti on dekooderissa vastaava operaa-

tio, jolla on neuroverkkokirjallisuudessa erilaisia nimiä, joista yksi on dekonvoluutio. Se on kuitenkin harhaanjohtava, koska neuroverkoissa konvoluution käänteisoperaatio ei tee digitaalisesta signaalinkäsittelystä tuttua dekonvoluutiota (*eng. deconvolution*) vaan transponoidun konvoluution (*eng. transpose convolution*). Käytännössä transponoitu konvoluutio toimii samalla periaatteella kuin konvoluutio, mutta nyt alkuperäinen piirrekartta pyritään palauttamaan suotimen ja konvoluution tuloksena syntyneen piirrekartan avulla. Kuva 4.5 havainnollistaa yksinkertaisen kaksiulotteisen esimerkin kautta transponoitua konvoluutiota, jota merkitään $\star^{\top}$.



Kuva 4.5: Transponoidun konvoluution idea.

Kun konvoluution laskenta esitetään matriisimuodossa, ottamalla painokerroinmatriisista eli suotimesta transpoosi saadaan transponoitu konvoluutio. Tästä transponoidun konvoluution nimi juontaa. Huomaa, että ylinäytteistyksen ja transponoidun konvoluution eräs keskeinen ero on se, että transponoitu konvoluutio on parametrinen eli opetettavissa osana arkkitehtuuria, kun taas ylinäytteistys on deterministinen. Lisäksi voidaan osoittaa, että transponoitu konvoluutio[2] on sama operaatio kuin yksittäisen normaalin konvoluution gradientin laskeminen [148, 72, 150, 42].

4.2 Suorituskykymittarit

Anomalian havaitsemismenetelmiin liittyvissä tutkimuksissa yleisesti käytettyihin binääriluokitteluun soveltuviin mittareihin lukeutuu sekaannusmatriisi (*eng. confusion matrix*). Se on binääriluokittelussa 2x2 matriisi[3], johon yläriville sijoitetaan normaalidatan suhteen oikeaksi (*eng. true negative, TN*) ja vääräksi (*eng. false positive, FP*) tunnistettujen havaintojen lukumäärät ja alariville vastaavasti hyökkäysda-

[2]Konvoluutioiden ja erityisesti transponoitujen konvoluutioiden aritmetiikkaa löytyy kattavasti lähteestä [42].

[3]Moniluokitteluongelmissa sekaannusmatriisi on kooltaan $M \times M$, missä M on luokkien määrä.

tan suhteen vääräksi (*eng. false negative, FN*) ja oikeaksi (*true positive, TP*) tunnistettujen havaintojen lukumäärät. Näin sijoiteltuna kukin sarake edustaa ennustettua luokkaa rivien edustaessa todellisia luokkia: esimerkiksi "Normaali" -riviä lukiessa TN havainnot ovat niitä normaaleja havaintoja, jotka malli ennusti oikeellisesti normaaleiksi havainnoiksi ja FP vastaavasti virheellisesti poikkeamiksi. Lisäksi diagonaalille sijoittuu oikein luokiteltujen havaintojen lukumäärät luokkineen ja diagonaalin ulkopuolelle jäävät väärin luokiteltujen havaintojen lukumäärät luokkineen. Taulukko 4.1 havainnollistaa matriisia.

Taulukko 4.1: Sekaannusmatriisi.

Luokka	Normaali	Anomalia
Normaali	TN	FP
Anomalia	FN	TP

Anomalioiden havaitsemisen kannalta kriittisin tieto sekaannusmatriisissa on FN eli kokonaan havaitsematta jääneiden toteutuneiden hyökkäysten lukumäärä. Näiden joukossa voi olla esimerkiksi kehittyneempiä normaalin liikenteen sekaan sopeutuvia APT-hyökkäyksiä, joiden huomaamatta jäämisellä on kriittisimmät seuraukset suojattavan järjestelmän toiminnan kannalta. Tällöin menetelmän tulee olla herkkä pienillekin poikkeamille, mikä toisaalta voi kasvattaa väärien hälytysten määrää. Sekaannusmatriisin pohjalta voidaan johtaa muita tyypillisesti käytettyjä mittareita anomalioiden havaitsemismenetelmien vertailuun. Taulukossa 4.2 on määritelty tässä työssä käytetyt mittarit[4].

Taulukko 4.2: Tässä työssä käytetyt suorituskykymittarit.

Mittari	Määritelmä
Virheettömyys (*eng. accuracy*)	$\frac{TP+TN}{TN+TP+FN+FP}$
Osumatarkkuus (*eng. precision, P*)	$\frac{TP}{TP+FP}$
Herkkyys (*eng. recall, R*)	$\frac{TP}{TP+FN}$
F1-mitta (*eng. F1-score*)	$2\,\frac{P \times R}{P+R}$

[4]Monet kahden luokan sekaannusmatriisista johdetut suorituskykymittarit eivät ole helposti yleistettävissä moniluokkaisiin sekaannusmatriiseihin.

Virheettömyydellä mitataan mallin kykyä tunnistaa normaalia ja poikkeavaa liikennettä onnistuneesti kaikkien tehtyjen luokitusten suhteen. Osumatarkkuudella pyritään saamaan käsitys siitä, miten hyvin malli välttää väärien hälytysten antamista. Sitä voidaan ajatella todennäköisyytenä, jolla mallin tekemä luokitus osuu oikeaan. Herkkyys taas kuvaa mallin tunnistamien hyökkäyksien määrää kaikista hyökkäyksistä eli miten herkästi malli tunnistaa pienetkin todelliset poikkeamat. Herkkyys on tärkeä mittari anomalian havaitsemismenetelmän suorituskyvyn kannalta. Huomaa, että osumatarkkuus ja herkkyys ovat toistensa vastakohtia: korkean osumatarkkuuden kohdalla herkkyys kärsii ja päinvastoin. Intuitio tässä on se, että oletusarvoisesti kaikkien hyökkäyksien havaitsemiseksi menetelmän tulee olla valmis prosessoimaan monia vääriä hälytyksiä, jolloin osumatarkkuus on huono. Jos tehokkuus on toisaalta ensisijainen tavoite, menetelmä oletetusti tunnistaa suuremmat poikkeamat pienien poikkeamien jäädessä vähemmälle huomiolle, jolloin osumatarkkuus on vähäisten väärien hälytyksen myötä korkea, mutta herkkyys jää matalaksi[5] [83, 78, 11, 51].

Hyvän osumatarkkuuden ja herkkyyden omaavat anomalian havaitsemismenetelmät ovat harvassa. F1-mitta on käytännössä osumatarkkuuden ja herkkyyden harmoninen keskiarvo, eli $\frac{N}{\sum_{n=1}^{N} \frac{1}{x_n}} \Rightarrow \frac{2}{\frac{1}{P}+\frac{1}{R}} = \frac{2}{\frac{P+R}{P\times R}} = 2\,\frac{P\times R}{P+R}$. Aritmeettisen keskiarvon sijaan harmoninen keskiarvo sopii paremmin suhdeluvuille, kuten osumatarkkuudelle ja herkkyydelle. Jos esimerkiksi mallin osumatarkkuus on 1.0 ja herkkyys 0.2, eli menetelmä luokittelee havaintoja anomalioiksi hyvin konservatiivisesti antaen vähän vääriä hälytyksiä, mutta jättää 80% hyökkäyksistä tunnistamatta, tulee mittareiden aritmeettiseksi keskiarvoksi 0.6 harmonisen keskiarvon ollessa 0.33. Tämä on huomattavasti realistisempi tulos tässä tapauksessa. F1-mitta saa korkean arvon vain silloin, kun osumatarkkuus ja herkkyys ovat molemmat korkeita [126, 51].

Binääriluokittelussa kuten anomalioiden havaitsemisessa käytetään mallien suorituskyvyn arviointiin toisinaan ROC AUC -käyrää (*eng. area under the receiving operating characteristic curve*) [156, 11]. Se on käyttökelpoinen silloin, kun datajoukossa havainnot ovat jakautuneet tasaisesti luokkien kesken [28]. Tässä työssä tämä ei kuitenkaan päde, joten ROC AUC -käyrän käsittely jätetään mainitsemisen tasolle.

[5] Triviaalissa tapauksessa täydellisen osumatarkkuuden saa, kun tekee yhden ennustuksen ja varmistaa, että se on anomalia. Tässä tapauksessa tosin kaikki muut anomaliat, paitsi tuo tunnistettu, jäävät havaitsematta. Usein onkin hyvä pohtia, että mikä on hyväksyttävä herkkyys tietyn osumatarkkuuden saavuttamiseksi (*eng. precision-recall -tradeoff*).

4.3 Optimointiin liittyviä haasteita

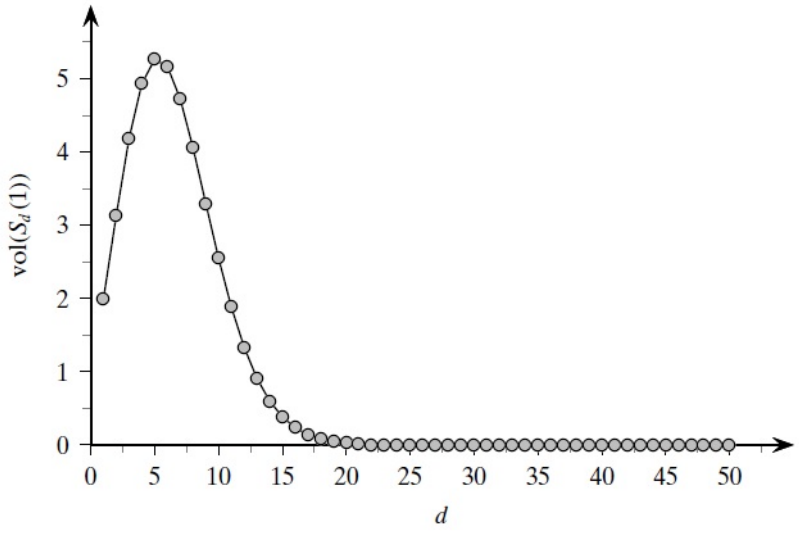
Suorituskykyisten syväoppimismallien kehittäminen ja optimointi ei ole aivan suoraviivaista. Niiden joustavuus ja poikkeuksellinen mallinnuskyky tuo mukanaan lukuisia säädettäviä parametreja, joilla voidaan sovittaa syväoppimismalleja saatavilla olevaan dataan. Käytännössä ei ole olemassa yksiselitteisiä ohjeita tai sääntöjä siitä, että minkälainen syväoppimismalli milläkin arkkitehtuurilla missäkin sovelluksessa takaa suorituskykyisimmät tulokset [21]. Syväoppinen kulkeekin vahvasti käsi kädessä empiirisen tutkimuksen kanssa ja muistuttaa insinööritiedettä. Tähän osaltaan vaikuttaa se, että syväoppimista sovelletaan perinteisten tilastotieteellisten menetelmien sovellusalueiden ulkopuolelle jääviin hyvin suuriin ja moniulotteisiin datajoukkoihin, mikä on nykypäivänä mahdollista muun muassa saatavilla olevan datan ja resurssien myötä. Lisäksi syväoppiminen on ollut esillä vasta muutaman vuoden: syväoppimisen täyttä potentiaalia ei vielä tunneta ja sille ilmenee jatkuvasti uusia käyttötapauksia [28].

Syväoppimismenetelmien optimoinnissa säädettäviä parametreja kutsutaan usein hyperparametreiksi, koska niillä vaikutetaan oppimisprosessiin ja näin niille saadaan myös mallin sisäisistä parametreista eroava käsite. Tällöin mallin parametreiksi kutsutaan niitä mallin sisäisiä parametreja, jotka se oppii optimoinnin aikana eli esimerkiksi painokertoimet ja vakiotermit. Monissa syväoppimismenetelmiä soveltavissa tutkimuksissa mallien optimaaliset parametrit etsitään usein kokeilemalla eli muuttamalla valittuja hyperparametreja saavutetun suorituskyvyn pohjalta. Hyperparametrien muuttaminen voi tapahtua esimerkiksi noudattamalla ennalta laadittuja konfiguraatiolistoja (*eng. grid search*) tai jonkin yksinkertaisen tai kehittyneemmän algoritmin ohjaamana. Käytännön sovelluksissa hyviksi todettuja vinkkejä syväoppimismallien optimointiin on kuitenkin olemassa paljon niin tutkimusten ja kirjojen kuin erilaisten blogikirjoitusten muodossa [28, 21].

Korkeiden ulottuvuuksien kirouksesta

Laskennallisen data-analyysin ja anomalian havaitsemisen yhteydessä puhutaan usein korkeiden ulottuvuuksien kirouksesta (*eng. curse of dimensionality*) Euklidisessä avaruudessa [156]. Tällä viitataan analysoitavien datajoukkojen sisältävien havaintojen $\mathbf{x}$ muuttujien määrään n eli dimensionaalisuuteen sekä miten datan ja erityisesti datan määrittelevän avaruuden ominaisuudet muuttuvat, kun siirrytään intuitiivisis-

ta 2- ja 3-ulotteisista avaruuksista korkeampiin[6] ulottuvuuksiin $\mathbf{x} \in \mathbb{R}^{n>3}$. Kaksi- ja kolmeulotteisista avaruuksista muodostunut geometrinen intuitio muuttuu nopeasti korkeaulotteisempiin avaruuksiin siirryttäessä. Tarkastellaan esimerkiksi, miten neliön sisälle piirretyn suurimman ympyrän pinta-ala ja kuution tapauksessa pallon tilavuus muuttuu. Neliön ja ympyrän tapauksessa saadaan $\frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$ eli ympyrä täyttää neliöstä hieman yli kolme neljäsosaa (≈ 0.785). Kuution ja pallon kohdalla tilanne on vastaavasti $\frac{\frac{4}{3}\pi r^3}{(2r)^3} = \frac{\pi}{6}$ eli pallo täyttää kuutiosta enää hieman yli puolet (≈ 0.524). Yleisessä tapauksessa eli ulottuvuuksien annettaessa kasvaa, suhde konvergoituu nollaan. Intuitio tässä on se, että ulottuvuuksien kasvaessa palloa ympäröivä tila saa lisää "ilmaa" reunoille ja kulmiin pallon supistuessa avaruuden keskelle. Tämä havainto ei vielä ole kovin intuition vastainen, mutta tarkastellaanpa palloa erikseen: voidaan osoittaa, että n-ulotteisen yksikköpallon tilavuus kasvaa aina viidenteen ulottuvuuteen asti, kunnes se kääntyy jyrkkään laskuun ja tulee lähestulkoon olemattomaksi 30 ulottuvuuteen mennessä. Kuva 4.6 havainnollistaa yksikköpallon tilavuuden muutosta ulottuvuuksien suhteen.



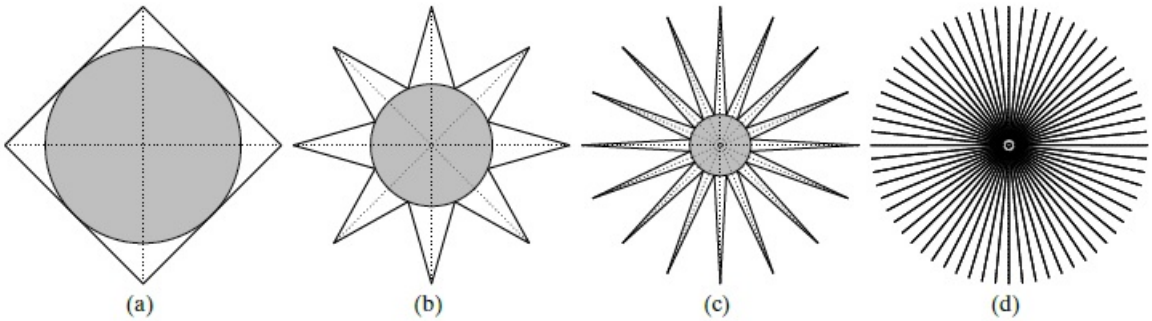
Kuva 4.6: Yksikköpallon tilavuuden muutos ulottuvuuden funktiona.

Lisäksi voidaan tarkastella ohutta rengasta tai kuorta (*eng. thin shell*), jonka sisäinen säde on $r - \epsilon$, ulkoinen säde r ja ϵ kuoren paksuus. Ohuen kuoren tilavuus saadaan

[6]Englannin kielessä törmää usein termiin *hyperspace* useamman kuin kolmen ulottuvuuden omaavalle avaruudelle. Samoin yli kolmen ulottuvuuden omaavat geometriat saa usein *hyper*-etuliitteen, kuten esimerkiksi *hyperplane*, *hypercube*, *hypersphere*, *hyperrectangle* ja *hyperball*.

vähentämällä ulomman säteen omaavan pallon tilavuus sisäisen säteen omaavasta pallosta. Jos verrataan kuoren tilavuuden suhdetta ulomman säteen omaavan pallon tilavuuteen ulottuvuuksien kasvaessa, suhde konvergoituu ykköseen[7]. Tällöin ulottuvuuksien kasvaessa kuoren tilavuus tulee sisältämään koko pallon tilavuuden lähes kokonaan. Tämä tarkoittaa sitä, että jos data on tasaisesti jakautunut n-ulotteiseen avaruuteen, kaikki datapisteet hajaantuvat avaruuden reunoille tai avaruutta edustavan geometrisen muodon pinnalle, tässä tapauksessa korkeaulotteisen pallon [147, 12]. Näin ollen korkeaulotteisessa avaruudessa sijaitsevat datapisteet ovat suurin piirtein yhtä kaukana toisistaan ja voidaankin osoittaa, että ulottuvuuksien kasvaessa jotakin pistettä lähimpänä ja kauimpana olevien pisteiden suhteellinen ero lopulta häviää (*eng. concentration effect*). Tämä ei toisaalta päde datapisteille, jotka eivät noudata muun datajoukon generoivaa prosessia. Itse asiassa, jos datassa esiintyä anomalia on peräisin muusta dataa generoivasta prosessista, ulottuvuuksien lisääminen saattaa korostaa sen poikkeavuutta muusta datasta [156].

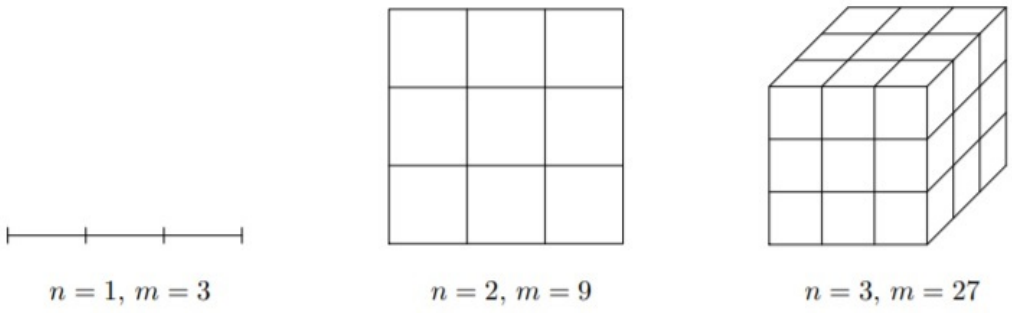
Lisäksi diagonaaleilla on erityinen ominaisuus: esimerkiksi kuution tapauksessa kaikkien diagonaalien ja kaikkien koordinaattiakselien välinen kulma kasvaa korkeissa ulottuvuuksissa käytännössä kohtisuoriksi. Tällöin diagonaalien lähellä sijaitsevia datapisteitä ei voida projisoida alempiin ulottuvuuksiin. Kuva 4.7 havainnollistaa konseptuaalisesti vielä tason näkökulmasta, miten a) neliö ja sen sisältämä ympyrä käyttäytyy kahdessa ulottuvuudessa, miten kuutio ja sen sisältämä pallo käyttäytyy b) kolmessa ja c) neljässä ulottuvuudessa sekä d) korkeammissa ulottuvuuksissa [147, 12]. Huomaa, että ulottuvuuksien kasvaessa kuution kulmien määrä kasvaa suhteessa 2^n, jolloin myös diagonaalien määriksi tulee 2^n.



Kuva 4.7: Kuution ja sen sisältämän pallon käyttäytyminen eri ulottuvuuksissa.

[7]Suhde sieventyy lopulta muotoon $1 - (1 - \frac{\epsilon}{r})^n$. Lisätietoa löytyy kattavasti muun muassa lähteestä [147].

Korkeiden ulottuvuuksien kirous perustuu alunperin havaintoon, että ulottuvuuksien kasvaessa avaruuden tilavuus kasvaa eksponentiaalisesti. Kuva 4.8 havainnollistaa tilannetta: jos avaruus jaetaan yhtä moneen ja yhtä suureen osaan, ulottuvuuksien n kasvaessa avaruuden yhtä suurien alueiden määrä m moninkertaistuu. Lisäksi kuvasta 4.8 voi huomata, että tilavuus kasvaa ulospäin reunoja kohti. Tämä tarkoittaa datapisteiden jakautuessa tasaisesti sitä, että pisteet sijaitsevat todennäköisimmin avaruuden reuna-alueilla eli kaukana keskustasta. Tässä tapauksessa reunalla olevien alueiden m lukumäärä kasvaa suhteessa $3^n - 1$ [130].



Kuva 4.8: Kuution tilavuuden muutos ulottuvuuksien kasvaessa.

Tilavuuden räjähdysmäinen kasvu ja Euklidisen avaruuden käyttäytyminen korkeissa ulottuvuuksissa heikentää esimerkiksi etäisyysmetriikkojen toimintaa ja siten etäisyysmetriikoihin perustuvia erilaisia klusterointimenetelmiä sekä lähimmän naapureiden menetelmiä [156]. Korkeampien ulottuvuuksien kiroukselta voidaan periaatteessa välttyä lisäämällä datan määrää. Teoriassa korkeammissa ulottuvuuksissa tarvittavan datan määrä suhteessa alempiin ulottuvuuksiin kasvaa kuitenkin eksponentiaalisesti. Tällöin esimerkiksi jo 100 muuttujaa vaatisi enemmän (tasaisesti jakautuneita) opetusesimerkkejä kuin atomeja on havaittavissa olevassa maailmankaikkeudessa, mikäli niiden keskinäisen etäisyyden eli tiheyden halutaan säilyvän keskimäärin kymmenesosan sisällä [51]. Tämän perusteella voidaan todeta, että moniulotteisten tyypillisesti luonteeltaan harvojen datajoukkojen kanssa syväoppimismenetelmät välttämättä ylisovittuvat, jollei optimointia kontrolloida.

Kaiken kaikkiaan korkeiden ulottuvuuksien kirous on tärkeää pitää mielessä data-analyysia tehdessä. Se ei kuitenkaan poissulje tai estä tutkimusta moniulotteiseen dataan soveltuvista menetelmistä, vaan ovat päinvastoin hyvin tärkeä tutkimusaihe sekä yleisesti että anomalioiden havaitsemismenetelmien kannalta [156,

130, 96]. Käytännön sovelluksissa data on usein kerääntynyt tarkasteltavan avaruuden johonkin tiettyyn osaan eli aliavaruuteen kuten pistejoukkoon tai monistoon, jota voidaan kuvata erilaisilla tekniikoilla. Esimerkiksi kolmiulotteisessa avaruudessa havainnot voivat olla kerääntyneet johonkin kaksiulotteiseen tasoon [12]. Muun muassa Johnson-Lindenstraussin lemman mukaan korkeaulotteisessa Euklidisessa avaruudessa sijaitsevat n pistettä voidaan lineaarisesti[8] muuntaa vähintään k-ulotteiseen aliavaruuteen, missä $k \geq \frac{24 \log n}{3\epsilon^2 - 2\epsilon^3} \in \mathbb{N}_+$ siten, että minkä tahansa kahden pisteen välinen etäisyys muuttuu $(1 \pm \epsilon)$ verran kun $0 < \epsilon < 1$ [38]. Tätä ominaisuutta kutsutaan toisinaan myös datan luontaiseksi avaruudeksi tai ulottuvuudeksi (*eng. intrinsic dimensionality*), jolla tarkoitetaan pienintä määrää alkuperäiden datajoukon ominaisuudet eli geometrian säilyttäviä muuttujia [96]. Tähän viitattiin jo edellä autoenkoodereiden yhteydessä ulottuvuuksien pienennystekniikkana. Tätä voidaan myös hyödyntää poikkeamien etsinnässä: kuten mainittua, ulottuvuuksien lisääminen saattaa tuoda anomaliat paremmin esiin. Toisaalta, jos ulottuvuuksia eli muuttujia on datassa paljon ja osa niistä eivät ole merkityksellisiä datan luonteen kannalta, ulottuvuuksien määrää on silloin parempi karsia. Haastavaa onkin löytää se ulottuvuus eli aliavaruus, missä datan luonne ja erityisesti poikkeamat tulevat parhaiten esiin.

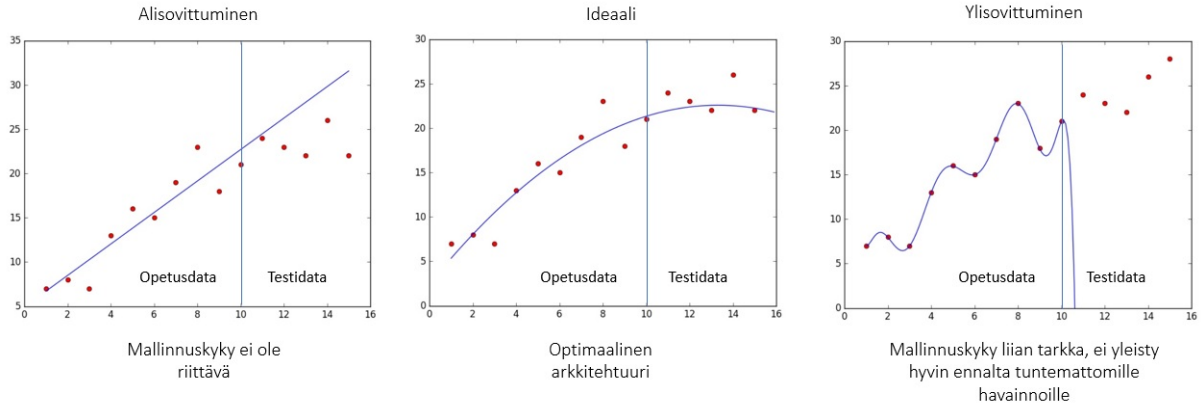
Yli- ja alisovittuminen

Vaikka hyperparametrien summittainen tai systemaattinen etsiminen ja kokeileminen auttaa optimaalisten parametrien etsinnässä, se on kuitenkin pitkälti mekaanista. Syväoppimismallin arkkitehtuuri ja suorituskyky ovat sovelluskohtaisia[9], jolloin syväoppimismallin käyttäjän tulee mallia optimoitaessa seurata ja analysoida sen käyttäytymistä ja suorituskyvyn kehittymistä. Eräs yleisimmistä ja samalla tärkeimmistä syväoppimismallien optimointiin liittyvistä haasteista on ylisovittuminen: universaalin approksimaatioteorian myötä syväoppimismallit pyrkivät sovittumaan aina ideaalisti datan määrästä ja saatavuudesta riippumatta. Tämä ei aina

[8] Lineaaristen mallien lisäksi on kehitetty monia menetelmiä, joilla voidaan mallintaa epälineaarisia muunnoksia ja aliavaruuksia [96]. Näihin lukeutuu myös autoenkooderi.

[9] Joissakin tapauksessa yhtä syväoppimismallia voidaan käyttää lähtökohtana jonkin toisen tehtävän ratkaisemiseen (*eng. transfer learning*). Tällöin esimerkiksi autoja tunnistavaa mallia voidaan hyödyntää kuorma-autojen tunnistukseen niin, että autojen kuvilla esiopetettua mallia opetetaan lisää kuorma-autojen kuvilla. Näin autoista opittua piirteitä voidaan hyödyntää myös kuorma-autojen tunnistukseen.

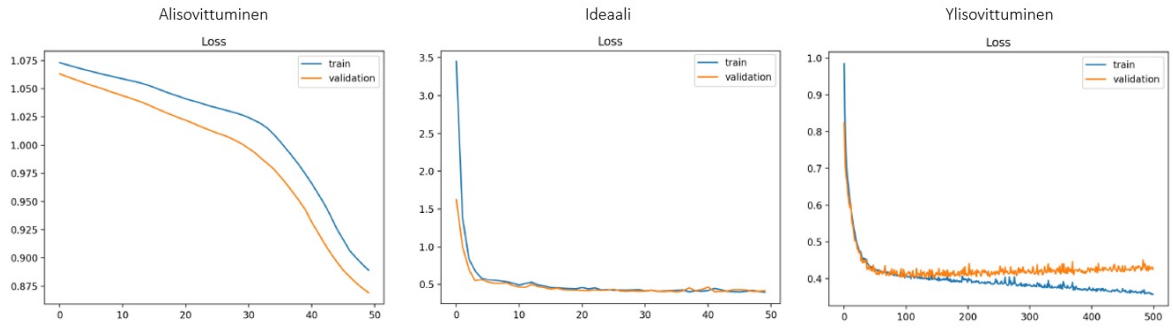
ole suotuisa ominaisuus, sillä reaalimaailman sovelluksista kerätty data on luontaisesti kohinaista ja erityisesti pienillä otosmäärillä havaintojen välistä jatkuvuutta ei voida taata. Käytännössä ei voida siis luottaa siihen, että käytössä oleva otos kuvaa täysin mallinnettavaa ilmiötä. Lisäksi syväoppimismalleja optimoidaan tyypillisesti opetus- ja validointidatajoukkojen avulla, kun taas mallin todellista suorituskykyä testataan mallille entuudestaan tuntemattomilla havainnoilla. Mallin validointidatajoukkoa vastaan mitatulle suorituskyvylle ei tule antaa liikaa painoarvoa. Näin ollen liiallisen sovittumisen sijaan syväoppimismallilla pyritään oppimaan ne tärkeimmät ja ilmaisukykyisimmät piirteet, joilla malli on suorituskykyinen ja hyödyllinen otoksen ulkopuolisten havaintojen tapauksessa [28, 21]. Kuva 4.9 havainnollistaa sovittumiseen liittyviä ongelmia [137, 54].



Kuva 4.9: Sovittumiseen liittyviä ongelmia.

Kuvassa 4.9 on kolme mallia, jotka on sovitettu opetusdataan kuvaajien esittelemällä tavalla. Vasemmassa laidassa lineaarinen malli näyttää sovittuvan suhteellisen hyvin opetusdataan, mutta yleistyy heikommin testidatan eli uusien havaintojen ilmaantuessa. Kuvan 4.9 oikeassa laidassa sen sijaan huomattavasti monimutkaisempi 9-asteen polynomifunktio sovittuu lähes täydellisesti opetusdataan, mutta ei yleisty ollenkaan ennalta tuntemattomille havainnoille. Keskimmäinen toisen asteen polynomisovitus on tässä tapauksessa ideaalisin, koska se ei ole liian monimutkainen, mutta silti yleistyy hyvin testidatalle. Sovittumiseen liittyviä ongelmia ei käytännössä kuitenkaan pystytä tunnistamaan kuvan 4.9 havainnollistamalla tavalla, vaan ongelmat tulee tunnistaa syväoppimismalleja optimoitaessa eli oppimisen aikana. Eräs yksinkertainen, mutta tehokas työkalu oppimisen diagnosointiin on oppimiskäyrä (*eng. learning curve*), jota käytetään yleisesti toistuvien iteraatioi-

den kautta suorituskykyään parantavien kone- ja syväoppimismallien yhteydessä. Oppimiskäyrä on mallin suorituskyvyn kuvaaja ajan funktiona, joka saadaan esimerkiksi piirtämällä kustannusfunktion arvo kullakin optimointikierroksella. Usein oppimis- ja validointidatajoukon oppimiskäyrät sijoitetaan samaan kuvaajaan kuvan 4.10 esittelemällä tavalla.



Kuva 4.10: Sovittumiseen liittyvien ongelmien ilmeneminen optimoinnissa.

Oppimiskäyrällä on kätevä tunnistaa sovittumiseen liittyviä ongelmia. Ylisovittuminen tapahtuu silloin, kun mallin suorituskyky validointidatajoukkoa vastaan alkaa heiketä, vaikka suorituskyky opetusdatajoukkoa vastaan jatkaa paranemistaan. Tämä näkyy kuvan 4.10 oikeassa laidassa. Kuvan vasemmassa laidassa oleva kuvaaja havainnollistaa alisovittumista: mallilla on vielä parantamisen varaa eli jatkamalla optimointia, sen mallinnuskyky oletettavasti paranee. Alisovittuminen voi ilmetä oppimiskäyrässä myös vaakasuorana viivana, jolloin mallin sen hetkinen arkkitehtuuri ei tarjoa riittävää mallinnuskykyä datan mallintamiseen. Ideaalitilanteessa oppimiskäyrä muistuttaa funktion $\frac{1}{x}$ kuvaajaa eli mallin suorituskyky niin opetuskuin validointidatajoukkoa vastaan paranee ja konvergoituu nopeasti. Tämä ilmenee kuvan 4.10 keskimmäisestä kuvaajasta. Joissakin tapauksissa opetusdatajoukon oppimiskäyrä voi olla ylempänä kuin validointidatajoukon oppimiskäyrä ja päinvastoin. Tällöin edeltävässä tilanteessa validointidatajoukkoa on helpompi ennustaa kuin opetusdatajoukkoa, jolloin validointidataa on syytä monipuolistaa esimerkiksi hankkimalla lisää dataa tai muuttamalla opetus- ja validointidatajoukkoja. Jälkimmäisessä tapauksessa mallilla on vaikeuksia validointidatajoukon kanssa opetusdataan ylisovittumisen takia. Ylisovittumista vastaan on monia erilaisia keinoja, joihin syvennytään erikseen myöhemmin. Lisäksi käyrissä esiintyvät erilaiset heilahtelut voivat antaa viitteitä siitä, että käytössä olevat datajoukot eivät ole tarpeeksi edustavia. Tällöin malli saattaa tarvita lisää dataa tai datan esiprosessointia [21].

Syväoppimismallien heikko suorituskyky johtuu usein niiden oppimiseen, yleistettävyyteen ja ennustuksiin liittyvistä haasteista, jotka ovat kytköksissä toisiinsa. Esimerkiksi oppimisen aikana esiintyvät ongelmat kuten ali- ja ylisovittuminen vaikuttavat mallin suorituskykyyn ennalta tuntemattomia havaintoja vastaan ja täten ennustusten tarkkuuteen. Lisäksi mallin tekemiin ennustuksiin ja niiden hajontaan vaikuttaa syväoppimismallien luontainen stokastisuus, joka ensi kädessä juontaa satunnaisesti alustettavista mallin sisäisistä parametreista. Mallin arkkitehtuurissa voi myös olla muita satunnaisuutta hyödyntäviä komponentteja, jotka lisäävät edelleen mallin stokastisuutta. Toisin sanoen syväoppimismallien tulokset ovat harvoin identtisiä eri optimointikerroilla, mikä pitää ottaa huomioon malleja optimoitaessa ja kehitettäessä. Vakaasti oppiva malli approksimoi samaa minimikohtaa eri aloituspisteestä eli parametrien aloitusarvoista huolimatta ja antaa pienen hajonnan omaavia tuloksia. Ideaalitilanteessa syväoppimismalli on suorituskykyinen niin opetus-, validointi- kuin testidatajoukkoa vastaan, antaa luotettavia tuloksia useampien optimointikertojen yli sekä oppii vakaasti ja nopeasti [21].

Optimointimenetelmistä

Moniulotteisessa ei-konveksisessa tapauksessa optimoitavan funktion pinta saattaa olla hyvinkin epälineaarinen, jolloin esimerkiksi yhden ulottuvuuden suunnassa pinta saattaa olla jyrkempi verrattuna toisiin paremman optimin omaaviin ulottuvuuksiin, mikä on yleistä muun muassa lokaalin minimin ympärillä. Näin ollen monissa perinteisen gradienttimenetelmän varianteissa hyödynnetään erilaisia tekniikoita, joilla oppimisnopeus pyritään sopeuttamaan dynaamisesti gradientin muodostaviin parametrikohtaisiin osittaisderivaattioihin. Oppimisnopeutta säädetään iteraatioiden mukana muuttujakohtaisesti gradientin avulla[10]. Käytännössä liian alhainen oppimisnopeus hidastaa konvergoitumista optimiin, liian suuri oppimisnopeus saattaa johtaa jopa hajaantumiseen eli divergoitumiseen ja muuten sopimaton oppimisnopeus aiheuttaa erilaisia heilahteluja liikuttaessa käyvien ratkaisujen joukkoa pitkin kohti minimiä. Monet perinteisen gradienttimenetelmän variantit perustuvat eksponentiaalisesti liukuvaan keskiarvoon (*eng. exponential moving average, EMA*) tuoreempien gradienttien painottamiseksi. Syväoppimismenetelmien optimointiin eniten käytetty menetelmä on Adam (*eng. adaptive moment estimation*), missä käytetään eksponentiaalisesti liukuvaa keskiarvoa gradienttien en-

[10]Huomaa, että oppimisnopeuteen vaikuttaa gradientin lisäksi askelkoko η, joka pysyy edelleen mukana erillisenä parametrina. Askelkokoa skaalataan muuttuvien gradienttien avulla.

simmäisiin ja toisiin momentteihin[11] ennen parametrien päivittämistä. Adamin suosioon on vaikuttanut empiiristen näyttöjen lisäksi sen suoraviivainen implementaatio ja laskennallinen tehokkuus sekä soveltuvuus haastaviin moniulotteisiin eikonveksisiin optimointiongelmiin [120, 80].

AMSGrad on perinteisen gradienttimenetelmän variantti, joka kuuluu eksponentiaalista liukuvaa keskiarvoa hyödyntäviin menetelmiin. On osoitettu, että on olemassa yksinkertainen yhden muuttujan konveksinen optimointiongelma, missä aiemmat tämän luokan optimointimenetelmät, kuten Adam, eivät konvergoitu. Tällöin optimointimenetelmällä on perustavanlaatuinen ongelma eli sen luotettavuus kärsii. Tähän on syyksi todettu se tapa, jolla eksponentiaalisesti liukuvaa keskiarvoa on hyödynnetty gradienttien painottamisessa ja parametrien päivittämisessä. Se ei ole säilyttänyt tietoa edellisten iteraatioiden gradienteista riittävän pitkään mukana, joka on osaltaan johtanut gradienttien häviämiseen edetessä iteraatiosta toiseen. Toisin sanoen menetelmillä on ollut liian lyhyt muisti: parametrien päivitysten tekeminen useampaan kuin vain muutamaan edelliseen gradienttiin takaa konvergoitumisen. AMSGrad on Adamista johdettu muunnos, missä edellä esitettyä ongelmaa ei ole ja sen kompleksisuus, laskennallinen tehokkuus sekä empiirinen suorituskyky säilyvät kilpailukykyisenä [118, 120]. Lisäksi konvergoitumisen nopeuteen ja laatuun vaikuttaa syväoppimismallin sisäisten parametrien alustusmekanismi eli miten optimointimenetelmän käynnistävä aloituspiste muodostetaan sekä mitä aktivointifunktiota käytetään alustusmekanismin kanssa. Empiiriset tutkimukset ovat osoittaneet, että esimerkiksi sigmoid-aktivointifunktiota tulee välttää standardinormaalijakaumalla[12] alustettujen parametrien yhteydessä [53].

Kun dataa ei ole esimerkiksi ulottuvuuksien puolesta tai muusta syystä ole tarpeeksi saatavilla, on syytä välttää ylisovittumista. Sen välttämiseksi syväoppimismallien optimointiin on kehitetty erilaisia keinoja, joiden tarkoituksena on rajoittaa niiden mallinnuskykyä. Ylisovittumista välttäviä tekniikoita kutsutaan toisinaan sääntelyksi eli regularisoinniksi (*eng. regularization*). Suoraviivaisin tapa on pitää syväoppimismallin arkkitehtuuri minimalistisena, johon tässäkin työssä pyrittiin

[11] Momentti on tilastotieteessä käytetty tunnusluku todennäköisyysjakaumien luonnehtimiseen, joka määritellään jakauman (tai satunnaismuuttujan) potenssiksi. Esimerkiksi jakauman keskusmomentit $\mathbb{E}[(X-\mu)^r]$, kun $r = 1, 2, 3, 4$, ovat odotusarvo, varianssi, vinous ja huipukkuus, vastaavasti. Adamin tapauksessa gradientit ovat origomomentteja eli $\mathbb{E}(X^r)$, missä $r = 1, 2$.

[12] Standardinormaalijakauma on muotoa $\mathcal{N}(0, 1)$.

tilastotieteestäkin tutun parsimoniaperiaatteen[13] (*eng. parsimony principle*) nojalla [54]. Mallin opetettavien parametrien määrä eli kapasiteetti (*eng. capacity*) pidetään pienenä, jota voidaan kontrolloida laskentayksiköiden ja piilokerrosten määrällä. Eräs toinen tapa rajoittaa syväoppimismalleja on säännellä painokertoimien suuruutta oppimisen aikana, jotta painokertoimien arvot pysyvät pieninä ja niiden hajonta pysyy tasaisempana[14]. Tämä on mahdollista lisäämällä kustannusfunktioon sakko- eli regularisointitermi, joka suurien painokertoimien kohdalla kasvattaa kokonaiskustannusta. Jos merkitään regularisointitermiä $\mathcal{R}(\cdot)$, saadaan kustannusfunktio yleiseen muotoon $\mathcal{L}(\mathbf{x}, \mathbf{x}') + \lambda\mathcal{R}(\cdot)$, missä regularisoinnin vaikutusta säädellään hyperparametrilla λ. Regularisointitermi $\mathcal{R}(\cdot)$ voi olla muun muassa $||\mathbf{W}||_p$, missä $p = 1, 2$. Edeltävässä tapauksessa, jota kutsutaan L_1-regularisoinniksi, lasketaan kaikkien painokertoimien absoluuttiset arvot yhteen ja jälkimmäisessä vastaavasti (L_2-regularisointi) painokertoimien neliöt lasketaan yhteen. Painokertoimien lisäksi myös vakiotermejä tai laskentayksiköiden ulostuloja voidaan rajoittaa, jolloin autoenkooderia sanotaan harvaksi (*eng. sparse autoencoder*), ja kun piilokerrosten aktivointifunktioiden osittaisderivaattoja[15] rajoitetaan, syntyy kontraktoiva autoenkooderi (*eng. contractive autoencoder*). Lisäksi gradientin L_2-normille voidaan asettaa yläraja, jonka ylitettäessä osittaisderivaatat normalisoidaan ehdon täyttymiseksi, tai gradientin arvolle voidaan yksinkertaisesti antaa minimi- ja maksimiarvot. [28, 21, 9].

Syväoppimismallien optimointi pienikokoisilla datajoukoilla voi johtaa myös siihen, että kuvauksen oppimisen sijaan malli oppii muistamaan opetusdatajoukon, jolloin malli ei yleisty ennalta tuntemattomille havainnoille. Lisäksi pienistä datajoukoista on hankalampaa approksimoida dataa generoivaa prosessia ja syöteavaruutta, koska harvat datajoukot eivät ole sileitä (*eng. smooth*). Eräs tapa kasvattaa mallin antamien tulosten luotettavuutta ja yleistettävyyttä dataa ollessa rajallisesti on hyödyntää k-kertaista ristiinvalidointia (*eng. k-fold cross-validation)* optimoinnin aikana. Siinä opetusdatajoukko jaetaan k:hon yhtä suureen osaan, jonka jälkeen mal-

[13]Parsimoniaperiaate tunnetaan myös Occamin partaveitsenä, jonka mukaan kilpailevista, yhtä selitysvoimaisista malleista, tulee valita yksinkertaisin. Lisäksi tulee muistaa, että mallin tulee olla laskennallisesti kevyt langattomien sensoriverkkojen asettamien rajoitteiden vuoksi.

[14]Pienet vaihtelut suurissa painokertoimissa voivat esimerkiksi aiheuttaa merkittäviä muutoksia syväoppimismallin ennustuksiin, jolloin mallista tulee epävakaa.

[15]Tällöin $\mathcal{R}(\cdot) = ||\mathbf{J}||_F$, missä $\mathbf{J}$ on syväoppimismallin aktivointifunktioiden osittaisderivaatat muuttujien suhteen sisältävä Jacobin matriisi ja F on Frobeniuksen normi, missä matriisialkioiden neliöiden summasta lasketaan neliöjuuri.

li optimoidaan $k - 1$ osalla dataa jäljelle jäävän osan jäädessä validoinnille. Samaa toistetaan niin kauan, että kukin yksittäinen osa on ollut validointidatana. Mallin suorituskyky saadaan näiden proseduurien keskiarvona. Ristiinvalidointi varmistaa, että opetusdatajoukon kullakin havainnolla on mahdollisuus päätyä myös validointidatajoukkoon [28]. Lisäksi eräs yksinkertainen, mutta tehokas tapa lisätä syväoppimismallin robustisuutta on tehdä opetusdatajoukosta kohinaisempi (*eng. noise*) mallia optimoidessa. Käytännössä satunnaisen kohinan lisääminen opetusdatajoukkoon kasvattaa näennäisesti opetusdatajoukon kokoa: optimoinnin aikana malli oppii jatkuvasti hieman erilaisista syötteistä. Kohinaa voidaan syötteiden lisäksi lisätä myös arkkitehtuurin eri osiin, kuten painokertoimiin, laskentayksiköiden ulostuloihin ja gradientteihin vastavirta-algoritmin aikana. Kohinaisesta tai korruptoituneesta datan esitysmuodosta alkuperäistä syötettä rekonstruoiva autoenkooderi tunnetaan myös kohinanpoistajana (*eng. denoising autoencoder*) [21, 9].

Ylisovittumista voidaan ehkäistä myös yksinkertaisella tekniikalla, missä syväoppimismallin optimoinnin aikana otetaan neuroneita pois käytöstä *(eng. dropout)* niihin saapuvat yhteydet eli painokertoimet mukaan lukien. Käytännössä valitaan satunnaisesti neuroneita, joiden aktivaatiot asetetaan nollaksi. Tällä tekniikalla pyritään ehkäisemään sattumanvaraisten vähemmän merkityksellisten riippuvuuksien syntyä syväoppimismallin piilokerroksilla eli kasvattamaan todennäköisyyttä, että malli kaappaa vain tärkeimpiä datan todellista luonnetta kuvaavia suhteita [28].

5 Data ja metodologia

Luku 5 käsittelee työn empiiristä osuutta: aliluvussa 5.1 esitellään työssä käytetty Bot-IoT -datajoukko ja eritellään, miten sitä sovellettiin syväoppimismenetelmissä. Työssä implementoituihin syväoppimismenetelmiin, niihin liittyviin yksityiskohtiin ja lopulta valittuihin konfiguraatioihin pureudutaan aliluvussa 5.2.

5.1 Data

Anomaliapohjaisten tunkeilijan havaitsemisjärjestelmien kehittämiseen ja niiden arvioimiseen tulee käyttää laadukkaita, runsaita ja monipuolisia tavoitemerkittyjä datajoukkoja (*eng. labeled dataset*). Silti monissa tuoreissakin tutkimuksissa menetelmien testaamiseen on tietoisesti hyödynnetty käyttökelpoisimpien vaihtoehtojen sijasta epärealistisia, vanhentuneita ja erilaisia muita ongelmia sisältäviä datajoukkoja jopa kahden vuosikymmenen takaa, jotka eivät edusta nykypäivän verkkoliikennettä ja verkkohyökkäyksiä. Langattomiin verkkoihin pohjautuvia käyttökelpoisia datajoukkoja tunkeilijan havaitsemisjärjestelmien testaamiseen on toisaalta harvassa [16, 15, 41, 119, 83]. Hiljattain Koroniotis ym. [83] julkaisivat ensimmäisen realistisen IoT-sovelluksissa esiintyviä verkkoja ja verkkoliikennettä jäljittelevän datajoukon nimeltään Bot-IoT[1], jota tässä työssä myös hyödynnetään. Eräs toinen varteenotettava julkinen datajoukko on Kolias ym. [82] esittelemä 802.11-verkkoon (WLAN) pohjautuva AWID[2]. Se ei kuitenkaan suoranaisesti jäljittele IoT-ympäristöä.

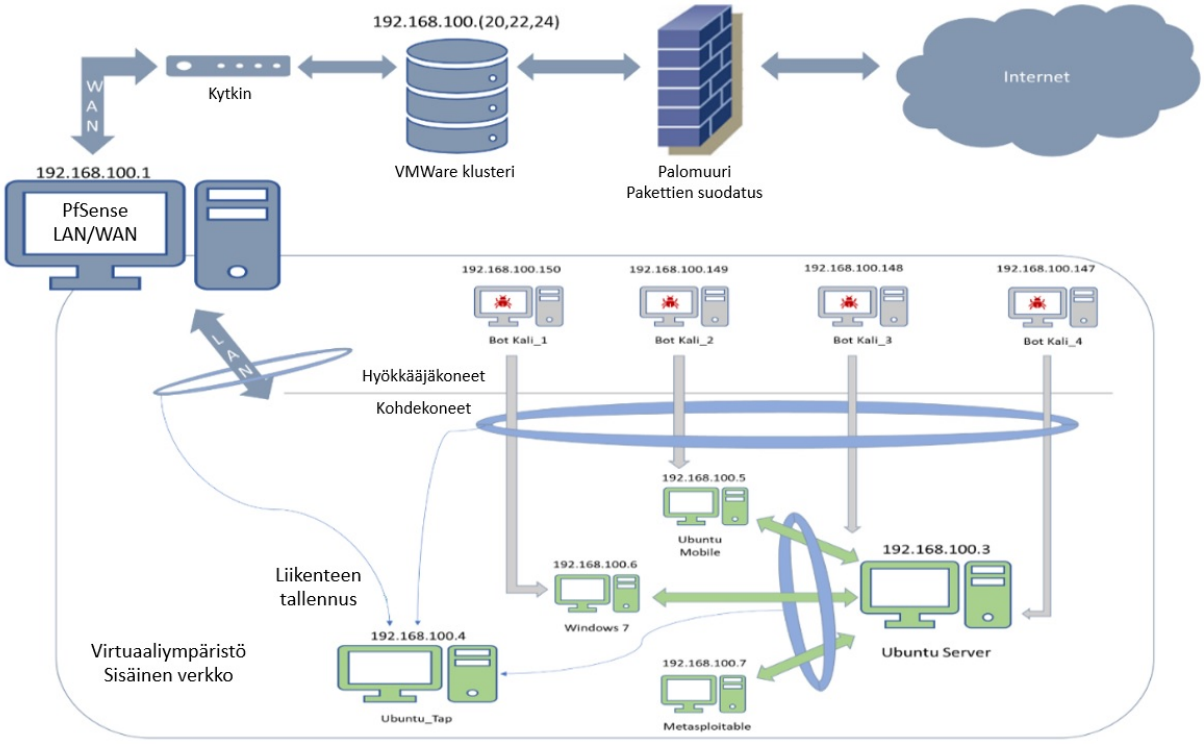
Bot-IoT datajoukon luontiin käytetty testiympäristö koostui yhteensä yhdeksästä virtuaalikoneesta eri käyttöjärjestelmillä: neljä Kalilla varustettua hyökkääjäkonetta, Ubuntu Server, Ubuntu Mobile ja Windows 7 -koneet eli palvelin, mobiililaite ja kuluttajakone, Metasploitable-kone[3] ja Ubuntu Tap -kone verkkoliiken-

[1]Lisätietoa artikkelissa [83] ja `https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot_iot.php`.

[2]Lisätietoa artikkelissa [82] ja `http://icsdweb.aegean.gr/awid/index.html`. Datajoukosta on tulossa uusi versio vuoden 2020 alussa, jolloin data jaetaan myös pakettimuodossa.

[3]Metasploitable on useita tietoturvahaavoittuvuuksia omaava virtuaalikone, jolla voidaan harjoitella esimerkiksi penetraatiotestausta. Lisätietoa `https://github.com/rapid7/metasploitable3`.

teen kuunteluun ja tallentamiseen. Ympäristössä Kali-koneet, jotka jäljittelivät IoT-ympäristössä toimivaa kaapatuista laitteista eli boteista koostuvaa bottiverkkoa (*eng. botnet*), suorittivat bottiverkolle tässä kontekstisssa tyypillisiä hyökkäyksiä palvelimeen, mobiililaitteeseen, kuluttajakoneeseen ja Metasploitableen. Suoritettuja hyökkäyksiä olivat i) tietojen urkinta (*eng. probing attacks*) porttiskannauksella ja käyttöjärjestelmätietojen kalastelulla (*eng. OS fingerprinting*), ii) yleiset ja protokollakeskeiset (hajautetut) palvelunestohyökkäykset ja iii) tietojen varastaminen murtautumalla ja näppäilytallentimella (*eng. keylogger*). Pakettien generointiin ja kaappaamiseen käytettiin Ostinatoa ja Wiresharkin terminaaliversiota tsharkia, vastaavasti[4]. Kuva 5.1 havainnollistaa testiympäristöä muine komponentteineen [83].



Kuva 5.1: Bot-IoT testiympäristö.

IoT-ympäristöä jäljittelevän liikenteen generoimiseksi testiympäristössä käytettiin laitteiden ja palvelinten välisen kommunikaation alustamista helpottavaa väliohjelmistoa (*eng. middleware*) Node-REDiä[5], jonka avulla luotiin erilaisia sensorei-

[4]Lisätietoa `https://ostinato.org/` ja `https://www.wireshark.org/docs/man-pages/tshark.html`.

[5]Lisätietoa `https://nodered.org/`.

ta sisältävä älykästä kotia jäljittelevä ympäristö. Tarkemmin eriteltynä testiympäristössä simuloitiin ilmanpaineesta, kosteudesta ja lämpötilasta tietoa generoivaa sääasemaa, jääkaapin lämpötilaa mittaavaa ja säätävää älykästä jääkaappia, liikkeellä ohjattavaa valaistusta, etänä avattavaa autotallin ovea ja ilmalämpöpumpun tarvittaessa aktivoivaa termostaattia. IoT-laitteet pitivät yhteyttä M2M-kommunikointiin (*eng. machine-to-machine*) kehitetyllä vähän resursseja vaativalla MQTT-protokollalla (*eng. message queuing telemetry transport*) palvelimeen, joka viesti edelleen AWS IoT-pilvipalvelun[6] kanssa [104].

Verkkoliikennepohjaisten tunkeilijan havaitsemisjärjestelmien testaamiseen tarkoitettujen datajoukkojen sisältämät havainnot ovat tyypillisesti joko verkkoliikennettä kuvailevaa metatietoa (*eng. flow-based*), raakaa pakettidataa (*eng. packet-based*) tai jotakin muuta, kuten pakettidatasta tai metatiedoista edelleen johdettuja muuttujia tai lisätietoa verkkoliikenteen ulkopuolelta, kuten lokitietoja. Edellä esitetystä testiympäristöstä kerättiin liki 70 gigatavun edestä pakettidataa .pcap-muodossa sisältäen yli 73 miljoonaa havaintoa. Pakettidata muunnettiin Argus-ohjelmalla[7] 16.7 gigatavun kokoiseksi .csv-tiedostoksi, missä havainnot esiintyvät metatietoina. Redusoidun version kokoamiseksi muunnetusta datajoukosta irrotettiin 5 prosenttia, jolloin jäljelle jäi noin gigatavun verran dataa sisältäen 3668522 kappaletta 46 muuttujaa omaavaa havaintoa (alkioita yhteensä 168752012). Lopuksi Koroniotis ym. [83] laativat vielä karsitumman version irrottamalla hyökkäysten tunnistamisen kannalta 10 hyödyllisintä piirrettä eli muuttujaa Pearsonin korrelaatiokertoimeen ja Shannonin yhteisentropiaan perustuen. Tästä datajoukosta on saatavilla myös valmiina opetus- ja testidatajoukot. Tässä työssä lähdetään liikkeelle kuitenkin 46 muuttujan omaavalla datajoukolla. Datajoukon sisältämät muuttujat ovat kuvattu liitteen A taulukossa A.1.

Eräs merkittävä tekijä Bot-IoT -datajoukossa anomalioiden havaitsemismenetelmien kehittämisen kannalta on hyökkäys- ja normaalidatan määrä. Redusoidussa versiossa on kaiken kaikkiaan 477 normaaliksi luokiteltua havaintoa hyökkäyksiksi luokiteltujen havaintojen ollessa 3668045. Normaalin profiilin luominen tulee onnistua täten 477 havainnolla. Tästä huomaa, että datajoukko soveltuu paremmin niiden menetelmien luomiseen, joiden opetusesimerkeissä on mukana myös hyökkäysdataa ja jotka luokittelevat havaintoja. Esimerkiksi tunnettuja hyökkäyksiä luokittelevan ohjatusti opetetun menetelmän kehittämiseen datajoukko sopii hyvin. Ano-

[6]Lisätietoa `https://aws.amazon.com/iot-core/features/`.
[7]Lisätietoa `https://openargus.org/`

malioiden havaitsemismenetelmät toisaalta ovat usein puoliohjattuja niin, että niiden opettamiseen käytetään vain normaaliksi luokiteltua dataa. Tässä työssä käytetään vain redusoitua datajoukkoa, vaikka havaintoja on suhteellisen vähän — se on muuttujien osalta monipuolisempi kuin alkuperäinen datajoukko[8].

Kirjallisuudessa datajoukko on esiintynyt suhteellisen harvoin, mikä on ymmärrettävää datajoukon nuoren iän vuoksi. Hyökkäyksien tunnistamista käsitteleviä tutkimuksia datajoukon avulla on datajoukon tekijöiden lisäksi kahdeksan, joista kaksi [142, 78] ovat katsauksia ja yksi liittyy väärinkäytöspohjaisiin menetelmiin [131]. Kahdessa työssä [45, 69] käytettiin syväoppimismenetelmiä, mutta ohjatusti hyökkäysten luokitteluun ja yhdessä niistä [5] keskityttiin (ohjatusti) vain palvelunestohyökkäysten tunnistamiseen. Kahdessa työssä toisaalta käytettiin puoliohjattuja menetelmiä, mutta yhdessä [2] otettiin alkuperäisestä datajoukosta satunnaisesti eri kokoisia näytteitä tilastolliseen menetelmään ja koneoppimiseen perustuvan algoritmin[9] testaamiseksi ja toisessa [79] kehitettiin kaksiosainen koneoppimiseen perustuva menetelmä, jossa väärinkäytös- ja anomaliapohjaisten menetelmien tulokset yhdistettiin[10]. Myös datajoukon laatineiden tutkimuksessa [83] testattiin kone- ja syväoppimismenetelmillä hyökkäysten tunnistamista, mutta ohjatusti luokittelijoina. Edellä mainitut tutkimukset ja niiden tulokset eivät siis ole suoraan verrannollisia tähän työhön, mutta joitain huomioita niitä vasten voidaan tehdä mallinnuksen ja saavutettujen tulosten osalta, joihin palataan jatkossa.

Redusoitu datajoukko oli ladattavissa neljänä .csv-tiedostona, jotka yhdistettiin yhdeksi tiedostoksi. Tämän jälkeen datajoukko järjestettiin havaintojen aloitusaikojen *stime* suhteen ja niin, että tietyn *stime* omaavat havainnot olivat rivi-indeksin *pkSeqID* mukaisessa järjestyksessä. Data-analyysiin eli neuroverkkomallien syötteeksi valittiin liitteen A taulukkoon A.1 listatuista muuttujista 33 muuttujaa keskittyen tietoverkkoliikennettä kuvaaviin, jonkin lukuarvon saaviin muuttujiin, jolloin i) rivi-indeksi, lähettäjän ja vastaanottajan IP-osoitteet sekä portit, ii) ensimmäisen ja viimeisen tiedonsiirron aloitusajat ja iii) tekstimuotoiset lippumuuttujat, yhteyk-

[8]Eräänä ajatuksena oli täydentää redusoidun datajoukon sisältämää normaalia liikennettä alkuperäisen datajoukon liikenteellä (9543 havaintoa), mutta tämä ei onnistunut eriävien muuttujien vuoksi.

[9]Normaali profiili luotiin Gaussisen sekoitemallin (*eng. gaussian mixture model, GMM*) avulla ja poikkeamien tunnistus tiheysperusteisella lähimmän naapurin menetelmällä LOFilla (*eng. local outlier factor*).

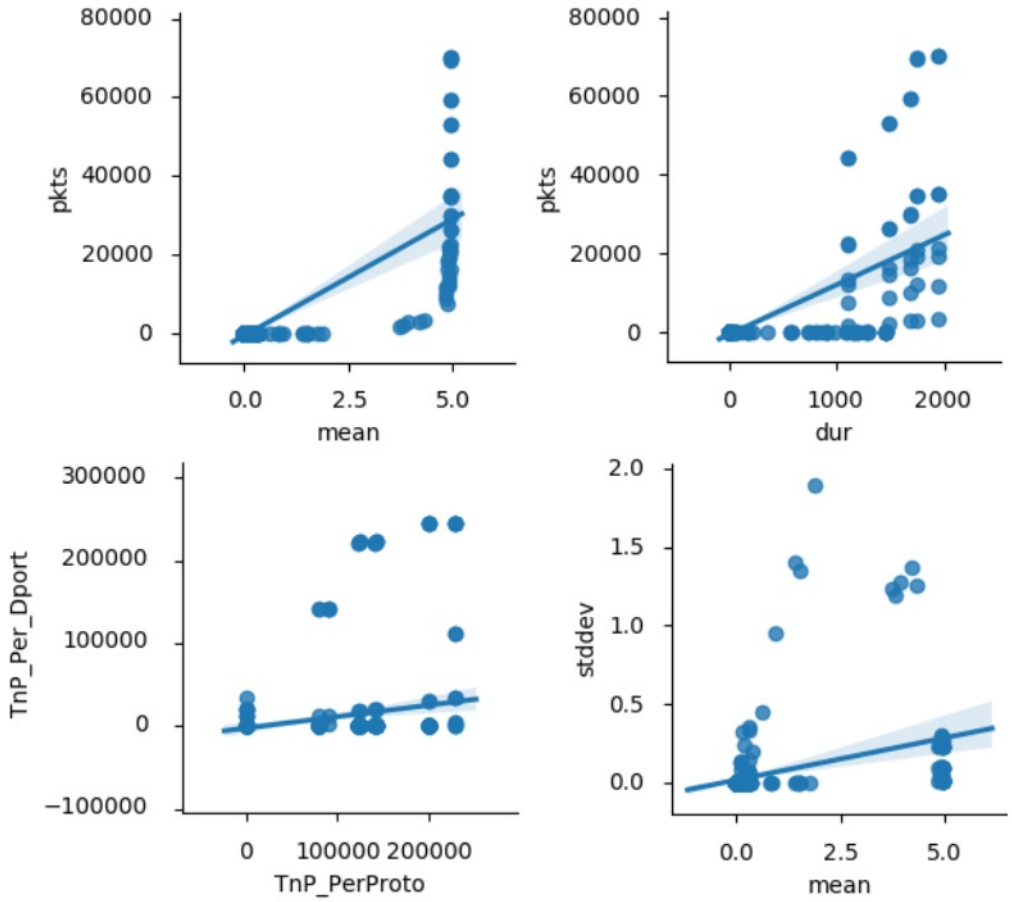
[10]Väärinkäytöspohjaisena menetelmänä käytettiin C5 päätöspuuta ja yhden luokan tukivektorikonetta (*eng. one-class support vector machine, one-class SVM*) poikkeamien tunnistamiseen.

sien tilat ja kuljetuskerroksen protokollat jäivät valittujen muuttujien ulkopuolelle[11]. Toisin sanoen data-analyysiin sisällytettiin tietoja i) lippumuuttujista, yhteyksien tiloista ja kuljetuskerroksen protokollista numeerisina kategorisina muuttujina, ii) lähetettyjen ja vastaanotettujen pakettien lukumääristä tavuineen sekä iii) yhteyksien kestoista ja niihin perustuvista erilaisista tilastollisista tunnusluvuista ja suhdemuuttujista taulukon A.1 mukaisesti. Näin ollen normaalin profiilin luomisen lähtökohtana oli 33 muuttujaa omaavat 477 havaintoa sisältäen yhteensä 15741 data-alkiota tai -pistettä. Kutakin havaintoa voidaan ajatella yhtenä vektorina 33-ulotteisessa Euklidisessa avaruudessa.

Yllä mainitusta normaalia liikennettä kuvaavasta 477 havainnosta edellä esitetyn järjestyksen mukaan viimeiset 50 jätettiin testidataksi, jolloin syväoppimismallien opettaminen perustui 427 havaintoa sisältävään opetusdatajoukkoon. Syväoppimismalleja optimoidessa opetusdatajoukko jaettiin vielä niin, että malleja opetettiin 327 havainnolla ja validoitiin 100 viimeisellä havainnolla. Eli optimoidessa normaalia profiilia mallinnettiin 327 havainnolla ja 100 havainnolla validoitiin muodostetun profiilin oikeellisuutta sekä tehtiin korjauksia tarpeen mukaan. Yleisesti ottaen opetus- ja validointidatajoukkojen keskinäinen suhde voi vaihdella 70:30:sta aina 90:10:een asti riippuen sovelluksesta: esimerkiksi 10 prosenttia validointidataa voi olla hyvinkin riittävä silloin, kun datan määrä on kertaluokkaa 10^6 tai enemmän. Tässä tapauksessa suhteesta muodostui likimain 77:23. Käytetty jakosuhde etsittiin muutamien iteraatioiden tuloksena jakosuhdetta vaihtamalla oppimiskäyriin ja pientä hyökkäysdatajoukkoa vastaan tehtyihin testeihin perustuen. Datajoukon suhteellisen pienuuden myötä ristiinvalidoinnin käyttöönotto oli oikeutettua, mutta siihen ei lopulta siirrytty toimivan jakosuhteen löydyttyä. Monissa tapauksissa onkin niin, että yksinkertainen jako on riittävä [28]. Ristiinvalidointi vaatii myös huomattavasti enemmän laskennallisia resursseja, mikä ei ollut osaltaan edullista.

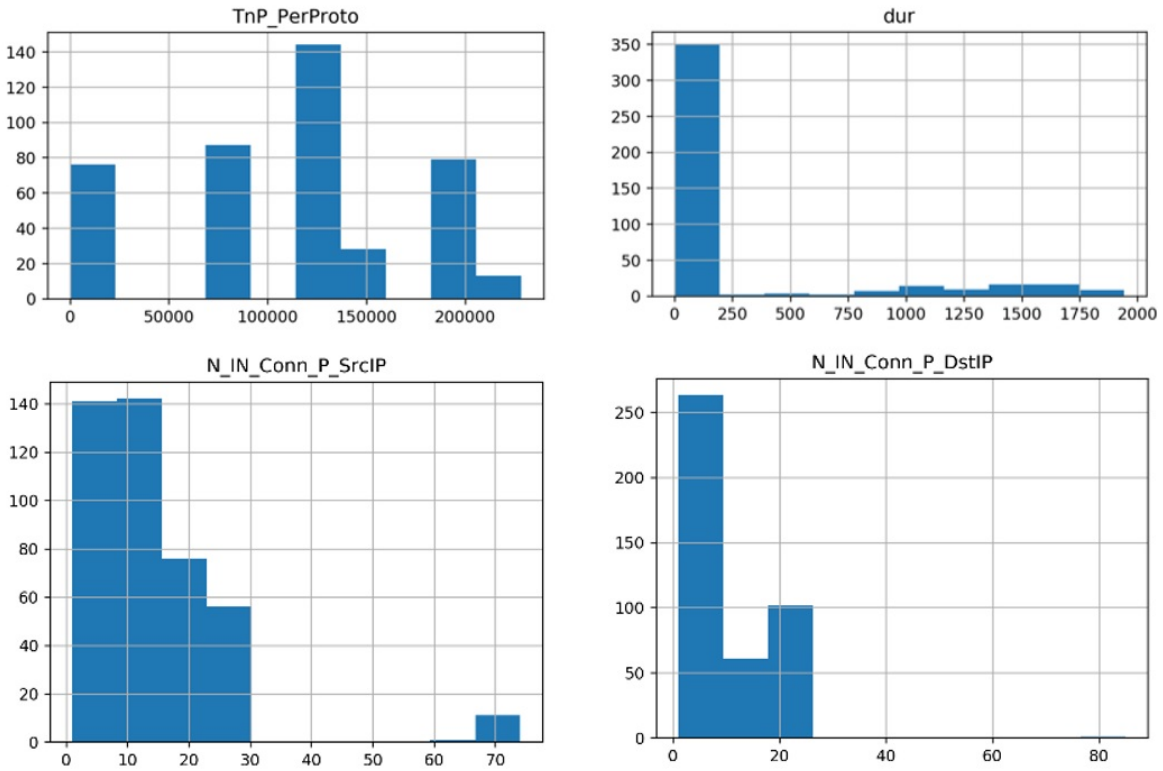
Tarkastellaan seuraavaksi näiden 427 havainnon perusteella muuttujien luonnetta ja niiden välisiä suhteita. Kuva 5.2 havainnollistaa eräiden datassa esiintyvien muuttujien keskinäisiä riippuvuuksia.

[11] Liitteen A taulukon A.1 muuttujat 1, 2, 3, 5, 7, 8, 9, 10, 13 ja 15. Huomaa, että viimeisimmät muuttujat 44, 45 ja 46 pelkästään luokittelevat havaintoja, eli ne jäävät myöskin pois.



Kuva 5.2: Eräiden muuttujien välisiä epälineaarisia suhteita opetusdatassa.

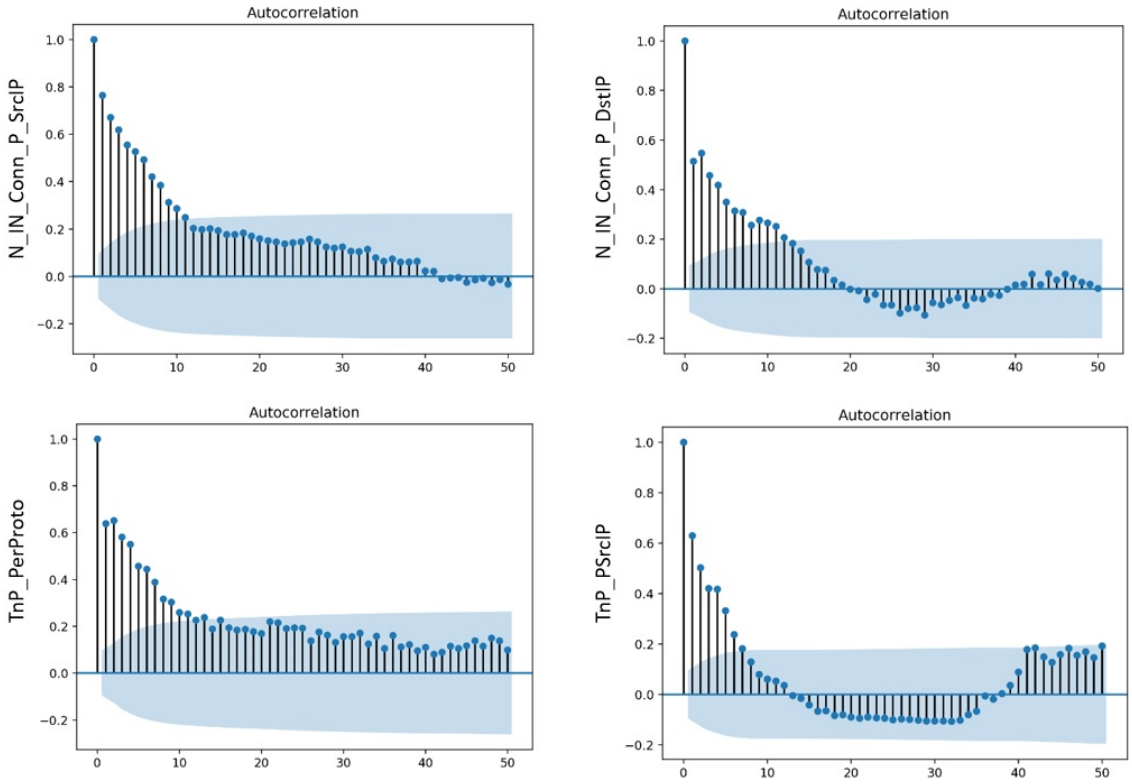
Kuvan 5.2 sisältämien kuvaajien x- ja y-akselit kuvaavat muuttujien saamia arvoja. Esimerkiksi vasemmassa ylälaidassa olevassa kuvaajassa muuttuja *mean*, joka on siirrettyjen pakettien siirtoaikojen keskiarvo, saa arvoja nollasta viiteen ja siirrettyjen pakettien lukumäärää edustava *pkts* saa arvoja aina 80 000 asti. Lisäksi kuvaajiin on sovitettu lineaarinen regressiomalli. Kuten nähdään, ei voida olettaa, että datassa esiintyvien muuttujien väliset suhteet ovat lineaarisia, toisistaan riippumattomia tai muuten yksiselitteisiä. Lisäksi kuva 5.3 havainnollistaa eräiden muuttujien jakaumia histogrammien avulla, missä x-akseli kuvaa kyseisen muuttujan saamia arvoja ja y-akseli niiden frekvenssejä. Kuvan 5.3 perusteella ei voida myöskään olettaa, että muuttujat ovat normaalisti tai identtisesti jakautuneita.



Kuva 5.3: Eräiden muuttujien jakaumia opetusdatassa.

Lisäksi kuvan 5.4 esittämät kuvaajat eräiden muuttujien autokorrelaatioista[12] antavat viitteitä siitä, että datassa on myös temporaalinen elementti eli joidenkin muuttujien havainnot ovat ajasta riippuvaisia ainakin kymmenenteen havaintoon asti 95 prosentin luottamusvälillä. Näin ollen voidaan todeta, että datan luonne kokonaisuudessaan edellyttää sellaisten menetelmien käyttöä, joilla on mahdollista mallintaa datassa esiintyviä epälineaarisia suhteita ja monimutkaisia riippuvuuksia.

[12] Autokorrelaatio kuvaa yksittäisen muuttujan saamien arvojen välistä riippuvuutta eli miten moneen edeltävään havaintoon nykyinen arvo perustuu.



Kuva 5.4: Eräiden muuttujien autokorrelaatioita opetusdatassa.

5.2 Implementaatiot

Tässä työssä implementoitiin neljä syväoppimiseeen perustuvaa anomalioiden havaitsemismenetelmää, joiden tehtävänä oli havaita verkkohyökkäyksiä Bot-IoT -datajoukosta. Anomalioiden havaitsemisongelma voidaan määritellä siten, että datajoukon $\mathcal{D} = \{\mathbf{x}_n\}_{n=1}^{N}$, parametrisen mallin $\mathcal{F}_\theta$ ja kynnysarvon ϵ avulla tulee löytää anomaliajoukko

$$\mathcal{A} = \{\mathbf{x}_n \in \mathcal{D} \mid \mathcal{L}(\mathcal{F}_\theta(\mathbf{x}_n), \mathbf{x}_n) > \epsilon\}, \tag{5.1}$$

missä $\mathcal{L}$ on tämän työn kontekstissa mallin rekonstruktiovirhe. Neljä implementoitua mallia olivat eteenpäin syöttävä ja konvoluutionaalinen autoenkooderi sekä LSTM- ja GRU-pohjaiset sekvenssioppijat. LSTM:stä käytetään perinteistä versiota.

Tässä työssä käytettiin piilokerroksilla muun muassa tasajakaumaan pohjautu-

vaa He-alustusta[13] ELU-aktivointifunktioilla suositusten [51, 62] mukaisesti. Lisäksi malleissa hyödynnettiin painokertoimien L_2-regularisointia, normaalijakautunutta kohinaa syötteissä keskihajontaa säätäen ja kaikkien mallien optimointiin käytettiin Adamin sijasta AMSGradia oletusparametrein. Opetusdatajoukon käsittelyyn otettiin mukaan myös esiprosessointimenetelmiä, koska joidenkin muuttujien minimi- ja maksimiarvot liikkuivat kymmenissä, kun taas toisten muuttujien arvot heilahtelivat nollasta ja kymmenistä aina kertaluokkaan 10^8 asti. Laajat vaihteluvälit arvoissa ja heilahtelut arvojen suuruuksissa voivat johtaa poikkeuksellisen suuriin gradientteihin optimoinnin aikana, jolloin oppiminen voi olla hidasta tai epävakaata [21, 28]. Opetusdataa prosessoitiin ottamalla kaikista arvoista logaritmi, normalisoimalla kukin muuttuja välille $[0, 1]$ tai standardisoimalla z-arvon (*eng. z-score*) mukaisesti. Lisäksi normalisointia kokeiltiin logaritmisista arvoista. Huomaa, että validointi- ja testidatajoukkojen normalisointiin käytettiin opetusdatajoukon minimejä ja maksimeja, ja standardisointiin vastaavasti opetusdatajoukon keskiarvoja ja -hajontoja [77]. Kustannusfunktioista kokeiltiin taulukon 3.1 funktioiden[14] lisäksi neliösumman neliöjuurta ja L_∞-normina tunnettua Chebyshev-etäisyyttä, joka määritellään $\max(|\mathbf{x}' - \mathbf{x}|)$. Chebyshev-etäisyydeksi muodostuu suurin erotus ulottuvuuksien eli koordinaattiakselien suhteen[15].

Syväoppimismallien sopivien arkkitehtuurien etsimiseksi kerralla annettavien opetusesimerkkien ja optimointikierrosten määrää myös vaihdeltiin. Kertauksen vuoksi kierroksilla (*eng. epochs*) tarkoitetaan koko opetusdatajoukon yhtä läpikäyntiä ja opetusdatan osajoukon (*eng. mini-batch*) perusteella tehdään aina uusi parametripäivitys. Jos esimerkiksi opetusdatajoukon koko on 30 havaintoa ja kerralla annettavan opetusdatajoukon osajoukon koko on 10 havaintoa (*eng. batch-size*), päivitetään mallin parametreja kolme kertaa yhden kierroksen aikana. Implementoiduissa syväoppimismalleissa eri kapasiteettien testaaminen oli suoraviivaista, mutta konvoluutionaalisessa autoenkooderissa tuli kiinnittää enemmän huomiota suotimen kokoon (*eng. filter size*) ja askelkokoon (*eng. stride*), jotka viime kädessä määrittävät syntyvän piirrekartan koon. Huomaa, että tässä työssä toimitaan 1-ulotteisilla syötteillä 2-ulotteisten sijaan.

---

[13] Olkoon n piilokerroksen painokertoimien lukumäärä. He-alustus on tällöin muotoa $\mathcal{N}(0, \sqrt{\frac{2}{n}})$, joka taipuu tasajakaumaksi välille $[-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}}]$.

[14] Kosinimitan sijasta käytettiin kosinietäisyyttä.

[15] Oletetaan pisteet $(1, 2, 3)$ ja $(3, 4, 8)$ 3-ulotteisessa avaruudessa. Tällöin niiden välinen Chebyshev-etäisyys on $\max(|3-1|, |4-2|, |8-3|) = 5$.

Kullakin testatulla konfiguraatiolla syväoppimismallit optimoitiin vähintään 4 kertaa niiden stokastisen luonteen vuoksi ja lopulliset rekonstruktiovirheet kullekin havainnolle laskettiin testien keskiarvona[16]. Lisäksi kaavan 5.1 kynnysarvoksi ϵ otettiin tehtyjen optimointien viimeisten validointidatajoukkoa vastaan mitattujen rekonstruktiovirheiden keskiarvo. Toisin sanoen kynnysarvo pohjautuu syväoppimismallin kykyyn tunnistaa normaalia liikennettä validointidatajoukosta. Koska opetusdataa oli suhteellisen vähän, malleja ja erityisesti edellä esitetyn kynnysarvon ϵ toimivuutta testattiin ensin pientä hyökkäysdatajoukkoa vastaan, missä kunkin luokan hyökkäysdataa oli pieni määrä[17]. Tämä menettely toisaalta saattoi tuoda tietovuotoa (*eng. information leak*) ennalta tuntemattomista havainnoista malleihin niiden optimoinnin aikana. Tietovuodon määrä katsottiin kuitenkin vähäiseksi, koska koko datajoukossa hyökkäyksiä on miljoonia ja kunkin hyökkäyskategorian sisällä hyökkäysten toteuttamiseen datajoukon tekijät käyttivät erilaisia työkaluja muuttuvine parametreineen [83]. On siis epätodennäköistä, että valittu 579 havainnon otos pieneksi hyökkäysdatajoukoksi edustaisi kaikkia hyökkäyshavaintoja. Taulukossa 5.1 on eritelty valitut arkkitehtuurit suurempaa hyökkäysdatajoukkoa vastaan testaamista varten.

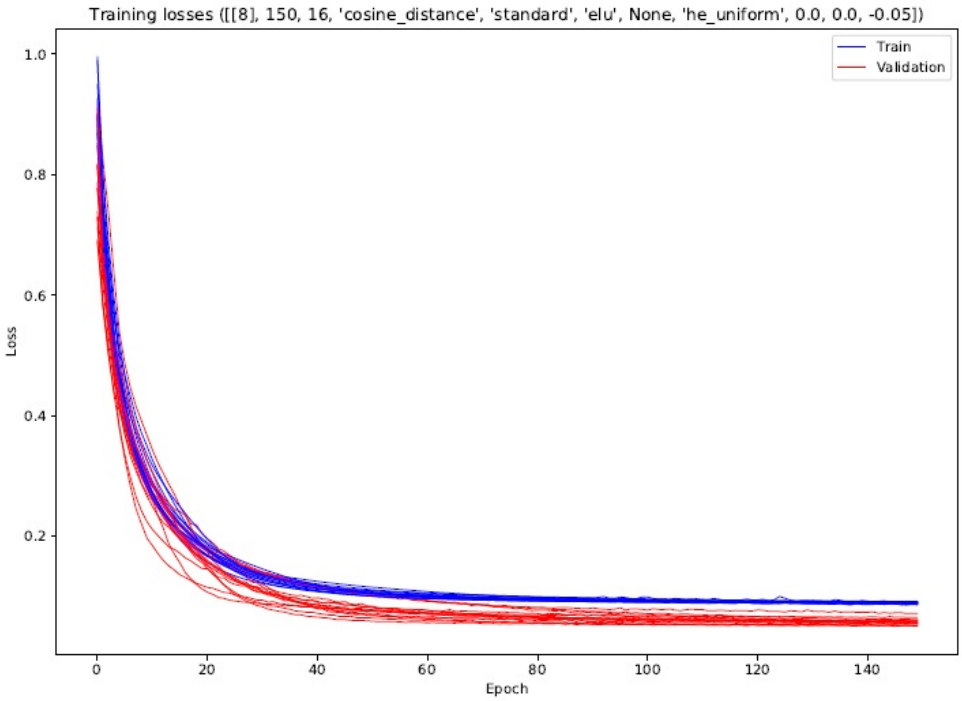
[16]Mallien optimoinnin yhteydessä testattiin kaiken kaikkiaan noin 1300 erilaista konfiguraatiota.

[17]Palvelunestohyökkäys (DoS: HTTP, TCP, UDP): 100 havaintoa kutakin, Tietojen urkinta (OS-tietojen kalastelu, porttiskannaus): 100 havaintoa kutakin, 73 näppäintallennushavaintoa ja 6 murtautumishavaintoa. Yhteensä 579 havaintoa.

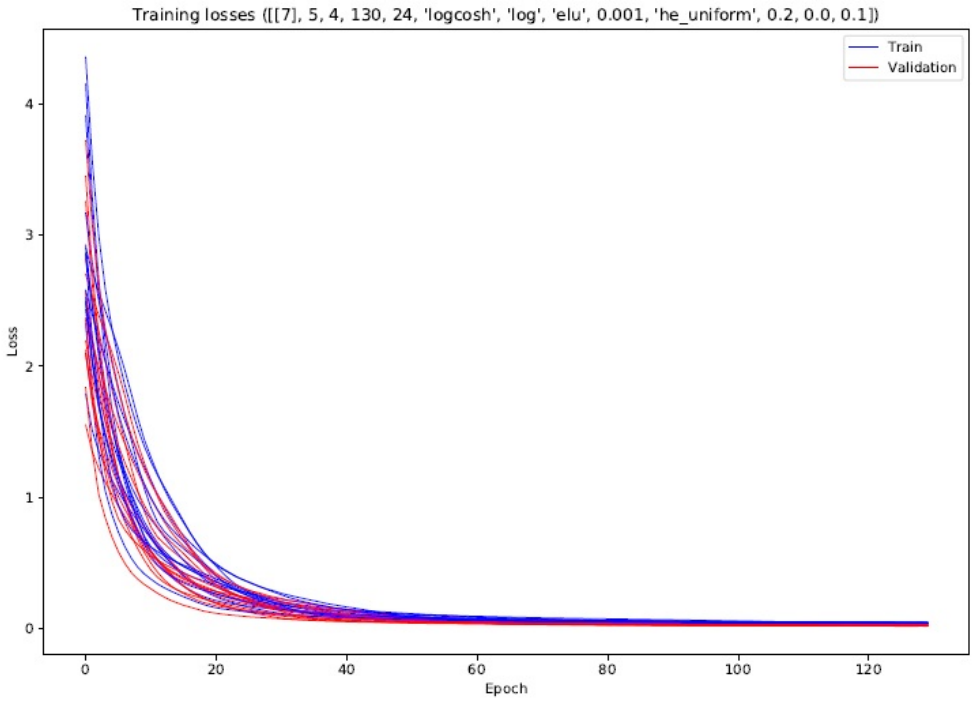
Taulukko 5.1: Valitut arkkitehtuurit syväoppimismalleille.

Malli	ANN-AE	CNN-AE	LSTM-AE	GRU-AE
Rakenne	33-8-33	33-8-33	33-6-33	33-6-33
epochs	150	130	200	200
batch-size	16	24	16	16
$\mathcal{L}$	Kosinietäisyys	logcosh	Kosinietäisyys	Kosinietäisyys
Esikäsittely	z-arvo	log	z-arvo	z-arvo
φ	ELU(0.5)	ELU(0.5)	tanh, σ	tanh, σ
$\mathcal{R}(\cdot) = L_2(\lambda)$	–	$\lambda = 0.001$	$\lambda = 0.001$	$\lambda = 0.001$
Alustus	*he_uniform*	*he_uniform*	*he_uniform*	*he_uniform*
noise	–	$\mathcal{N}(0, (0.2)^2)$	$\mathcal{N}(0, (0.1)^2)$	$\mathcal{N}(0, (0.1)^2)$
$\epsilon(1+\delta)$	$\delta = -0.05$	$\delta = 0.1$	$\delta = -0.1$	–
Parametrit	1691	330	1503	1085
Lisätiedot		*filters* = 7 *filter size* = 5 *stride* = 4		

Taulukossa 5.1 rakenne kertoo, että minkä ulottuvuuden omaavaan aliavaruuteen syöte muunnetaan autoenkooderissa. Eteenpäin syöttävässä (ANN-AE) ja konvoluutionaalisessa (CNN-AE) autoenkooderissa rekonstruktio tehdään kahdeksan ulottuvuuden esityksestä, kun taas toistuvissa malleissa (LSTM-AE ja GRU-AE) käytetään kuutta ulottuvuutta. Lisäksi toistuvissa malleissa käytetään niille ominaisia aktivointifunktioita ja niissä regularisointi kohdistuu vain syötteitä vastaaviin painokertoimiin. δ-arvolla on mahdollista painottaa laskettua kynnysarvoa eli käytännössä sillä voidaan joko nostaa tai laskea kynnysarvoa. Parametrit-sarake ilmoittaa kunkin mallin sisäisten parametrien lukumäärän. Konvoluutionaalisessa autoenkooderissa tulee antaa myös konvoluutiokohtaisia parametreja, jotka on esitetty lisätiedoissa. Siinä käytetään seitsemää suodinta ja viiteen muuttujaan kerrallaan tehdään konvoluutio neljän askeleen välein, jonka tuloksena syöte kutistuu niin ikään kahdeksaan ulottuvuuteen. Konvoluutionaalisessa autoenkooderissa käytettiin ali- ja ylinäytteistykseen vain konvoluutiota. Kuvat 5.5 ja 5.6 havainnollistavat eteenpäin syöttävän ja konvoluutionaalisen autoenkooderin oppimiskäyriä 15 optimoinnin tuloksena.

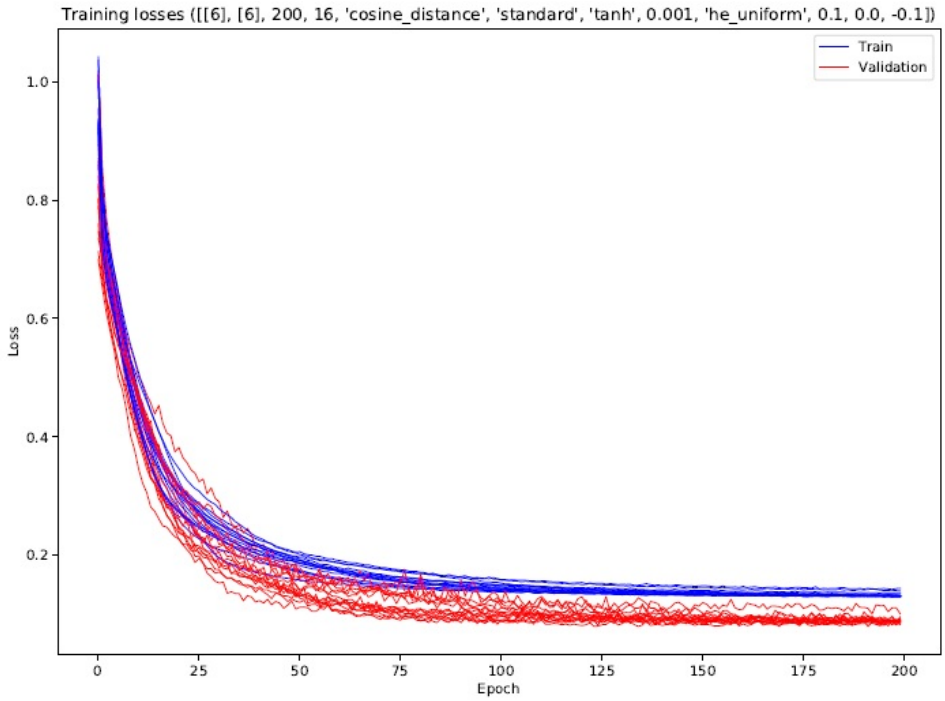


Kuva 5.5: ANN-AE -mallin oppimiskäyriä 15 optimoinnin jälkeen.

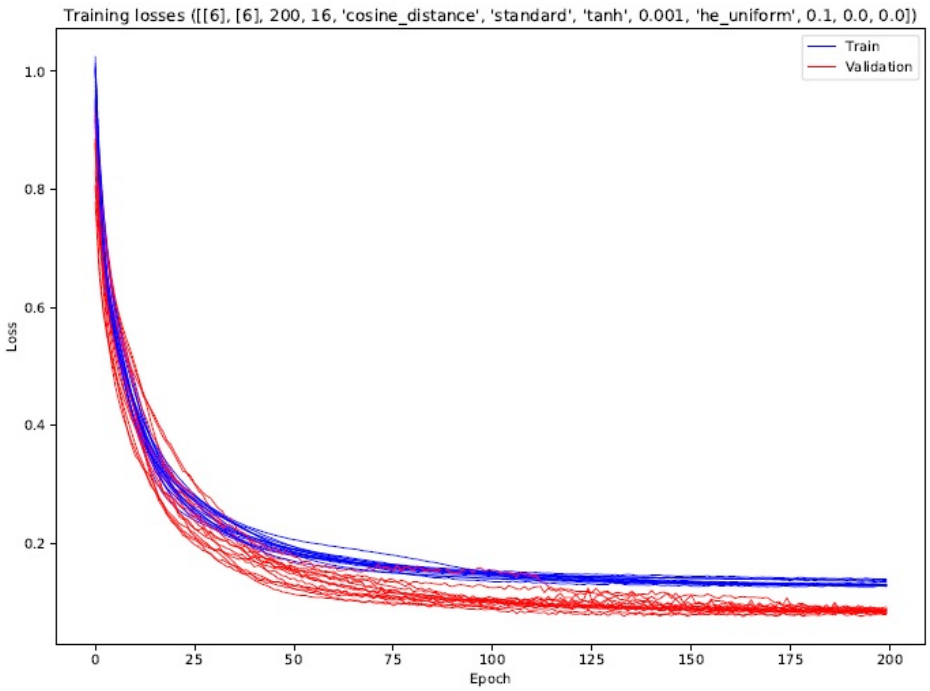


Kuva 5.6: CNN-AE -mallin oppimiskäyriä 15 optimoinnin jälkeen.

Kuvien 5.5 ja 5.6 oppimiskäyristä voi huomata, että mallit konvergoituvat nopeasti ja melko vakaasti kun huomioidaan, että kukin malli optimoitiin 15 kertaa. Eteenpäin syöttävän autoenkooderin oppimiskäyrässä validointidatajoukon rekonstruktiovirhe on hieman alempana kuin opetusdatajoukon, mikä voi antaa viitteitä siitä, että validointidatajoukko ei ole riittävän monipuolinen ja haastava. Ero on toisaalta pieni, kun tarkastellaan tarkemmin y-akselin arvoja. Konvoluutionaalisen autoenkooderin oppimiskäyrät kulkevat silminnähden päällekkäin eli opetus- ja validointidatajoukkon rekonstruktiovirheissä ei näytä olevan suuria eroja. Y-akselin skaala on tosin laajempi kuin eteenpäin syöttävän autoenkooderin kuvaajassa, joten mahdollinen pieni ero saattaa jäädä kuvasta näkymättä. Erot ovat joka tapauksessa marginaalisia eli niiden merkitys jää vähäiseksi. Toistuvien mallien oppimiskäyrät ovat kuvaajissa 5.7 ja 5.8.



Kuva 5.7: LSTM-AE -mallin oppimiskäyriä 15 optimoinnin jälkeen.



Kuva 5.8: GRU-AE -mallin oppimiskäyriä 15 optimoinnin jälkeen.

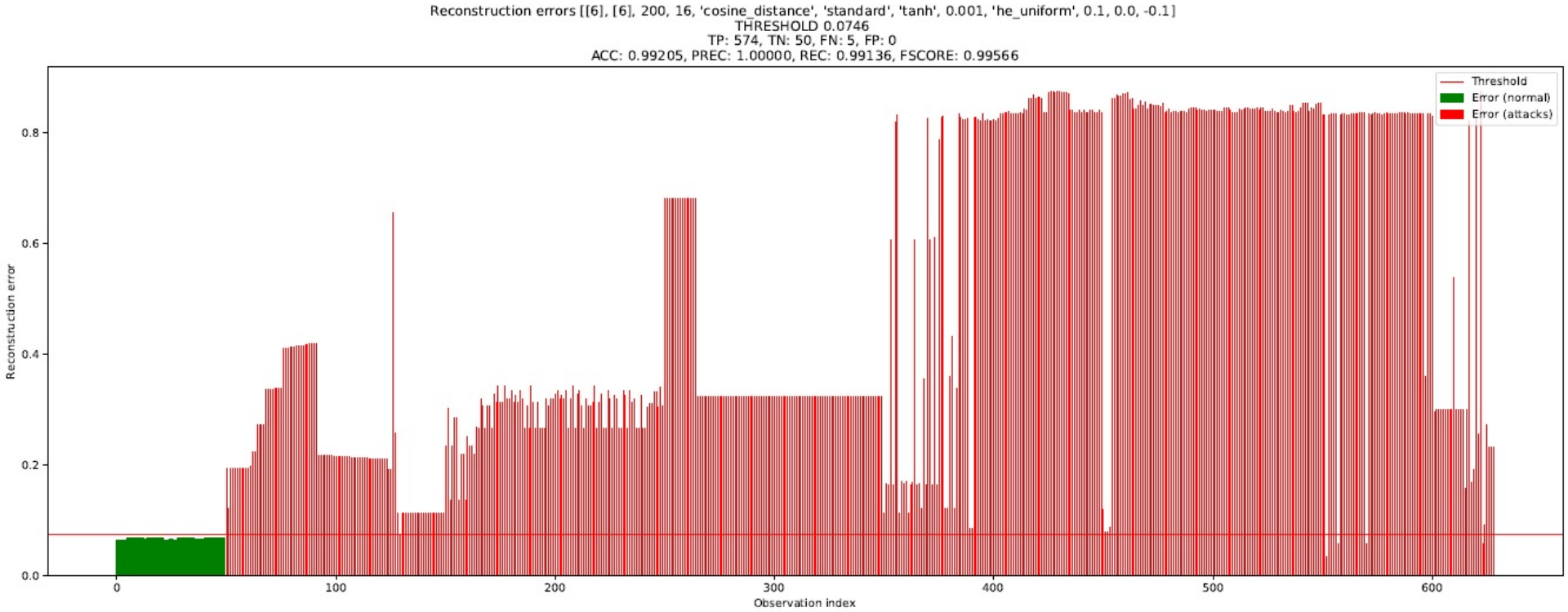
Toistuvissa malleissa voi havaita pientä vaihtelua, mikä johtuu mallien sisäisestä rakenteesta eli niiden kompleksisuudesta: lohkoissa tapahtuu useampia datamuunnoksia ja ne ovat luonteeltaan syviä arkkitehtuureja rekursiot avattaessa. Vaihteluista huolimatta mallit konvergoituvat samaan tapaan kuin eteenpäin syöttävässä autoenkooderissa eli opetus- ja validointidatajoukkojen rekonstruktiovirheiden ero on pieni kun huomioidaan y-akselin arvot. Oppimiskäyrien pohjalta kaikki syväoppimismallit ovat oletetusti mallintaneet normaalin liikenteen profiilin opetusdatajoukosta nopeasti ja riittävän vakaasti konvergoituen.

Kuvassa 5.9 on havainnollistava esimerkki LSTM-AE -mallin suorituskyvystä 15 toistolla aiemmin esiteltyä pientä hyökkäysdatajoukkoa vastaan sisältäen aiemmin testidataksi jätetyt 50 normaalia havaintoa. Kuvassa 5.9 on piirretty pylväin kullekin havainnolle laskettu rekonstruktiovirhe ja väreillä erotellaan normaalit havainnot hyökkäyksistä. Vaakasuora viiva kuvastaa kynnysarvoa, jolla erotellaan poikkeavat havainnot. Kuvaajan otsikossa näkyy edelleen käytetty kynnysarvo ja aiemmin esitetyt suorituskykymittarit, joihin ei tässä kohtaa pureuduta sen tarkemmin. Oleellista on huomata, miten tasaisena normaalin liikenteen rekonstruktiovirhe pysyy, mihin kohtaan kynnysarvo asettuu ja miten huonosti malli rekonstruoi normaalista poikkeavat havainnot eli miten hyvin malliin tarttuu poikkeamat. Tässä tapauk-

sessa LSTM-AE -mallilta jää viisi hyökkäystä 579:stä tunnistamatta, mikä koettiin tyydyttäväksi. Saman tyyppinen testi ja kuvaaja laadittiin kaikille malleille: konvoluutionaaliselta, eteenpäin syöttävältä ja GRU-sekvenssioppijalta jäi 28, 4 ja 7 hyökkäystä tunnistamatta, vastaavasti. Valinnat arkkitehtuureista lopullisia testejä varten tehtiin tämän tyyppisiin testeihin perustuen.

Työn aikana laadittu ja käytetty ohjelmakoodi on tehty Pythonilla (3.4.10), ja siihen perustuvia ohjelmointikirjastoja käyttäen. Työssä kehitetyt mallit implementoitiin Kerasilla[18] (2.2.4), joka on korkean tason ohjelmointikirjasto syväoppimismenetelmien soveltamiseen. Keras pohjautuu TensorFlowhon (tensorflow-gpu 1.14.0). Datan käsittelyyn hyödynnettiin paljon data-anlyysiin ja tieteelliseen laskentaan soveltuvia kirjastoja kuten pandasia (0.22.0) ja numpya (1.16.4) sekä kuvaajien piirtämiseen matplotlibia (2.2.4) ja seabornia (0.9.0). Lisäksi scipyyn (1.2.2) pohjautuva koneoppimiskirjasto scikit-learn (0.20.4) tarjosi valmiita implementaatioita erilaisille metriikoille ja suorituskykymittareille.

[18] `https://keras.io/`



Kuva 5.9: LSTM-AE -mallin suorituskyky 15 toistolla pientä testidatajoukkoa vastaan.

6 Tulokset ja analyysi

Työssä implementoidut ja optimoidut mallit testattiin ensin 15 toistolla suurempaa testidatajoukkoa vastaan taulukon 6.1 mukaisesti.

Taulukko 6.1: Hyökkäysten jakauma koko datassa ja ensimmäisessä testidatajoukossa.

Kategoria	Alikategoria	Bot-IoT	1. testidata
Tietojen urkinta	Porttiskannaus	73168	73068
	OS-tietojen kalastelu	17914	17814
Palvelunestohyökkäys	TCP	1593180	150000*
	UDP	1981230	100000*
	HTTP	2474	2374
Tietojen varastaminen	Murtautuminen	6	6
	Näppäintallennus	73	73
Normaali		477	50
Yhteensä		3668522	343385

Taulukossa 6.1 merkintä * tarkoittaa satunnaista otosta, joka tehtiin kullekin mallille erikseen[1]. Tämä tehtiin laskennallisten resurssien ja ajankäytön säästämiseksi. Huomaa, että TCP, UDP ja HTTP sisältää myös hajautetut palvelunestohyökkäykset[2]. Testidatasta on jätetty ulkopuolelle myös aiemmin esitellyssä pienessä testidatajoukossa olleet havainnot, eli 200 havaintoa tietojen urkintaa ja 300 havaintoa pal-

[1]TCP- ja UDP-pohjaisten palvelunestohyökkäyksien osuus koko datajoukossa on suuri ja testidatan sisältämä otos on likimain 7 prosenttia siitä. Näin ollen yksittäisellä satunnaisotoksella saattaa olla harhainen vaikutus eli mallikohtaisilla satunnaisotoksilla kasvatettiin todennäköisyyttä, että palvelunestohyökkäyksiä käsitellään mahdollisimman monipuolisesti.

[2]Merkitään DDoS + DoS = yhteensä, jolloin TCP: 977380 + 615800 = 1593180, UDP: 948255 + 1032975 = 1981230 ja HTTP: 989 + 1485.

velunestohyökkäyksiä. Tietojen varastamiseen liittyvät havainnot pidettiin mukana molemmissa datajoukoissa niiden vähyyden vuoksi. Testidata muodostui yhteensä 343385 havainnosta, joista 50 havaintoa olivat mallien optimoinnissa ja kynnysarvon testaamisessa käytetyt normaalit havainnot. Taulukoissa 6.2 ja 6.3 on esitetty saadut tulokset.

Taulukko 6.2: Syväoppimismallien sekaannusmatriisit ensimmäisessä testissä.

ANN-AE		
Luokka	Normaali	Anomalia
Normaali	50	0
Anomalia	486	342849

CNN-AE		
Luokka	Normaali	Anomalia
Normaali	50	0
Anomalia	9318	334017

LSTM-AE		
Luokka	Normaali	Anomalia
Normaali	50	0
Anomalia	86	343249

GRU-AE		
Luokka	Normaali	Anomalia
Normaali	50	0
Anomalia	233	343102

Taulukko 6.3: Syväoppimismallien suorituskyvyt ensimmäisessä testissä.

Mittari	ANN-AE	CNN-AE	LSTM-AE	GRU-AE
Virheettömyys (*eng. accuracy*)	0.99858	0.97286	0.99975	0.99932
Osumatarkkuus (*eng. precision*)	1	1	1	1
Herkkyys (*eng. recall*)	0.99858	0.97286	0.99975	0.99932
F1-mitta (*eng. F1-score*)	0.99929	0.98624	0.99987	0.99966

Sekaannusmatriisien 6.2 perusteella syväoppimismallit erottelivat normaalin liikenteen onnistuneesti. Tähän toki vaikuttaa se, että testidatan sisältämää normaalia liikennettä käytettiin myös mallien optimointiin, eli ne eivät olleet malleille ennalta tuntemattomia havaintoja. Huomionarvoista on kuitenkin tunnistettujen hyökkäysten ja väärien hälytysten määrä. Kaikkien mallien suorituskyky on poikkeuksellisen korkea, mikä heijastuu taulukon 6.3 suorituskykymittareista. Tämä oli hieman odottamatonta, koska hyökkäyksiä on nyt satoja tuhansia ja aiemmin optimoinnin apu-

na käytetty pieni testidatajoukko sisälsi vain satoja hyökkäyksiä[3]. Mallien keskinäistä suorituskykyä sekaannusmatriiseja tarkastelemalla huomataan toisaalta selkeitä eroja. Konvoluutionaalinen autoenkooderi jättää yli 9000 hyökkäystä tunnistamatta, kun esimerkiksi seuraavaksi suorituskykyisimmältä autoenkooderilta ANN-AE:lta jää tunnistamatta liki 500. Sekvenssioppijat ovat testin suorituskykyisimpiä, missä LSTM-AE jättää hyökkäyksiä vähiten tunnistamatta.

Tarkempi tarkastelu tunnistamatta jääneisiin hyökkäyksiin paljastaa, että tietojen urkinta oli hankalinta tunnistaa kaikilla malleilla. Erityisesti konvoluutionaaliselta autoenkooderilta jäi 8979 porttiskannaukseksi luokiteltua havaintoa tunnistamatta. Myös muilla malleilla valtaosa tunnistamattomista hyökkäyksistä oli porttiskannauksia. Palvelunestohyökkäyksiä jäi tunnistamatta konvoluutionaalisella autoenkooderilla 85 ja GRU-sekvenssioppijalla 43, joissa molemmissa tapauksissa oli hajautettujakin joukossa. Eniten hyökkäyksiä tunnistaneella LSTM-AE:lla jäi 57 porttiskannausta, 25 OS-tietojen kalasteluyritystä ja 3 näppäintallennusta tunnistamatta. Huomionarvoista on myöskin se, että eteenpäin syöttävällä autoenkooderilla jäi tunnistamatta pelkästään tietojen urkintaan liittyviä porttiskannauksia 483 ja näppäintallennuksia 2.

Ensimmäisestä testistä suorituskykyisin malli eli LSTM-AE valittiin toiseen testiin, jossa otettiin 50 normaalin havainnon lisäksi kaikki hyökkäykset käyttöön pienessä testidatajoukossa olleita 200 tietojen urkinnaksi ja 300 palvelunestohyökkäyksi luokiteltuja havaintoja lukuunottamatta. Testidatajoukon kooksi saatiin kaiken kaikkiaan 3668522-500-427 = 3667595 havaintoa. Testi tehtiin 10 toistolla ajan säästämiseksi. Taulukko 6.4 havainnollistaa mallin sekaannusmatriisia ja suorituskykyä tässä tapauksessa.

Taulukko 6.4: LSTM-AE -mallin suorituskyky kaikkia hyökkäyksiä vastaan.

LSTM-AE		
Luokka	Normaali	Anomalia
Normaali	50	0
Anomalia	101	3667444

Mittari	LSTM-AE
Virheettömyys (*eng. accuracy*)	0.99997
Osumatarkkuus (*eng. precision*)	1
Herkkyys (*eng. recall*)	0.99997
F1-mitta (*eng. F1-score*)	0.99999

[3]Pienen testidatajoukon käytöstä johtuva tietovuoto on saattanut olla odotettua suurempaa eli koko datajoukko on saattanut olla oletettua homogeenisempää.

Mallilta jää tunnistamatta vain 15 hyökkäystä enemmän ensimmäiseen testiin verrattuna, vaikka hyökkäyksiä oli datassa yli 10-kertaisesti. Toisaalta datajoukko pysyi muuten samana, mutta nyt palvelunestohyökkäyksistä otettiin kaikki havainnot mukaan. Ensimmäiseen testiin verrattuna mallilta tunnistamatta jääneet havainnot olivat muuten identtisiä, mutta nyt porttiskannauksia oli 14 lisää ja yksi TCP-protokollaan kohdistunut palvelunestohyökkäys jäi havaitsematta.

Testien tulokset olivat rohkaisevia, kun otetaan huomioon, että normaaliksi luokiteltuja havaintoja oli niukasti. Optimoidut syväoppimismallit olivat arkkitehtuurillisesti ja parametrien määrien puolesta yksinkertaisia syväoppimismenetelmien mittapuulla[4]. Mallit ovat myös laskennallisen keveyden puolesta soveltuvia langattomien sensoriverkkojen asettamiin rajotteisiin. Erilaisia konfiguraatioita testattiin myös kattavasti laajalta rintamalta erikoistuneempia menetelmiä kuten regularisointia sekä erilaisia alustusmekanismeja ja aktivointifunktioita hyödyntäen. Yllättävää oli, että kustannusfunktioista kosinietäisyys ja datan esikäsittelymenetelmistä z-arvolla standardointi toimivat parhaiten niin syväoppimismallien optimoinnin kuin suorituskyvyn puolesta. Samoin regularisoinnilla ja kohinan lisäämisellä oli tärkeä rooli mallien suorituskyvyn kannalta. Lisäksi mallien optimoinnissa kerralla prosessoitavien opetusesimerkkien ja kierrosten lukumäärällä oli merkittävä vaikutus mallien oppimisprosessiin.

Joissakin muissa hyökkäyksien tunnistamista Bot-IoT -datajoukon avulla käsittelevissä tutkimuksissa kehitettyjen mallien syötteenä käytettävien muuttujien määrä määritettiin joko itse tai soveltamalla Bot-IoT -datajoukon tekijöiden Koroniotis ym. [83] löytämää 10 muuttujan joukkoa [69]. Alkadi ym. [2] käyttivät muuttujien välisiä lineaarisia riippuvuuksia etsivää pääkomponenttianalyysia 10 muuttujan irrottamiseen alkuperäisestä yli 73 miljoonan havainnon datajoukosta. Khraisat ym. [79] hyödynsivät Koroniotis ym. [83] tapaan informaatioteoreettista entropiaan pohjautuvaa lähestymistapaa, jolla valittiin yhteensä 13 muuttujaa. Tämän työn tulokset kuitenkin antavat viitteitä siitä, että pienempikin määrä muuttujia riittää tunnistamaan hyökkäyksiä Bot-IoT -datajoukosta: eteenpäin syöttävässä ja konvoluutionaalisessa autoenkooderissa muuttujille johdettiin kahdeksan muuttujan esitysmuoto, kun taas sekvenssioppijat rekonstruoivat syötteitä dekooderissa kuudesta ulottuvuudesta. Huomionarvoista on myös se, että muissa Bot-IoT -datajoukkoa soveltavissa töissä ei ole käytetty ohjaamatonta lähestymistapaa autoenkoodereiden avulla

[4]Myös muissa Bot-IoT -datajoukkoa soveltaneissa töissä kuten [83, 45] käytetyt syväoppimismenetelmät olivat arkkitehtuureiltaan huomattavasti monimutkaisempia.

kuten tässä työssä on tehty. Lisäksi tässä työssä saavutetut tulokset ovat yleisesti ottaen parempia kuin muissa töissä. Tähän on toisaalta saattanut vaikuttaa se, että mallien kynnysarvon toimivuutta testatiin pienellä määrällä hyökkäysdataa.

Jatkokehitysideoita

Eräs syväoppimismenetelmien haasteista on niiden musta laatikkomaisuus eli varsinkin operaatiokriittisissä sovelluksissa herää kysymys, miten syväoppimismallien soveltaja voi luottaa siihen, että mallin antamat tulokset perustuvat oikeelliseen esitykseen käsiteltävästä ongelmasta. Tässä työssä ei tarkasteltu sen tarkemmin, miten esimerkiksi yksittäiselle havainnolle laskettu rekonstruktiovirhe jakaantui muuttujien kesken eli millä muuttujalla oli suurin vaikutus poikkeamien havaitsemiseen. Rekonstruktiovirheiden jakautuneisuuden määrittäminen ei toisaalta ollut suoraviivaista kosinietäisyyden takia: siinä lasketaan kahden vektorin välistä kulmaa eikä niinkään kahden yksittäisen muuttujan. Konvoluutionaalisessa mallissa logaritmisen hyperbolisen kosinin myötä rekonstruktiovirheiden laskeminen per muuttuja olisi voinut onnistuakin, mutta konvoluutionaalinen malli suoriutui tässä työssä heikoimmin.

Eräs tapa tutkia muuttujien tärkeyttä mallissa on mitata ennustuksissa tapahtuvan virheen suuruutta silloin, kun jonkin muuttujan arvot sekoitetaan satunnaisesti (*eng. permutation feature importance*). Jos ennustusvirhe kasvaa kyseisen muuttujan arvojen sekoittamisen jälkeen, malli mitä ilmeisemmin luotti kyseiseen muuttujaan aiemmin ennustuksia tehdessään [102]. Tämän tyyppistä menettelyä voisi soveltaa apuna niin normaalia kuin poikkeavaa liikennettä karakterisoivien muuttujien tutkimisessa. Se ei kuitenkaan ole täysin ongelmatonta esimerkiksi korrelaation tai kollineaarisuuden ollessa läsnä: yhden muuttujan arvojen sekoittamisella ei ole kovin paljon vaikutusta ennustuksiin, koska se ei yksin määritä ennustusten laatua. Lisäksi olisi mielenkiintoista selvittää, mitkä alkuperäisistä muuttujista vaikuttavat eniten autoenkoodereissa pienemmän ulottuvuuden omaavan esitysmuodon syntyyn. Syväoppimismenetelmien sisäisten toiminnallisuuksien tulkitseminen ja analysointi sekä niiden antamien tulosten selittäminen on hyvin nuori ja jatkuvasti kehittyvä tutkimusala, jota on syytä seurata [103].

Implementoitujen syväoppimismallien syötteiden muodostamiseksi alkuperäisten muuttujien joukkoa karsittiin. Näihin muuttujiin lukeutuivat muun muassa lähettäjän ja vastaanottajan IP-osoitteet sekä portit. Näiden muuttujien sisällyttäminen malleihin saattaa lisätä mallien erottelukykyä normaalista ja poikkeavasta lii-

kenteestä. Haasteeksi muodostuu kuitenkin se, että missä esitysmuodossa muuttujat tulisi sisällyttää. IP-osoitteiden tapauksessa suoraviivainen tapa on kategorisoida antamalla kullekin uniikille IP-osoitteelle kokonaisluku niiden esiintymisjärjestyksessä. Tällöin kuitenkin suuremman kokonaisluvun omaavalla IP-osoitteella on syväoppimismallin näkökulmasta suurempi arvo, mikä ei ole realistista. Eräs vaihtoehto on myös irrottaa IP-osoitteen numerot erikseen ja syöttää ne kokonaislukuina syväoppimismalliin muiden muuttujien lisäksi, mutta tällöin IP-osoitteiden välillä on mahdollista tehdä aritmetiikkaa[5]. IP-osoitteet on mahdollista enkoodata myös bittijonoiksi (*eng. one-hot encoding*) niin, että kullekin uniikille IP-osoitteelle johdetaan oma bittiesitys [6] ja kukin bitti syötetään omana ulottuvuutena syväoppimismallille. Jos uniikkeja IP-osoitteita on paljon, syväoppimismallin syöte tulee hyvin moniulotteiseksi, mikä osaltaan kasvattaa mallin kompleksisuutta. IP-osoitteet voidaan muuttaa myös niille ominaiseksi 32-bittiseksi binääriesitykseksi tai käyttää tiivistefunktiota (*eng. hash function*). Vaikka tässä tapauksessa on mahdollista mallintaa hyvinkin monia uniikkeja osoitteita, niiden välinen erottelu voi olla syväoppimismallille hankalaa. Toisin sanoen toisiaan muistuttavat bittijonoesitykset voivat olla saman kaltaisia syväoppimismallille, kun taas todellisuudessa ne ovat kaksi täysin eri IP-osoitetta. Tiivistefunktioilla törmäykset (*eng. hash collision*) eli duplikaatit ovat myös mahdollisia.

Suotuisin tapa saattaa olla jonkin edellä mainitun esitysmuodon laatiminen, kuten kokonaislukuesitys tai *one-hot* -enkoodaus, ja saatujen esitysmuotojen projisointi tai muuntaminen mahdollisesti alempiulotteiseen vektoriavaruuteen jatkuva-arvoisiksi vektoreiksi (*eng. embedding*) jonkin yksinkertaisen säännön tai kehittyneemmän algoritmin avulla. Tämän tyyppistä datan esikäsittelyä tehdään paljon muun muassa luonnollisen kielen prosessoinnin (*natural language processing, NLP*) erilaisissa sovelluksissa, missä hyödynnetään syväoppimismenetelmiä tekstimuotoisen datan kanssa toimimiseen. Eräs yleinen haaste esikäsittelymenetelmästä riippumatta on se, että miten ennalta tuntemattomien kategorioiden esitysmuodot laaditaan dynaamisesti esimerkiksi syväoppimismenetelmän opettamisessa ja testausvaiheessa. Edellä esitetyt argumentit IP-osoitteilla havainnollistettuna pätevät myös porttien kohdalla. Lisäksi, kun lisätään konteksti mukaan, niin eri IP-osoitteiden välisiin suhteisiin vaikuttaa itse IP-osoitteistus eli millä säännöillä IP-osoitteet allokoidaan

[5]Esimerkiksi 1.1.1.1 + 2.2.2.2 = 3.3.3.3.

[6]Esimerkiksi kolmen IP-osoitteen tapauksessa niille voidaan johtaa uniikit bittiesitykset $[100, 010, 001]$.

päätelaitteille ja miten esimerkiksi samoihin aliverkkoihin kuuluvien päätelaitteiden IP-osoitteet muodostetaan. Lisäksi porteilla on erilaisia merkityksiä: osa porteista ovat hyvin tunnettuja ja käytettyjä (0-1024) osan ollessa harvinaisempia. Tietyt sovellukset käyttävät tiettyjä portteja ja osa verkkohyökkäyksistäkin kohdistuu tiettyihin portteihin. IP-osoitteiden ja porttien sisällyttäminen syväoppimismalleihin monimuotoisen kontekstin huomioivalla tavalla on mielenkiintoinen ongelma.

Olemassa olevien anomalioiden havaitsemismenetelmien eräs ongelmista on tarve datan esikäsittelylle kuten erilaisille standardisoinneille ja normalisoinneille, joiden seurauksena datan luonne saattaa muuttua ei-toivotulla tavalla. Tämä voi olla ratkaisevaa esimerkiksi silloin, kun esikäsittely tekee poikkeavista havainnoista vähemmän poikkeavia tai vastaavasti normaaleista havainnoista poikkeavampia. Lisäksi esikäsittelyiden yhteydessä informaation täydellistä säilymistä ei voida taata vaan informaatiota häviää väistämättä jonkin verran [156]. Verkkoliikenteeseen suunnatuissa anomalian havaitsemismenetelmissä voi olla edullista prosessoida raakaa pakettidataa tässäkin työssä analysoitujen metatietojen sijaan. Raaka pakettidata on mahdollista muuntaa edelleen tietokoneen natiivikieleen eli binäärimuotoon, jolloin verkkoliikennettä kuvaava data on käytännössä hienojakoisimmissa muodossaan ja saadessaan arvoja väliltä $[0, 1]$, esikäsittelylle ei ole lähtökohtaisesti tarvetta. TCP/IP-verkoissa käytettävien Ethernet-pakettien binäärinen esitys voi kasvaa yli 12 000 ulottuvuuteen[7], mutta nykyään saatavilla olevien laskennallisten resurssien ja syväoppimismenetelmien vuoksi hyvin moniulotteisten binääripakettien prosessoiminenkaan ei ole enää este. Lisäksi rajoittuneemmissa ympäristössä kuten langattomissa sensoriverkoissa käytettävillä standardeilla voi olla edelleen pienemmät kehyskoot, kuten esimerkiksi 6LoWPANin 127 tavua eli 1016 bittiä [128]. Binääridatan kanssa toimiminen on lupaava lähestymistapa, koska se mahdollistaa tarkemman verkkoliikenteen analysoinnin. Tällä voi olla merkittävä vaikutus esimerkiksi edistyksellisempien ja hienovaraisten sekä erityisesti ennalta tuntemattomien verkkohyökkäysten havaitsemisessa [93, 17].

Kone- ja syväoppimismenetelmissä on yleistä esittää reaalimaailman ilmiöitä edustava data n-ulotteisessa reaalisessa vektoriavaruudessa $\mathbb{R}^n$, jolle pätevät euklidisen geometrian aksioomat. Syväoppimismenetelmien menestys on saavutettu juurikin euklidisen datan käsittelyllä ja jota tässäkin työssä tehtiin. Hiljattain tieteel-

[7] Ethernet-paketti koostuu kuudesta kentästä, missä datakentän koko voi kasvaa aina 1500 tavuun asti. Muut kentät ovat vakiokokoisia, jolloin suurin mahdollinen koko on 1526 tavua eli yhteensä 12208 bittiä [86].

lisessä yhteisössä on kuitenkin havaittu, että monilla reaalimaailman ilmiöillä on epäeuklidinen rakenne, joka tulee ottaa huomioon myös ilmiötä mallinnettaessa [19, 49]. On osoitettu, että joissakin tapauksissa euklidinen avaruus ei tarjoa mallinnuskyvyltään parhainta ja merkityksellisintä esitystapaa. Epäeuklidisen datan eräitä tunnetuimpia esitysmuotoja ovat kaarevat pinnat sekä graafit ja monistot, joita esiintyy monissa sovelluksissa kuten sosiaalisissa, biologisissa, kemiallisissa, taloudellisissa, maantieteellisissä, fysikaalisissa, keinotekoisissa ja teknisissä verkottuneissa systeemeissä, kuten langattomissa sensoriverkoissa.

Graafipohjaisen syväoppimisen ideana on löytää keinoja sisällyttää verkottuneista järjestelmistä johdettujen graafien sisältämää informaatio osaksi mallinnusta [59]. Graafeilla on mahdollista ottaa huomioon tarkasteltavan systeemin rakenne ja siinä tapahtuvia muutoksia, systeemissä toimivien osallistujien ominaisuuksia ja niiden keskinäisiä riippuvuuksia. Monissa tilanteissa monimuotoista tietoa sisältävät graafit ovat luontevampi tapa mallintaa ilmiöitä [145]. Erääksi graafipohjaisen syväoppimisen tulevaisuuden sovellukseksi on todettu poikkeavan käyttäytymisen havainnointi erilaisissa ajan mittaan muuttuvissa verkottuneissa rakenteissa, jolla voi olla suuria hyötyjä esimerkiksi langattomissa sensoriverkoissa [19, 125]. Anomalioiden havaitseminen tämän tyyppisistä graafeista on ollut vahvemmin esillä vasta puoli vuosikymmenen [116]. Dynaamisesti muuttuvien graafien mallintaminen graafipohjaisilla syväoppimismenetelmillä on myöskin vielä laajalti ratkaisematon tutkimuskysymys, joka on tärkeä ongelma ja jota tutkitaan tiedeyhteisössä [155, 153, 19, 143, 59]. Tiedeyhteisössä ei myöskään ole vielä sovellettu graafipohjaisia syväoppimismenetelmiä anomalioiden havaitsemiseen langattomissa sensoriverkoissa.

Ihmisen tapa ratkaista ongelmia pohjautuu usein kykyyn osittaa käsillä oleva ongelma osiin etsien yhtäläisyyksiä omiin olemassa oleviin tietoihin ja osaamiseen pohjautuen. Ihmisälykkyyden eräs tunnetuimpia ominaisuuksia onkin kyky hahmottaa ympäröivä maailma pieninä osakokonaisuuksista koostuvina rakenteina ja tunnistaa niiden välisiä riippuvuussuhteita. Tätä kutsutaan kombinatoriseksi yleistettävyydeksi (*eng. combinatorial generalization*). Graafipohjaisen syväoppimisen katsotaan olevan eräs tärkeä katalyytti kohti ihmismäisempää tekoälyä, koska sillä on mahdollista lisätä syväoppimismallien kombinatorisen yleistettävyyden kykyä. Tällöin syväoppimismallit voivat oppia tunnistamaan ja hyödyntämään datassa esiintyvien toimijoiden ja objektien konstruktioita ja osittuvuutta sekä niiden välisiä riippuvuussuhteita ilman, että niitä tulee erikseen ennalta määrittää. Tekemällä lasken-

taa rakenteellisista esitysmuodoista kuten graafeista ja niiden sisältämien toimijoiden välisistä suhteista, syväoppimismalliin syötetään induktiivisesti tietoja erilaisista riippuvuuksista (*eng. relational inductive bias*). Toisin sanoen mallien itsenäinen päättelykyky saattaa olla monipuolisempaa ja näin syväoppimismallit voivat suoriutua paremmin tilanteista, jotka ovat sille entuudestaan täysin tuntemattomia [7].

Tekoälytutkimuksesta

Syväoppiminen on ollut tekoälytutkimuksen ja -sovellusten kantava voima viime vuosikymmenen aikana. Syväoppimismenetelmillä on saavutettu huipputuloksia niin kuvantunnistuksessa, luonnollisen kielen prosessoinnissa kuin lauta- ja videopelejä pelaavien ohjelmistojen kehityksessä. Yksittäistä ongelmaa vastaan optimoitu suorituskyky on hyödyllistä, jos alkuperäisenä tavoitteena ei ole saavuttaa menestystä kyseisen tehtävän ulkopuolelle jäävissä tai muuten hieman erilaisissa tehtävissä. Monissa tapauksissa haasteena onkin se, että tiettyihin tehtäviin tai ongelmiin kehitetyt mallit eivät yleisty niiden ulkopuolelle tai edes niiden sisällä. Esimerkiksi OpenAI:n DotA2-videopeliä pelaava tekoäly Five, joka kykeni päihittämään huippupelaajia ympäri maailman, hävisi suuremman yleisön edessä ei-huippupelaajien yksinkertaisimmille strategioille. Lisäksi Five kykeni pelaamaan videopelistä vain rajoitettua versiota. DeepMindin AlphaGo:lle ei myöskään ole löydetty vielä muuta sovellusta lautapelien ulkopuolelta. Mikäli tekoälytutkimusta halutaan ohjata merkittävästi eteenpäin kohti vahvaa tai yleistä tekoälyä (*eng. general artificial intelligence*), suorituskyvyn arvioiminen yksittäisessä tehtävässä ei sellaisenaan riitä [29].

Tähän astisen tekoälytutkimuksen katsotaan keskittyneen vain lokaalin yleistettävyyden (*eng. local generalization*) saavuttamiseen, missä pieni määrä tehtäviä tunnetaan ja niissä esiintyviin tuntemattomiin havaintoihin[8] pyritään tiedostetusti varautumaan. Laaja yleistettävyys (*eng. broad generalization*) sen sijaan kuvaa järjestelmän kykyä sopeutua moniin eri tehtäviin erilaisissa ympäristöissä ilman ihmisen väliintuloa. Tähän sisältyy kyky sopeutua sellaisiin tunnettuja piirteitä omaaviin tilanteisiin, joita järjestelmä eivätkä myöskään järjestelmän tekijät ole tiedostaneet ennalta[9]. Äärimmäisellä yleistettävyydellä (*eng. extreme generalization*) viitataan kykyyn, jossa järjestelmä osaa sopeutua sellaisiin täysin uusiin tilanteisiin, joilla on moniin eri tehtäviin sovellettavia yhtäläisyyksiä ennalta tunnettuihin ongelmiin nähden, mutta vain hyvin abstraktilla ja yleisellä tasolla. Tällöin järjestelmällä

[8] *eng. adaptation to known unknowns within a single task or well-defined set of tasks.*

[9] *eng. adaptation to unknown unknowns across a broad category of related tasks.*

on kyky ja potentiaali sopeutua ennalta tuntemattomaan määrään ja kirjoon ennalta tiedostamattomia ongelmia ja tilanteita[10]. Nykypäivän kehittyneimpien tekoälyjärjestelmien ei katsota omaavan laajaa yleistettävyyden kykyä ja tällä hetkellä vain ihmisellä tunnetaan olevan äärimmäisen yleistettävyyden kyky [29].

Jotta lokaalin yleistettävyyden omaavista järjestelmistä päästään eteenpäin kohti ihmismäistä tekoälyä, tulee tekoälymenetelmien suorituskyvyn (*eng. skill*) sijaan arvioida niiden kykyä omaksua uusia kykyjä ja soveltaa niitä erilaisissa ongelmissa, joista osa on täysin ennalta tuntemattomia. Keskeinen ero nykypäivän tekoälytutkimukseen tässä on se, että äärimmäisen suorituskyvyn sijasta tulee pyrkiä saavuttamaan jokin kynnysarvo, mikä merkitsee tehtävän ratkaisemista. Se, mitä mitataan, on tehtävän ratkaisemiseen vaadittujen kykyjen käytön ja mahdollisesti uusien kykyjen oppimisen tehokkuus (*eng. skill-acquisition efficiency*). Tehtäviä tulee myös olla monia mahdollisimman monipuolisista tilanteista. Huippuluokan suorituskyvyn vertaileminen yksittäisessä tehtävässä ei vastaa älykkyyden vertailua. Lisäksi älykkäiden menetelmien vertailussa menetelmillä tulee olla samat lähtökohdat tehtävien ratkaisemiseen. Tekoälymenetelmien ihmismäisen älykkyyden tunnistamisen vuoksi tuloksia tulee verrata ihmisasiantuntijan saavuttamaan ongelmanratkaisukykyyn ja oppimistehokkuuteen vastaavissa tehtävissä. Tämä ei toisaalta tarkoita sitä, etteikö syväoppimismenetelmillä voi saavuttaa laajempaa yleistettävyyden kykyä, vaan syväoppimismenetelmiin perustuvien järjestelmien kehitys tulee keskittyä äärimmäisen suorituskyvyn sijasta niiden yleistettävyyteen [29]. Tämän työn osalta voidaan todeta, että kehitetty syväoppimismenetelmä omaa lokaalin yleistettävyyden.

[10] *eng. adaptation to unknown unknowns across an unknown range of tasks and domains.*

7 Yhteenveto

Tässä työssä tutustuttiin langattomiin sensoriverkkoihin ja niiden ominaisuuksiin sekä haavoittuvuuksiin ja uhkiin tietoturvallisuuden näkökulmasta. Lisäksi käsiteltiin anomalioiden havaitsemista. Tämän jälkeen annettiin kattava teoreettinen katsaus syväoppimiseen ja moderneihin syväoppimismenetelmiin kuten eteenpäin syöttäviin, konvoluutionaalisiin ja toistuviin neuroverkkoihin. Lisäksi esitettyjen syväoppimismenetelmien soveltamista tarkasteltiin anomalioiden havaitsemisen ja puoliohjatun oppimisen näkökulmasta autoenkooderiarkkitehtuurin avulla. Syväoppimismenetelmien optimointiin liittyviin haasteisiin kuten ali- ja ylisovittumiseen, korkeiden ulottuvuuksien kiroukseen ja optimointimenetelmiin sekä niiden mahdollisiin ratkaisukeinoihin pureuduttiin myöskin syvemmin.

Työn empiirisessä osuudessa implementoitiin annetun teorian pohjalta neljä autoenkooderiarkkitehtuuriin pohjautuvaa syväoppimismenetelmää, joiden avulla tutkittiin verkkohyökkäysten tunnistamista hiljattain julkaistusta esineiden internetin sovellusympäristöä jäljittelevästä Bot-IoT -datajoukosta. Implementoidut syväoppimismenetelmät olivat suorituskykyisiä ja laskennallisesti kevyitä sopeutuen langattomien sensoriverkkojen asettamiin rajotteisiin. Suorituskykyisimmäksi malliksi osoittautui toistuvaan neuroverkkoon pohjautuva LSTM-autoenkooderi, joka tunnisti yli 3,6 miljoonaa hyökkäystä jättäen vain 101 hyökkäystä tunnistamatta. Tiedeyhteisössä ei ole aiemmin tehty vastaavaa tutkimusta Bot-IoT -datajoukolla eikä ole myöskään saavutettu vastaavia tuloksia. Tulosten läpikäynnin lisäksi esitettiin myös erilaisia jatkokehitysideoita ja pohdittiin tekoälytutkimuksen asemaa tällä hetkellä sekä sen suuntaviivoja tulevaisuudessa.

Tiedeyhteisössä sovelletaan yhä laajemmin ja kasvavissa määrin älykkäitä syväoppimismenetelmiin pohjautuvia ratkaisuja eri käyttötapauksiin. Näihin lukeutuu myös langattomat verkot ja mobiili- sekä IoT-sovellukset haastavine ja vaativine ympäristöineen. Erityisesti langattomiin sensoriverkkoihin kohdistuu laaja kirjo erilaisia uhkia, joilta suojautuminen on tärkeä tutkimusaihe ja tulee olemaan tulevaisuudessa niin IoT:n ja 5G:n edelleen kiihtyvän kasvun kuin alati kehittyvän kyberrikollisuuden ja lisääntyvien tietoturvariskien myötä.

Lähteet

[1] AKOGLU, L., TONG, H., JA KOUTRA, D. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery 29*, 3 (2015), 626–688.

[2] ALKADI, O., MOUSTAFA, N., TURNBULL, B., JA CHOO, K.-K. R. Mixture localization-based outliers models for securing data migration in cloud centers. *IEEE Access 7* (2019), 114607–114618.

[3] AMINI, A. Homepage of alexander amini. URL: `https://www.mit.edu/~amini/`, viitattu 25.11.2019.

[4] ARORA, R., BASU, A., MIANJY, P., JA MUKHERJEE, A. Understanding deep neural networks with rectified linear units. *ArXiv abs/1611.01491v6* (2018).

[5] BAIG, Z. A., SANGUANPONG, S., FIRDOUS, S. N., NGUYEN, T. G., JA SO-IN, C. Averaged dependence estimators for dos attack detection in iot networks. *Future Generation Computer Systems 102* (2020), 198–209.

[6] BALL, J. E., ANDERSON, D. T., JA SR., C. S. C. Comprehensive survey of deep learning in remote sensing: theories, tools, and challenges for the community. *Journal of Applied Remote Sensing 11*, 4 (2017), 1–54.

[7] BATTAGLIA, P. W., HAMRICK, J. B., BAPST, V., SANCHEZ-GONZALEZ, A., ZAMBALDI, V. F., MALINOWSKI, M., TACCHETTI, A., RAPOSO, D., SANTORO, A., FAULKNER, R., ÇAGLAR GÜLÇEHRE, SONG, H. F., BALLARD, A. J., GILMER, J., DAHL, G. E., VASWANI, A., ALLEN, K. R., NASH, C., LANGSTON, V., DYER, C., HEESS, N. M. O., WIERSTRA, D., KOHLI, P., BOTVINICK, M. M., VINYALS, O., LI, Y., JA PASCANU, R. Relational inductive biases, deep learning, and graph networks. *ArXiv abs/1806.01261v3* (2018).

[8] BENGIO, Y. Learning deep architectures for ai. *Foundations and Trends in Machine Learning 2*, 1 (2009), 1–127.

[9] BENGIO, Y., COURVILLE, A., JA VINCENT, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence 35*, 8 (2013), 1798–1828.

[10] BERA, A. 80 iot statistics (infographic). URL: https://safeatlast.co/blog/iot-statistics/#gref, viitattu 30.11.2019.

[11] BHUYAN, M. H., BHATTACHARYYA, D. K., JA KALITA, J. K. Network anomaly detection: methods, systems and tools. *Ieee communications surveys & tutorials 16*, 1 (2013), 303–336.

[12] BISHOP, C. M. *Pattern Recognition and Machine Learning*. Springer Science+ Business Media, 2006.

[13] BLAINE, G. Sonicwall: Encrypted attacks, iot malware surge as global malware volume dips. URL: https://blog.sonicwall.com/en-us/2019/10/sonicwall-encrypted-attacks-iot-malware-surge-as-global-malware-volume-dips/, viitattu 30.11.2019.

[14] BLOCK, H.-D. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics 34*, 1 (1962), 123.

[15] BODSTRÖM, T., JA HÄMÄLÄINEN, T. State of the art literature review on network anomaly detection. Kirjassa *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*. Springer, 2018, ss. 89–101.

[16] BODSTRÖM, T., JA HÄMÄLÄINEN, T. State of the art literature review on network anomaly detection with deep learning. Kirjassa *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*. Springer, 2018, ss. 64–76.

[17] BODSTRÖM, T., JA HÄMÄLÄINEN, T. A novel deep learning stack for apt detection. *Applied Sciences 9*, 6 (2019), 1055.

[18] BRONSTEIN, M. M. Geometric deep learning on graphs and manifolds going beyond euclidean data - part i of ii. 2018 SIAM Annual Meeting. URL: https://www.pathlms.com/siam/courses/8264/sections/11773, viitattu 5.11.2019.

[19] BRONSTEIN, M. M., BRUNA, J., LECUN, Y., SZLAM, A., JA VANDERGHEYNST, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine 34*, 4 (2017), 18–42.

[20] BROWNLEE, J. What is machine learning? URL: `https://machinelearningmastery.com/what-is-machine-learning/`, viitattu 19.10.2019.

[21] BROWNLEE, J. *Better Deep Learning*, v1.3 ed. Machine Learning Mastery, 2019.

[22] BROWNLEE, J. *Deep Learning for Time-Series Forecasting*, v1.5 ed. Machine Learning Mastery, 2019.

[23] BRUCKNER, A. M., BRUCKNER, J. B., JA THOMSON, B. S. *Real Analysis, Second Edition*. 2008. URL: `http://classicalrealanalysis.info/com/Real-Analysis.php`.

[24] BUTUN, I., MORGERA, S. D., JA SANKAR, R. A survey of intrusion detection systems in wireless sensor networks. *IEEE Communications Surveys & Tutorials 16*, 1 (2014), 266–282.

[25] CHALAPATHY, R., JA CHAWLA, S. Deep learning for anomaly detection: A survey. *ArXiv abs/1901.03407v2* (2019).

[26] CHEN, Z., YEO, C. K., LEE, B. S., JA LAU, C. T. Autoencoder-based network anomaly detection. Julkaisusarjassa *2018 Wireless Telecommunications Symposium (WTS)* (2018), IEEE, 1–5.

[27] CHO, K., VAN MERRIENBOER, B., ÇAGLAR GÜLÇEHRE, BAHDANAU, D., BOUGARES, F., SCHWENK, H., JA BENGIO, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. Julkaisusarjassa *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), Association for Computational Linguistics, 1724–1734.

[28] CHOLLET, F. *Deep Learning with Python*, 1st ed. Manning Publications Co., 2017.

[29] CHOLLET, F. On the measure of intelligence. *ArXiv abs/1911.01547v2* (2019).

[30] CHONG, E., HAN, C., JA PARK, F. C. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications 83*, 1 (2017), 187–205.

[31] CISCO. Cisco visual networking index: Forecast and trends, 2017-2022 white paper. URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html, viitattu 30.11.2019.

[32] CLEVERT, D.-A., UNTERTHINER, T., JA HOCHREITER, S. Fast and accurate deep network learning by exponential linear units (elus). *ArXiv abs/1511.07289v5* (2016).

[33] CLOUD SECURITY ALLIANCE. Security guidance for early adopters of the internet of things (iot). URL: https://downloads.cloudsecurityalliance.org/whitepapers/Security_Guidance_for_Early_Adopters_of_the_Internet_of_Things.pdf, viitattu 3.11.2019.

[34] COHEN, N., SHARIR, O., JA SHASHUA, A. On the expressive power of deep learning: A tensor analysis. *ArXiv abs/1509.05009v3* (2016).

[35] COLUMBUS, L. 2018 roundup of internet of things forecasts and market estimates. URL: https://www.forbes.com/sites/louiscolumbus/2018/12/13/2018-roundup-of-internet-of-things-forecasts-and-market-estimates/, viitattu 30.11.2019.

[36] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems 2*, 4 (1989), 303–314.

[37] DARGIE, W., JA POELLABAUER, C. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Wiley Publishing, 2010.

[38] DASGUPTA, S., JA GUPTA, A. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms 22*, 1 (2003), 60–65.

[39] DEEPLEARNING.AI. Deep learning specialization - neural networks and deep learning. URL: https://www.coursera.org/learn/neural-networks-deep-learning, viitattu 29.11.2019.

[40] DERTAT, A. Applied deep learning - part 1: Artificial neural networks. URL: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6, viitattu 29.11.2019.

[41] DIVEKAR, A., PAREKH, M., SAVLA, V., MISHRA, R., JA SHIROLE, M. Benchmarking datasets for anomaly-based network intrusion detection: Kdd cup 99 alternatives. Julkaisusarjassa *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)* (2018), IEEE, 1–8.

[42] DUMOULIN, V., JA VISIN, F. A guide to convolution arithmetic for deep learning. *ArXiv abs/1603.07285v2* (2018).

[43] F-SECURE. Attack landscape h1 2019: Iot, smb traffic abound. URL: https://blog.f-secure.com/attack-landscape-h1-2019-iot-smb-traffic-abound/, viitattu 30.11.2019.

[44] F-SECURE. State of cyber security 2017. URL: https://fsecurepressglobal.files.wordpress.com/2017/02/cyber-security-report-2017.pdf, viitattu 30.11.2019.

[45] FERRAG, M. A., JA MAGLARAS, L. Deepcoin: A novel deep learning and blockchain-based energy exchange framework for smart grids. *IEEE Transactions on Engineering Management* (2019).

[46] FISCHER, T., JA KRAUSS, C. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operations Research 270*, 1 (2018), 654–669.

[47] FRANÇOIS-LAVET, V., HENDERSON, P., ISLAM, R., BELLEMARE, M. G., JA PINEAU, J. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning 11*, 3-4 (2018).

[48] FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics 36*, 4 (1980), 193–202.

[49] GANEA, O.-E., BÉCIGNEUL, G., JA HOFMANN, T. Hyperbolic neural networks. Julkaisusarjassa *Proceedings of the 32Nd International Conference on Neural Information Processing Systems* (2018), Curran Associates Inc., 5350–5360.

[50] GELLER, M., JA NAIR, P. 5g security innovation with cisco. URL: https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/service-provider-security-solutions/5g-security-innovation-with-cisco-wp.pdf, viitattu 30.11.2019.

[51] GÉRON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. O'Reilly Media, 2019.

[52] GERS, F. A., SCHRAUDOLPH, N. N., JA SCHMIDHUBER, J. Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research 3*, 1 (2003), 115–143.

[53] GLOROT, X., JA BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. Julkaisusarjassa *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), 249–256.

[54] GOODFELLOW, I., BENGIO, Y., JA COURVILLE, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[55] GRAVES, A. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012. Studies in Computational Intelligence 385.

[56] GRAVES, A. Generating sequences with recurrent neural networks. *ArXiv abs/1308.0850v5* (2014).

[57] GREFF, K., SRIVASTAVA, R. K., KOUTNÍK, J., STEUNEBRINK, B. R., JA SCHMIDHUBER, J. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems 28*, 10 (2017), 2222–2232.

[58] GRUSLYS, A., MUNOS, R., DANIHELKA, I., LANCTOT, M., JA GRAVES, A. Memory-efficient backpropagation through time. Julkaisusarjassa *Proceedings of the 30th International Conference on Neural Information Processing Systems* (2016), Curran Associates Inc., 4132–4140.

[59] HAMILTON, W. L., YING, R., JA LESKOVEC, J. Representation learning on graphs: Methods and applications. *ArXiv abs/1709.05584* (2018).

[60] HAMMER, B. On the approximation capability of recurrent neural networks. *Neurocomputing 31*, 1 (2000), 107–123.

[61] HARDESTY, L. Explained: Linear and nonlinear systems. URL: http://news.mit.edu/2010/explained-linear-0226, viitattu 3.11.2019.

[62] HE, K., ZHANG, X., REN, S., JA SUN, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Julkaisusarjassa *Proceedings of the IEEE international conference on computer vision* (2015), 1026–1034.

[63] HE, K., ZHANG, X., REN, S., JA SUN, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778.

[64] HELENIUS, T. Takaisinkytketyt neuroverkot. Informaatiotieteiden yksikkö, Tampereen yliopisto. URL: https://trepo.tuni.fi/handle/10024/98682, viitattu 22.10.2019.

[65] HERMANS, M., JA SCHRAUWEN, B. Training and analyzing deep recurrent neural networks. Julkaisusarjassa *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1* (2013), 190–198.

[66] HORNIK, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks 4*, 1 (1991), 251–257.

[67] HU, J., SHEN, L., JA SUN, G. Squeeze-and-excitation networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018).

[68] HUTTUNEN, H. Signaalinkäsittelyn perusteet. Signaalinkäsittelyn laitos, Tampereen teknillinen yliopisto. URL: http://www.cs.tut.fi/kurssit/SGN-11000/SGN-11000.pdf, viitattu 5.11.2019.

[69] IBITOYE, O., SHAFIQ, O., JA MATRAWY, A. Analyzing adversarial attacks against deep learning for intrusion detection in iot networks. *ArXiv abs/1905.05137* (2019).

[70] ICHI FUNAHASHI, K. On the approximate realization of continuous mappings by neural networks. *Neural Networks 2*, 1 (1989), 183–192.

[71] JOHNSON, J. Lecture 10: Recurrent neural networks. URL: http://cs231n.stanford.edu/2017/, viitattu 18.11.2019.

[72] JOHNSON, J. Lecture 11: Detection and segmentation. URL: http://cs231n.stanford.edu/2017/, viitattu 18.11.2019.

[73] JORDAN, J. Neural networks: training with backpropagation. URL: https://www.jeremyjordan.me/neural-networks-training/, viitattu 25.11.2019.

[74] KARHU, A. Kompaktien avaruuksien ominaisuuksia. Pro-gradu tutkielma. Luonnontieteiden ja metsätieteiden tiedekunta/Fysiikan ja matematiikan laitos, Itä-Suomen yliopisto, 2019. URL: http://epublications.uef.fi/pub/urn_nbn_fi_uef-20190977/, viitattu 2.11.2019.

[75] KARPATHY, A. Convolutional neural networks: Architectures, convolution/pooling layers. URL: http://cs231n.github.io/convolutional-networks/, viitattu 2.11.2019.

[76] KARPATHY, A. Neural networks part 1: Setting up the architecture. URL: http://cs231n.github.io/neural-networks-1/, viitattu 20.10.2019.

[77] KARPATHY, A. Neural networks part 2: Setting up the data and the loss. URL: http://cs231n.github.io/neural-networks-2/, viitattu 20.10.2019.

[78] KHRAISAT, A., GONDAL, I., VAMPLEW, P., JA KAMRUZZAMAN, J. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity 2*, 1 (2019), 20.

[79] KHRAISAT, A., GONDAL, I., VAMPLEW, P., KAMRUZZAMAN, J., JA ALAZAB, A. A novel ensemble of hybrid intrusion detection system for detecting internet of things attacks. *Electronics 8*, 11 (2019), 1210.

[80] KINGMA, D. P., JA BA, J. Adam: A method for stochastic optimization. *ArXiv abs/1412.6980v9* (2017).

[81] KINGMA, D. P., REZENDEY, D. J., MOHAMEDY, S., JA WELLING, M. Semi-supervised learning with deep generative models. Julkaisusarjassa *Advances in Neural Information Processing Systems 27* (2014).

[82] KOLIAS, C., KAMBOURAKIS, G., STAVROU, A., JA GRITZALIS, S. Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset. *IEEE Communications Surveys & Tutorials 18*, 1 (2015), 184–208.

[83] KORONIOTIS, N., MOUSTAFA, N., SITNIKOVA, E., JA TURNBULL, B. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems 100* (2019), 779–796.

[84] KRAUS, M., FEUERRIEGEL, S., JA OZTEKIN, A. Deep learning in business analytics and operations research: Models, applications and managerial implications. *European Journal of Operations Research* (2019). Article in Press.

[85] KUMAR, D. P., AMGOTH, T., JA ANNAVARAPU, C. S. R. Machine learning algorithms for wireless sensor networks: A survey. *Information Fusion 49*, 1 (2019), 1–25.

[86] KUROSE, J. F., JA ROSS, K. W. *Computer Networking: A Top-Down Approach*, 7th ed. Pearson Education, 2017.

[87] KWON, D., KIM, H., KIM, J., SUH, S. C., KIM, I., JA KIM, K. J. A survey of deep learning-based network anomaly detection. *Cluster Computing* (2017), 1–13.

[88] LABATUT, V., JA OZGOVDE, A. Topological measures for the analysis of wireless sensor networks. *Procedia Computer Science 10*, 1 (2012), 397–404.

[89] LE, Q. V. A tutorial on deep learning part 1: Nonlinear classifiers and the backpropagation algorithm. URL: `http://ai.stanford.edu/~quocle/tutorial1.pdf`, viitattu 20.10.2019.

[90] LE, Q. V. A tutorial on deep learning part 2: Autoencoders, convolutional neural networks and recurrent neural networks. URL: `http://robotics.stanford.edu/~quocle/tutorial2.pdf`, viitattu 1.11.2019.

[91] LECUN, Y., BENGIO, Y., JA HINTON, G. Deep learning. *Neurocomputing 521*, 7553 (2015), 436–444.

[92] LECUN, Y., HAFFNER, P., BOTTOU, L., JA BENGIO, Y. Object recognition with gradient-based learning. Julkaisusarjassa *Shape, Contour and Grouping in Computer Vision* (1999), Springer-Verlag, 319–.

[93] LEPPÄNEN, R. F., JA HÄMÄLÄINEN, T. Network anomaly detection in wireless sensor networks: A review. Julkaisusarjassa *Internet of Things, Smart*

Spaces, and Next Generation Networks and Systems (2019), Springer International Publishing, 196–207.

[94] LITJENS, G., KOOI, T., BEJNORDI, B. E., SETIO, A. A. A., CIOMPI, F., GHAFOORIAN, M., VAN DER LAAK, J. A., VAN GINNEKEN, B., JA SÁNCHEZ, C. I. A survey on deep learning in medical image analysis. *Medical Image Analysis 42*, 1 (2017), 60–88.

[95] LIU, W., WANG, Z., LIU, X., ZENG, N., LIU, Y., JA ALSAADI, F. E. A survey of deep neural network architectures and their applications. *Neurocomputing 234*, 1 (2017), 11–26.

[96] MAATEN, L. V. D., POSTMA, E., JA DEN HERIK, J. V. Dimensionality reduction: a comparative.

[97] MAKRIDAKIS, S. Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting 9*, 4 (1993), 527–529.

[98] MCCAFFREY, J. Step up to recurrent neural networks. URL: `https://visualstudiomagazine.com/articles/2015/10/01/recurrent-neural-networks.aspx`, viitattu 29.11.2019.

[99] MIRZA, A. H., JA COSAN, S. Computer network intrusion detection using sequential lstm neural networks autoencoders. Julkaisusarjassa *2018 26th Signal Processing and Communications Applications Conference (SIU)* (2018), IEEE, 1–4.

[100] MITCHELL, T. *Machine Learning*, 1st ed. McGraw-Hill Education, 1997.

[101] MODENIA, A., JA JHA, N. K. A comprehensive study of security of internet-of-things. *IEEE Transactions on Emerging Topics in Computing 5*, 4 (2017), 586–602.

[102] MOLNAR, C. Interpretable machine learning - a guide for making black box models explainable. URL: `https://christophm.github.io/interpretable-ml-book/`, viitattu 28.11.2019.

[103] MONTAVON, G., SAMEK, W., JA MÜLLER, K.-R. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing 73* (2018), 1–15.

[104] NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. Julkaisusarjassa *2017 IEEE international systems engineering symposium (ISSE)* (2017), IEEE, 1–7.

[105] NIELSEN, M. How the backpropagation algorithm works. URL: http://neuralnetworksanddeeplearning.com/chap2.html, viitattu 20.10.2019.

[106] OLAH, C. Conv nets: A modular perspective. URL: https://colah.github.io/posts/2014-07-Conv-Nets-Modular/, viitattu 2.11.2019.

[107] OLAH, C. Understanding convolutions. URL: https://colah.github.io/posts/2014-07-Understanding-Convolutions/, viitattu 2.11.2019.

[108] OLAH, C. Understanding lstm networks. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/, viitattu 23.10.2019.

[109] OLENICK, D. Iot devices attacked faster than ever, ddos attacks up dramatically: Netscout. URL: https://www.scmagazine.com/home/security-news/cybercrime/iot-devices-attacked-faster-than-ever-ddos-attacks-up-dramatically-netscout/, viitattu 3.11.2019.

[110] OLSSON, J. 6lowpan demystified. URL: http://www.ti.com/lit/wp/swry013/swry013.pdf, viitattu 4.11.2019.

[111] PARR, T., JA HOWARD, J. The matrix calculus you need for deep learning. *ArXiv abs/1802.01528v3* (201).

[112] PASCANU, R., ÇAGLAR GÜLÇEHRE, CHO, K., JA BENGIO, Y. How to construct deep recurrent neural networks. *ArXiv abs/1312.6026v5* (2014).

[113] PASCANU, R., MONTÚFAR, G., JA BENGIO, Y. On the number of response regions of deep feedforward networks with piecewise linear activations. *ArXiv abs/1312.6098v5* (2014).

[114] POGGI, N. 24 cybersecurity statistics that matter in 2019. URL: https://preyproject.com/blog/en/24-cybersecurity-statistics-that-matter-in-2019/, viitattu 3.11.2019.

[115] RAJASEGARAR, S., LECKIE, C., JA PALANISWAMI, M. Anomaly detection in wireless sensor networks. *IEEE Wireless Communications 15*, 4 (2008), 34–40.

[116] RANSHOUS, S., SHEN, S., KOUTRA, D., HARENBERG, S., FALOUTSOS, C., JA SAMATOVA, N. F. Anomaly detection in dynamic networks: A survey. *WIREs Computational Statistics 7*, 3 (2015), 223–247.

[117] RAWAT, P., SINGH, K. D., CHAOUCHI, H., JA BONNIN, J. M. Wireless sensor networks: recent developments and potential synergies. *Journal of Supercomputing 68*, 1 (2014), 1–48.

[118] REDDI, S. J., KALE, S., JA KUMAR, S. On the convergence of adam and beyond. *ArXiv 1904.09237v1* (2019).

[119] RING, M., WUNDERLICH, S., SCHEURING, D., LANDES, D., JA HOTHO, A. A survey of network-based intrusion detection data sets. *Computers & Security* (2019).

[120] RUDER, S. An overview of gradient descent optimization algorithms. URL: `http://ruder.io/optimizing-gradient-descent/`, viitattu 20.10.2019.

[121] RUDER, S. An overview of gradient descent optimization algorithms. *ArXiv abs/1609.04747v2* (2017).

[122] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., BERG, A. C., JA FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[123] SAKURADA, M., JA YAIRI, T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. Julkaisusarjassa *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis* (2014), ACM, 4.

[124] SANDERSON, G. Neural networks. URL: `https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi`, viitattu 25.11.2019.

[125] SANDRYHAILA, A., JA MOURA, J. M. F. Discrete signal processing on graphs: Frequency analysis. *IEEE Transactions on Signal Processing 62*, 12 (2014), 3042–3054.

[126] SASAKI, Y. The truth of the f-measure. *Teach Tutor mater 1*, 5 (2007), 1–5.

[127] SHA, K., YANG, W. W. T. A., WANG, Z., JA SHI, W. On security challenges and open issues in internet of things. *Future Generation Computer Systems 83*, 1 (2018), 326–337.

[128] SHELBY, Z., JA BORMANN, C. *6LoWPAN: The wireless embedded Internet*, vol. 43. John Wiley & Sons, 2009.

[129] SHI, X., CHEN, Z., WANG, H., YEUNG, D.-Y., KIN WONG, W., JA CHUN WOO, W. Convolutional lstm network: A machine learning approach for precipitation nowcasting. Julkaisusarjassa *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (2015), MIT Press, 319–.

[130] SIDOROFF, I. Korkeauloitteisen datan dimensionaalisuuden pienentäminen. Pro-gradu tutkielma. Tietojenkäsittelytiede, Itä-Suomen yliopisto, 2010. URL: `http://cs.uef.fi/sipu/MSc_Thesis_Sidoroff_Ilja.pdf`, viitattu 22.11.2019.

[131] SOE, Y. N., FENG, Y., SANTOSA, P. I., HARTANTO, R., JA SAKURAI, K. Rule generation for signature based detection systems of cyber attacks in iot environments. *Bulletin of Networking, Computing, Systems, and Software 8*, 2 (2019), 93–97.

[132] SOLEIMANY, A. Introduction to deep learning 6.s191: Lecture 2. Massachusetts Institute of Technology. URL: `http://introtodeeplearning.com/2019/`, viitattu 23.10.2019.

[133] SUTSKEVER, I., VINYALS, O., JA LE, Q. V. Sequence to sequence learning with neural networks. *Advances in NIPS* (2014).

[134] TELGARSKY, M. Benefits of depth in neural networks. *ArXiv abs/1602.04485v2* (2016).

[135] TOFALLIS, C. A better measure of relative prediction accuracy for model selection and model estimation. *Journal of the Operational Research Society 66*, 8 (2015), 1352–1362.

[136] TOMIĆ, I., JA MCCANN, J. A. A survey of potential security issues in existing wireless sensor network protocols. *IEEE Internet of Things Journal 4*, 6 (2017), 1910–1923.

[137] TRAN, T. Machine learning part 5: Underfitting and overfitting problems. Massachusetts Institute of Technology. URL: https://chunml.github.io/ChunML.github.io/tutorial/Underfit-Overfit/, viitattu 27.11.2019.

[138] TRIEPELS, R., DANIELS, H., JA HEIJMANS, R. Detection and explanation of anomalous payment behavior in real-time gross settlement systems. Julkaisusarjassa *International Conference on Enterprise Information Systems* (2017), Springer, 145–161.

[139] TUOMINEN, H., JA NEITTAANMÄKI, P. *Tekoälyn perusteita ja sovelluksia*. Jyväskylän yliopisto, 2019. https://jyx.jyu.fi/handle/123456789/64975.

[140] VEEN, F. V. Neural network zoo. The Asimov Institute. URL: https://www.asimovinstitute.org/neural-network-zoo/, viitattu 27.11.2019.

[141] VELICKOVIC, P. Overview of neural network architectures for graph-structured data analysis. URL: https://www.cl.cam.ac.uk/~pv273/slides/UCLGraph.pdf, viitattu 1.11.2019.

[142] VISHWAKARMA, R., JA JAIN, A. K. A survey of ddos attacking techniques and defence mechanisms in the iot network. *Telecommunication Systems* (2019), 1–23.

[143] WU, Z., PAN, S., CHEN, F., LONG, G., ZHANG, C., JA YU, P. S. A comprehensive survey on graph neural networks. *ArXiv abs/1901.00596v3* (2019).

[144] XIE, M., HAN, S., TIAN, B., JA PARVIN, S. Anomaly detection in wireless sensor networks: A survey. *Journal of Network and Computer Applications 34*, 1 (2011), 1302–1325.

[145] XU, K., WU, L., WANG, Z., FENG, Y., WITBROCK, M., JA SHEININ, V. Graph2seq: Graph to sequence learning with attention-based neural networks. *ArXiv abs/1804.00823* (2018).

[146] YOUSEFI-AZAR, M., VARADHARAJAN, V., HAMEY, L., JA TUPAKULA, U. Autoencoder-based feature learning for cyber security applications. Julkaisusarjassa *2017 International joint conference on neural networks (IJCNN)* (2017), IEEE, 3854–3861.

[147] ZAKI, M. J., JA JR., W. M. *Data mining and analysis: Fundamental concepts and algorithms*. Cambridge University Press, 2014.

[148] ZEILER, M. D., JA FERGUS, R. Visualizing and understanding convolutional networks. Julkaisusarjassa *Computer Vision – ECCV 2014* (2014), Springer International Publishing, 818–833.

[149] ZENG, X., OUYANG, W., YANA, J., LI, H., XIAO, T., WANG, K., LIUA, Y., ZHOU, Y., YANG, B., WANG, Z., ZHOU, H., JA WANG, X. Crafting gbd-net for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 40*, 9 (2018), 2109–2123.

[150] ZHANG, A., LIPTON, Z. C., LI, M., JA SMOLA, A. J. *Dive into Deep Learning*, release 0.7 ed. 2019. URL: `http://www.d2l.ai`.

[151] ZHANG, C., PATRAS, P., JA HADDADI, H. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials* (2019).

[152] ZHANG, R. Making convolutional networks shift-invariant again. *ArXiv abs/1904.11486v2* (2019).

[153] ZHANG, Z., CUI, P., JA ZHU, W. Deep learning on graphs: A survey. *ArXiv abs/1812.04202v2* (2019).

[154] ZHOU, C., JA PAFFENROTH, R. C. Anomaly detection with robust deep autoencoders. Julkaisusarjassa *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), ACM, 665–674.

[155] ZHOU, J., CUI, G., ZHANG, Z., YANG, C., LIU, Z., JA SUN, M. Graph neural networks: A review of methods and applications. *ArXiv abs/1812.08434v4* (2019).

[156] ZIMEK, A., SCHUBERT, E., JA KRIEGEL, H.-P. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal 5*, 5 (2012), 363–387.

[157] ZOU, Y., ZHU, J., WANG, X., JA HANZO, L. A survey on wireless security: Technical challenges, recent advances and future trends. *Proceedings of the IEEE 104*, 9 (2016), 1727–1765.

A Bot-IoT datajoukon muuttujien kuvaus

Taulukko A.1: Bot-IoT -datajoukon muuttujat, kuvaukset ja arvojoukot.

Nro.	Muuttuja[1]	Kuvaus	Muoto ja tyyppi
1.	*pkSeqID*	Rivi-ID	Kokonaisluku (ordinaalinen)
2.	*stime*	Tallennuksen aloitusaika	Millisekunti, desimaaliluku (väli)
3.	*flgs*	Erilaisia liikenteen tilaa ja protokollia kuvaavia lippuja	Teksti (nominaalinen)
4.	*flgs_number*	Lippumuuttujan arvojoukon numeerinen esitys	Kokonaisluku (nominaalinen)
5.	*proto*	Kuljetuskerroksen protokolla	Teksti (nominaalinen)
6.	*proto_number*	Kuljetuskerroksen protokolla numeerisena	Kokonaisluku (nominaalinen)
7.	*saddr*	Lähettäjän osoite	Teksti (nominaalinen)
8.	*sport*	Lähettäjän portti	Kokonaisluku (ordinaalinen)
9.	*daddr*	Vastaanottajan osoite	Teksti (nominaalinen)
10.	*dport*	Vastaanottajan portti	Kokonaisluku (ordinaalinen)
11.	*pkts*	Datapakettien lukumäärä	Kokonaisluku (suhde)
12.	*bytes*	Tavujen lukumäärä	Kokonaisluku (suhde)
13.	*state*	Yhteyden tila	Teksti (nominaalinen)
14.	*state_number*	Tilamuuttujan numeerinen esitys	Kokonaisluku (nominaalinen)
15.	*ltime*	Viimeisen siirron aloitusaika	Millisekunti, desimaaliluku (väli)

[1]Muuttujat 30-43 ovat erikseen generoituja muuttujia.

16.	*seq*	Argus-ohjelman antama sekvenssinumero	Kokonaisluku (nominaalinen)
17.	*dur*	Yhteyden kesto	Millisekunti, desimaaliluku (väli)
18.	*mean*	Siirrettyjen pakettien siirtoaikojen keskiarvo	Millisekunti, desimaaliluku (väli)
19.	*stddev*	Siirrettyjen pakettien siirtoaikojen keskihajonta	Millisekunti, desimaaliluku (väli)
20.	*sum*	Siirrettyjen pakettien siirtoaikojen summa	Millisekunti, desimaaliluku (väli)
21.	*min*	Siirrettyjen pakettien lyhin siirtoaika	Millisekunti, desimaaliluku (väli)
22.	*max*	Siirrettyjen pakettien pisin siirtoaika	Millisekunti, desimaaliluku (väli)
23.	*spkts*	Pakettien määrä lähettäjältä vastaanottajalle	Kokonaisluku (suhde)
24.	*dpkts*	Pakettien määrä vastaanottajalta lähettäjälle	Kokonaisluku (suhde)
25.	*sbytes*	Tavujen määrä lähettäjältä vastaanottajalle	Kokonaisluku (suhde)
26.	*dbytes*	Tavujen määrä vastaanottajalta lähettäjälle	Kokonaisluku (suhde)
27.	*rate*	Siirrettyjen pakettien lukumäärä	Kokonaisluku (suhde)
28.	*srate*	Siirrettyjen pakettien lukumäärä lähettäjältä vastaanottajalle per sekunti	Kokonaisluku (suhde)
29.	*drate*	Siirrettyjen pakettien lukumäärä vastaanottajalta lähettäjälle per sekunti	Kokonaisluku (suhde)
30.	*TnBPSrcIP*	Lähetettyjen tavujen lukumäärä per IP	Kokonaisluku (suhde)
31.	*TnBPDstIP*	Vastaanotettujen tavujen lukumäärä per IP	Kokonaisluku (suhde)
32.	*TnP_PSrcIP*	Lähetettyjen pakettien lukumäärä per IP	Kokonaisluku (suhde)

33.	*TnP_PDstIP*	Vastaanotettujen pakettien lukumäärä per IP	Kokonaisluku (suhde)
34.	*TnP_PerProto*	Pakettien lukumäärä per protokolla	Kokonaisluku (suhde)
35.	*TnP_Per_Dport*	Pakettien lukumäärä per vastaanottajan portti	Kokonaisluku (suhde)
36.	*AR_P_Proto_P _SrcIP*	Keskimääräinen siirtonopeus (pkts/dur) per protokolla ja per lähettäjän IP	Desimaaliluku (suhde)
37.	*AR_P_Proto_P _DstIP*	Keskimääräinen siirtonopeus (pkts/dur) per protokolla ja per vastaanottajan IP	Desimaaliluku (suhde)
38.	*N_IN_Conn_P _SrcIP*	Sisääntulevien yhteyksien määrä per lähettäjän IP	Kokonaisluku (suhde)
39.	*N_IN_Conn_P _DstIP*	Sisääntulevien yhteyksien määrä per vastaanottajan IP	Kokonaisluku (suhde)
40.	*AR_P_Proto_P _Sport*	Keskimääräinen siirtonopeus (pkts/dur) per protokolla per lähettäjän portti	Desimaaliluku (suhde)
41.	*AR_P_Proto_P _Sport*	Keskimääräinen siirtonopeus (pkts/dur) per protokolla per vastaanottajan portti	Desimaaliluku (suhde)
42.	*Pkts_P_State_P _Protocol_P_DestIP*	Pakettien lukumäärä tilojen ja protokollien suhteen ryhmiteltynä per vastaanottajan IP	Kokonaisluku (suhde)
43.	*Pkts_P_State_P _Protocol_P_SrcIP*	Pakettien lukumäärä tilojen ja protokollien suhteen ryhmiteltynä per lähettäjän IP	Kokonaisluku (suhde)
44.	*Attack*	Luokitus	Kokonaisluku (nominaalinen)
45.	*Category*	Kategoria	Teksti (nominaalinen)[2]
46.	*Subcategory*	Alikategoria	Teksti (nominaalinen)[3]

[2]Normal, Reconnaissance, DoS, DDoS, Theft.

[3]Normal, Service_Scan, OS_Fingerprint, TCP, UDP, HTTP, Data_Exfiltration, Keylogging.