Janika Mäkipää

Vahvistettu oppiminen kotihoitajien reittisuunnittelussa

Tietotekniikan pro gradu -tutkielma

9. toukokuuta 2023

Jyväskylän yliopisto

Tietotekniikka

Tekijä: Janika Mäkipää

Yhteystiedot: janika.e.makipaa@student.jyu.fi

Ohjaaja: Raino Mäkinen

Työn nimi: Vahvistettu oppiminen kotihoitajien reittisuunnittelussa

Title in English: Reinforcement learning on home care route optimization

Työ: Pro gradu -tutkielma

Sivumäärä: 63+0

Tiivistelmä: Tekoälyn rooli yhteiskunnassamme kasvaa jatkuvasti. Tekoäly halutaan tuoda osaksi niin ihmisten arkielämän askareita kuin osaksi vaativia tietojärjestelmiä. Tässä pro gradu -tutkielmassa perehdytään vahvistettuun oppimiseen, joka on yksi koneoppimisen suuntaus. Vahvistettu oppiminen perustuu kokemusperäiseen oppimiseen, jossa agentit tekevät ympäristössään toimintoja, ja saavat palkkioita toiminnon kannattavuuden perusteella. Tutkielman päämääränä on selvittää, voidaanko kotihoitajien reitinkulkua optimoida vahvistetun oppimisen algoritmeilla. Tutkielmassa toteutettiin tapaustutkimus, jossa mallinnettiin kotihoitajien reitinkulkua potilaskohteisiin kahdella eri tavalla. Kotihoitajien reitinkulkua optimoitiin DQN-, A2C- ja PPO-algoritmien avulla. Lopputulokseksi tapaustutkimuksesta saatiin, että toinen malleista pystyi oppimaan kotihoitajien reitinkulun optimoinnin. Johtopäätöksenä voidaan todeta, että vahvistetun oppimisen algoritmeilla on mahdollista optimoida kotihoitajien reitinkulkua.

Avainsanat: Vahvistettu oppiminen, koneoppiminen, Q-oppiminen, työnjärjestelyongelma, Gymnasium

Abstract: The role of the artificial intelligent is growing every day in our society. We want to bring artificial intelligence into our everyday lives. This master's thesis investigates reinforcement learning which is an area of machine learning. The idea of reinforcement learning is that agent must learn to perform actions by trial and error. This thesis investigates if it's possible to optimize home care nurse's route by reinforcement learning algorithms. This thesis was implemented as case study where the routes of the home care nurses were modeled in

two different ways. The algorithms used for route optimizations were DQN, A2C and PPO. The result of the case study was that one of the models was able to learn to optimize the route of the home care nurses. As a conclusion it can be stated that it's possible to optimize home care nurse's route by reinforcement learning algorithms.

Keywords: Reinforcement learning, machine learning, Q-learning, job shop scheduling, Gymnasium

Kuviot

Sisältö

1 Johdanto

Yhteiskunnassamme ihmiset asettavat yhä enemmän odotuksia erilaisille älykkäille järjestelmille. Älykkäät järjestelmät halutaan tuoda osaksi niin arkielämän askareita kuin vaativiin tietojärjestelmiinkin, kuten terveydenhuolto alalle (Kulkarni 2012). Digitalisaation ja tekoälyä hyödyntävien sovellusten myötä pystytään vastaamaan yhä laajemmin erilaisiin yhteiskunnallisiin haasteisiin, sillä niiden avulla pystytään kehittämään uusia ratkaisuja esimerkiksi viime aikoina paljon uutisoituun työntekijöiden pulaan eri toimialoilla. Tekoälyn sovellusten myötä voidaan myös pohtia, voisiko tekoälyn avulla helpottaa paljon yhteiskunnassamme puhuttanutta hoitajapulaa terveydenhuoltoalalle.

Tässä pro gradu -tutkielmassa perehdytään vahvistettuun oppimiseen (engl. reinforcement learning). Vahvistettua oppimista tutkitaan erityisesti kotihoidon reittisuunnittelun optimoinnin näkökulmasta. Tutkielman alussa toteutetaan kirjallisuuskatsaus, jonka alussa tehdään yleiskatsaus koneoppimiseen. Tämän jälkeen kirjallisuuskatsauksessa määritellään mitä on vahvistettu oppiminen, millaisista periaatteista se koostuu, millaisia algoritmeja vahvistettuun oppimiseen kuuluu ja mitä on vahvistettu syväoppiminen (engl. deep reinforcement learning). Lopuksi tutkielmassa esitellään teoreettinen viitekehys, jota seuraa tapaustutkimus, jonka päämääränä on selvittää, voidaanko kotihoitajien työnkulkua optimoida vahvistetun oppimisen algoritmeilla.

Tutkielma toteutetaan yhteistyössä Tietoevry Oy:n kanssa, minkä johdosta tutkimusaiheen valinta pyrittiin tekemään niin yritykselle lisäarvoa tuovaksi kuin henkilökohtaisesti kiinnostavaksi. Tutkimusaihetta voidaan pitää tietotekniikan tutkimusalueella trendikkäänä ja ajankohtaisena, sillä yhä digitalisoituvassa yhteiskunnassamme tekoälyn rooli kasvaa jatkuvasti. Tekoälyä hyödyntäviä sovelluksia kehitetään jatkuvasti ja tekoäly halutaan myös tuoda osaksi suomalaista sosiaali -ja terveydenhuoltoa. Näin ollen tutkimusaihe kotihoitajien työnkulun optimoinnista tekoälyä hyödyntäen on erittäin ajankohtainen. Lisäksi tutkimusalueina koneoppiminen ja vahvistettu oppiminen ovat jatkuvasti kasvavia osa-alueita tietotekniikassa.

2 Koneoppiminen ja syväoppiminen

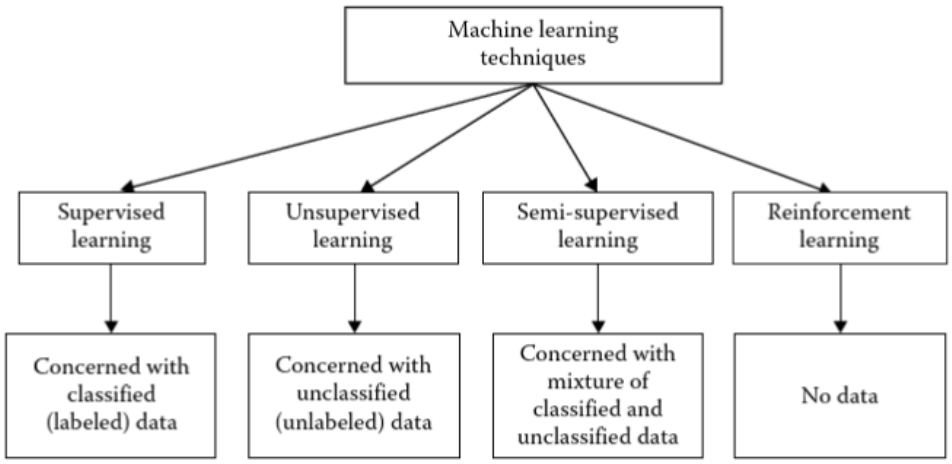
Tässä luvussa käsitellään mitä on koneoppiminen, millaisiin suuntauksiin se voidaan jakaa sekä lopuksi käydään läpi syväoppimisen ja neuroverkkojen käsitteitä. Viimeisessä alaluvussa käsitellään lyhyesti tunnetuimpia neuroverkkotyyppejä.

2.1 Koneoppiminen

Koneoppimisella (engl. machine learning) tarkoitetaan tekoälyn suuntausta, jonka päämääränä on saada tietokone oppimaan erilaisten kokemusten perusteella. Koneoppiminen pohjautuu useaan eri tieteensuuntaukseen, sillä se hyödyntää oppimisprosessissaan niin tiedonlouhintaa (engl. data mining), tilastotieteitä kuin psykologiaa. Tiedonlouhintaa käytetään koneoppimisessa oleellisen datan löytämiseen suuremmasta datajoukosta. Tilastotieteen avulla taas pyritään ymmärtämään toimintamalleja datan ja tilaston välillä. Lisäksi tietokoneen oppimisen on tarkoitus simuloida ihmisellekin tyypillistä kokemusperäistä oppimista. Psykologinen näkökulma koneoppimisen taustalla pyrkiikin ymmärtämään ihmisen erilaisia oppimismalleja (Kulkarni 2012).

Käytännössä koneoppiminen tapahtuu havaitsemalla toimintamalleja kerätystä datasta ja suorittamalla niiden avulla tietokoneelle annettuja tehtäviä (François-Lavet ym. 2018). Tietokoneen on tarkoitus oppia tekemään parempia päätöksiä aiemmin tehtyjen päätösten ja tehtävien pohjalta (Kulkarni 2012).

Koneoppiminen voidaan jakaa useampaan eri suuntaukseen. Koneoppimisen suuntauksia ovat ohjattu oppiminen (engl. supervised learning), ohjaamaton oppiminen (engl. unsupervised learning), puoliohjattu oppiminen (engl.semi-supervised learning) ja vahvistettu oppiminen (engl. reinforcement learning). Näitä koneoppimisen suuntauksia on havainnollistettu kuviossa 1. Kuten kuviosta voidaan huomata, tietokoneen opettamiseen käytettävän datajoukon muoto määrittelee yleensä, mitä koneoppimisen suuntausta, ja näin ollen myös algoritmeja, tullaan hyödyntämään. Eri koneoppimisen suuntaukset käyttävät erilaisia algoritmeja, mutta funktion approksimointi on oleellisessa osassa koneoppimisongelmien ratkaisua (François-Lavet ym. 2018).



Kuvio 1. Koneoppimisen suuntaukset (Mohammed, Bashier ja Mohammed 2017)

Koneoppimista voidaan soveltaa monilla eri tieteen aloilla. Yksi esimerkki koneoppimisen soveltamisesta on terveydenhuollon ala. Koneoppimisen avulla on mahdollista tunnistaa esimerkiksi potilaaseen liittyviä terveysriskejä hänen terveystietojensa perusteella. Sen lisäksi että koneoppimista voidaan käyttää terveysriskien tunnistamiseen, sitä voidaan käyttää terveydenhuollon alalla myös hallinnollisissa asioissa. Koneoppimisen avulla on pystytty myös ennustamaan osastoille tarvittavia potilaspaikkojen määriä (Sheikh 2018). Toinen esimerkki koneoppimisen soveltamisesta on genetiikan osa-alueella. Koneoppiminen soveltuu genetiikan tutkimusalalle, sillä geeneihin liittyvät datajoukot ovat usein laajoja. Koneoppimisen avulla voidaan oppia tunnistamaan esimerkiksi jokin tietty sekvenssi geeniperimästä (Libbrecht ja Noble 2015).

Vaikka koneoppimisen avulla voidaankin saavuttaa hyötyjä, sitä hyödyntäessä voidaan törmätä myös erilaisiin eettisiin kysymyksiin. Erityisesti yhteiskunnan toimivuuden kannalta kriittisillä osa-aluilla, kuten terveydenhuollon parissa, koneoppimisen tuottamat ratkaisut voivat olla eettisen näkökulman kannalta haastavia. Voidaan kuvitella esimerkki, jossa koneoppimista hyödynnettäisiin syöpätutkimusten parissa. Koneoppimisen tuottama malli antaisi datan perusteella tulokseksi, että syövän hoitaminen ei olisi kannattavaa, sillä kasvain on tyypiltään sellainen, johon hoidot eivät tehoa. Tällöin voidaan miettiä, että olisiko eettisesti oikein jättää yrittämättä parantaa syöpää, kun koneoppimisen malli niin ehdottaa (Sheikh 2018.)

2.2 Ohjattu oppiminen

Ohjattu oppiminen on yksi yleisimmin käytetyistä koneoppimisen suuntauksista. Sen ideana on opettaa tietokonetta toimimaan älykkäästi valmiiksi jäsennellyn (engl. labeled) harjoitusdatan perusteella. Ajatuksena on, että ohjatun oppimisen algoritmi oppii harjoitusdatan perusteella ennustamaan lopputuloksen samankaltaiselle datajoukolle kuin harjoitusdata on ollut (Kulkarni 2012). Jäsennelty harjoitusdata sisältää valmiiksi sekä syötteen että oikean lopputuloksen. Ohjattu oppiminen voidaan yksinkertaisimmillaan esittää funktiona $f : X \rightarrow Y$, jossa syöte $x \in X$ ja lopputulos $y \in Y$:

$$y = f(x)$$

Syötteenä toimiva jäsennelty data voi esimerkiksi olla valokuvajoukko, johon on valittu ainoastaan kuvia, jotka sisältävät talon tai auton kuvan. Tällöin algoritmin ei tarvitse erotella valokuvien joukosta niitä kuvia, jotka sisältävät tietyn piirteen, kuten talon tai auton (Mohammed, Bashier ja Mohammed 2017).

Ohjattu oppiminen voidaan jaotella kahteen eri suuntaukseen. Ensimmäinen suuntaus on luokittelu (engl. classification). Luokittelua käytetään sellaisen datan yhteydessä, joka on jaettavissa erilaisiin luokkiin. Esimerkiksi dokumentteja voidaan käsitellä luokittelun avulla, jolloin dokumentti jaetaan kuuluvaksi johonkin tiettyyn luokkaan esimerkiksi dokumentin tyypin mukaan. Yleisemmin voidaan sanoa, että luokittelua hyödyntävissä ohjatun oppimisen ongelmissa ohjelman luokittelija (engl. classifier) jaottelee kohteena olevalle datalleen erilaisia luokkia (Alpaydin 2016; Kulkarni 2012).

Luokittelussa voidaan hyödyntää muun muassa päätöspuuta (engl. decision trees). Päätöspuussa harjoitusdata asetetaan puun juureen, minkä jälkeen se jaetaan tilastollisten mallien mukaan erilaisiin attribuutteihin. Data kulkee juuresta puurakenteen läpi ja lopulta päätyy lehteen, joka edustaa oikeaa luokkaa. ID3 (Iterative Dichotomiser 3) sekä sen seuraaja C4.5 ovat esimerkkejä tilastollisista malleista, joita käytetään datan jakamisessa erilaisiin attribuutteihin (Mohammed, Bashier ja Mohammed 2017). Ohjatussa oppimisessa voidaan käyttää päätöspuiden lisäksi muun muassa sääntöön perustuvia luokittelijoita (engl. rule-based

classifiers), k:n lähimmän naapurin algoritmia (engl. K-nearest neighbors algorithm, k-NN) tai neuroverkkoja (engl. neural networks) (Mohammed, Bashier ja Mohammed 2017).

Toinen ohjatun oppimisen suuntaus on regressio. Regressiota käytetään silloin, kun data on jatkuvaa eikä dataa ei pystytä helposti luokittelemaan. Regressiossa pyritään ennustamaan jotakin muuttuvaa arvoa, kuten auton hintaa. Ohjatun oppimisen regression avulla voidaan ennustaa auton hinta, kun kerätään harjoitusdataksi tietoa myydyistä autoista ja niiden myyntihinnoista. Ennustuksen luotettavuuden kannalta on tärkeää valita oikeanlainen datajoukko ja datan käsittelymalli, jolla harjoittaa koneoppimista. Kun datajoukko ja käsittelymalli ovat valittu käyttötapaukseen nähden sopiviksi, voidaan luoda yleistetty malli, jonka avulla on mahdollista ennustaa sellaisiakin tapauksia, joita harjoitusdatassa ei ollut (Alpaydin 2016.)

2.3 Ohjaamaton oppiminen

Ohjaamaton oppiminen on koneoppimisen toinen suuntaus. Kun ohjatussa oppimisessa käytettävissä oli valmiiksi jäsennelty syöte ja lopputulos, ohjaamattomassa oppimisessa on käytettävissä ainoastaan syöte. Ohjaamattoman oppimisen päämääränä on etsiä syötteestä säännönmukaisuuksia, joiden perusteella voidaan arvioida lopputulos. Puheentunnistus on yksi kohdealue, jossa ohjaamatonta oppimista voidaan hyödyntää. Tällöin syötteenä voi toimia esimerkiksi radiosta kuuluva puhe, mikä edustaa syötettä, joka ei ole jäsenneltyä. Puheentunnistuksen täytyy oppia tunnistamaan tyypilliset piirteet radiokeskustelusta ohjaamattoman oppimisen keinoin (Alpaydin 2016.)

Klusterointi (engl. clustering) on yksi tapa tulkita syötteitä ohjaamattomassa oppimisessa. Klusteroinnin ideana on ryhmitellä tai muodostaa klustereita syötteistä erilaisten piirteiden perusteella. Aiemmassa alaluvussa mainittu dokumenttien luokittelu voidaan tehdä myös ohjaamattoman oppimisen avulla. Tällöin syötteenä voi toimia esimerkiksi erilaiset uutiset, jotka voidaan ryhmitellä erilaisiin klustereihin, kuten politiikkaan tai urheiluun liittyviin uutisiin. Tilastotieteissä tällaista tapaa kutsutaan sekoitemalliksi (engl. mixture models) (Alpaydin 2016.)

Ohjaamattoman oppimisen klusterointialgoritmeja on olemassa useita. Yksi tunnettu klusterointialgoritmi on k:n keskiarvon klusterointialgoritmi (engl. K-Means Clustering algo-

rithm), jossa data jaotellaan k osaan. K:n keskiarvon klusteroinnin perustana on jaotella n syötettä k klusteriin siten, että jokainen syöte kuuluu sitä lähimpään keskiarvon klusteriin. Muita ohjaamattoman oppimisen klusterointialgoritmeja ovat esimerkiksi Gaussin sekoitemalli (engl. Gaussian mixture model) ja Hidden Markov Model (Mohammed, Bashier ja Mohammed 2017.)

2.4 Puoliohjattu oppiminen

Perinteisten koneoppimisen suuntausten, ohjatun ja ohjaamattoman oppimisen, lisäksi puoliohjattua oppimista voidaan pitää yhtenä koneoppimisen suuntauksena. Puoliohjattu oppiminen yhdistää toiminnassaan ohjatun ja ohjaamattoman oppimisen suuntauksia. Sen päämääränä on parantaa näiden kahden perinteisen suuntauksen mukaisia laskentatapoja yhdistelemällä jäsenneltyä ja jäsentelemätöntä dataa (Van Engelen ja Hoos 2020.)

Puoliohjattua oppimista hyödynnetään erityisesti luokittelun yhteydessä. Tällaista jäsennellyn ja jäsentelemättömän datan yhdistämistä voidaan pitää hyödyllisenä suuntauksena erityisesti sellaisissa tapauksissa, kun jäsenneltyä dataa on vain vähän saatavilla. Tällöin luokittelua voi olla haastava toteuttaa ohjatun oppimisen menetelmillä luotettavasti, joten puoliohjattu oppiminen tarjoaa vaihtoehtoisen tavan ratkaista ongelman. Toisaalta puoliohjattua oppimista voidaan hyödyntää myös sellaisissa tilanteissa, joissa jäsenneltyä dataa on riittävästi, mutta jäsentelemättömän datan uskotaan tarjoavan oleellista lisätietoa ennustamisen tueksi (Van Engelen ja Hoos 2020.)

Yksi tunnettu tapa ratkaista puoliohjatun oppimisen ongelmia on käyttää graafipohjaisia algoritmeja. Niiden ideana on ratkaista puoliohjatun oppimisen ongelma siten, että graafin solmut edustavat jäsenneltyjä ja jäsentelemättömiä datapisteitä, kun taas graafin kulmat kertovat pisteiden välisestä samankaltaisuudesta. Jäsenneltyjä tietoja käytetään levittämään informaatiota graafin kaikille solmuille (Chapelle, Schölkopf ja Zien 2006.)

2.5 Syväoppiminen ja neuroverkot

Syväoppiminen (engl. deep learning) on koneoppimisen suuntaus, joka mahdollistaa yhä monimutkaisempien ohjatun oppimisen ongelmien ratkaisemisen. Syväoppimisen avulla on mahdollista esimerkiksi tunnistaa, mikä objekti valokuvassa on, vaikka syötteenä saadaan ainoastaan valokuvan pikselit. Lisäksi syväoppimisen tekniikoita voidaan hyödyntää puheentunnistuksessa, jossa se onkin vähentänyt huomattavasti puheentunnistuksessa ilmenevien virheiden määrää. Suurin osa syväoppimista hyödyntävistä sovelluksista perustuu parametrisoidun funktion approksimointiin.

Neuroverkkojen hyödyntäminen on tyypillinen tapa ratkaista syväoppimisen ongelmia (Ian Goodfellow 2016). Neuroverkko on malli, jonka idea perustuu biologisiin neuroverkkoihin, kuten ihmisen tai eläimen aivoihin. Neuroverkot muodostuvat toisiinsa kytkeytyneistä neuroneista, joiden tehtävänä on vastaanottaa dataa, prosessoida sitä sekä välittää tieto toisille neuroneille (Mohammed, Bashier ja Mohammed 2017). Neuroverkkojen hyödyllisyys verrattuna perinteisiin koneoppimisen menetelmiin tulee esiin, kun rakennetaan monimutkaisempia useiden kerrosten neuroverkkoja. Neuroverkot opetetaan harjoitusdatan perusteella oppimaan ennustamaan mahdollisimman hyvin oikea lopputulos syötteen avulla (Aggarwal 2018).

Syväoppimisen algoritmit pystyvät käsittelemään vielä monimutkaisempia datajoukkoja kuin ohjatun oppimisen algoritmit, sillä syväoppimisen avulla on mahdollista yhdistellä pienempiä datajoukkoja suuremmiksi ja kompleksisimmaksi kokonaisuudeksi. Tämä tapahtuu toteuttamalla neuroverkko, jossa on useita eri kerroksia (engl. layer), joissa datan prosessointi tapahtuu (Ian Goodfellow 2016). Ensimmäistä kerrosta kutsutaan syötekerrokseksi (engl. input layer), joka ottaa vastaan syötteen, kuten valokuvan pikselit. Syötekerroksen jälkeen data prosessoidaan mahdollisesti useiden piilokerrosten (engl. hidden layer) läpi. Nämä piilokerrokset tunnistavat pikseleistä vaihe vaiheelta erilaisia muotoja, kuten kulmia, ja lopulta muodostavat niistä kokonaisen kuvan. Lopputulos saadaan ulostulokerroksesta (engl. output layer), joka kertoo esimerkiksi, mikä objekti pikseleistä muodostuu (Alpaydin 2016).

Neuroverkoissa neuronit ovat kytkeytyneet toisiinsa synapsien avulla. Jokaisella synapsilla on paino (engl. weight), joka vaikuttaa seuraavaan neuroniin (Hervé Abdi ja Edelman

1999). Käytännössä synapsien painolla on merkittävä rooli neuroverkkojen hyödyllisyydessä, sillä oppiminen tapahtuu painoja vaihdellen. Neuroverkkojen harjoitusdata sisältää syöte-ulostulo -pareja, joiden avulla neuroverkkoa harjoitetaan. Harjoitusdatan avulla neuroverkko saa palautetta painojen sopivuudesta, sillä harjoitusdata antaa palautteen neuroverkolle sen mukaan, kuinka hyvin se on onnistunut ennustamaan ulostulodatan syötteen avulla. Mikäli ennustus ei ole ollut onnistunut, painoja muutetaan. Painojen muuttaminen tapahtuu matemaattisesti, ja useimmiten luotettavan lopputuloksen saamiseksi neuroverkkoja harjoitetaan useamman harjoitusdatan avulla ja näin ollen myös painojakin muutetaan useasti (Aggarwal 2018).

Aktivointifunktio on oleellisessa osassa neuroverkkoja, sillä se määrittelee, aktivoituuko neuroni vai ei. Yksinkertaisimmillaan aktivointifunktio voi olla jokin raja-arvo (engl. threshold), jonka ylittyessä neuroni aktivoituu ja lähettää datan muille neuroneille. Lähetettävä data koostuu useimmiten aktivoituneen neuronin syötteiden painotetusta summasta, joka lähetetään seuraavalle neuronille (Alpaydin 2016.)

Kaikkein yksinkertaisinta neuroverkkoa kutsutaan perseptroniksi (engl. perceptron). Se koostuu yhdestä neuronista, joka voi ottaa vastaan useampia syötteitä, mutta sillä on ainoastaan yksi ulostulo y. Neuroni ottaa vastaan painotetun summan sen syötteistä. Tämä summa voidaan esittää kaavana, jossa $m \in \mathbb{N}$, w edustaa synapsin painoa ja x neuronille annettavaa syötettä:

$$y = \sum_{j=1}^{m} w_j \cdot x_j = w_1 \cdot x_1 + ... + w_m \cdot x_m$$

Perseptroneilla aktivointifunktio vertaa syötteiden painotettua summaa ennalta määrättyyn raja-arvoon, joka määrittelee neuronin aktivoitumisen. Perseptronien yleisimmät aktivointifunktiot ovat porrasfunktio (engl. step function), merkkifunktio (engl. sign function) ja sigmoidifunktio. Kyseisillä funktioilla raja-arvo on määritelty nollaksi, minkä jälkeen aktivointifunktiosta riippuen ulostulo palauttaa joko arvon -1, 0 tai 1. Usein on kuitenkin hyödyllisempää asettaa raja-arvoksi jokin muu arvo kuin nolla. Tällöin neuroverkoissa hyödynnetään vakiotermiä (engl. bias), joka mahdollistaa raja-arvon kasvattamisen tai vähentämisen. Näin ollen voidaan muuttaa arvoa, joka aktivoi neuronin (Mohammed, Bashier ja Mohammed

2017.)

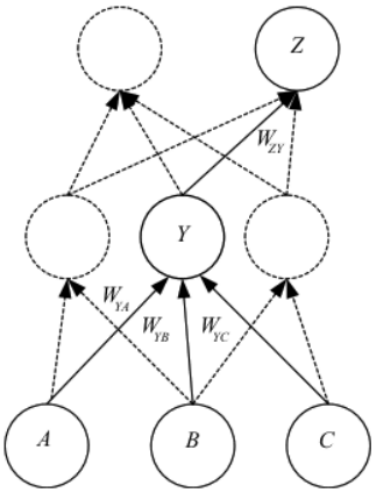
Koska neuroverkkojen, ja yleisestikin koneoppimisen ongelmien, oleellisessa roolissa on harjoitusdata, tällainen harjoitusdataan perustuva opettaminen tuo mukanaan myös haasteita. Kun koneoppimisen mallia harjoitetaan, voidaan harjoitusdatan perusteella laskea harjoitusdataan perustuva virhefunktio (engl. training error). Tämän lisäksi koneoppimisessa pyritään löytämään yleistetty virhefunktio, jota kutsutaan myös testidatan virhefunktioksi (engl. test error) (Ian Goodfellow 2016.)

Ylisovittaminen (engl. overfitting) ja alisovittaminen (engl. underfitting) ovat yleisiä haasteita koneoppimisessa. Mikäli harjoitusdataan perustuva virhefunktio ja yleisestetty virhefunktio tuottavat toisistaan selvästi eroavat arvot, voidaan sanoa, että malli on ylisovitettu. Tällöin malli toimii harjoitusdatalle hyvin, mutta testidatalle se tuottaa heikkoja tuloksia. Ylisovitettu malli on mukautunut liian tarkasti harjoitusdataansa eikä pysty sovittamaan osaamistaan erilaisiin datajoukkoihin. Ylisovittamisen lisäksi neuroverkko voi olla myös alisovitettu. Tällä tarkoitetaan sitä, että malli ei pysty saavuttamaan tarpeeksi matalaa virhearvoa harjoitusdataan perusteella. Alisovitetussa mallissa oppiminen on heikkoa (Ian Goodfellow 2016.)

2.6 Erilaisia neuroverkkoja

Neuroverkkoja voidaan muodostaa usealla eri tavalla. Usein neuroverkon muodostamistapa vaikuttaa myös sen käyttötarkoitukseen. Tyypillisin syväoppimisessa käytetty neuroverkkomalli on myötäkytkentä (engl. feedforward neural networks), joka antaa perustan useille neuroverkkoja hyödyntävälle sovellukselle.

Myötäkytketyssä neuroverkossa päämääränä on approksimoida jokin funktio mahdollisimman tarkasti. Myötäkytketyssä neuroverkossa data kulkee neuroverkon läpi syötekerroksen ja mahdollisten piilokerrosten läpi ulostulokerrokselle. Kyseisessä verkossa neuronit ovat yhdistetty toisiinsa siten, että saman kerroksen neuronit eivät ole kytköksissä toisiinsa. Tällöin datan ei ole mahdollista palata uudestaan samaan neuroniin, vaan data liikkuu kerrokselta toiselle, kuten voimme havaita Kuviosta 2 (Ian Goodfellow 2016). Yleisimmin käytetyssä myötäkytketyssä neuroverkossa voidaan pitää oletuksena, että kaikki kerroksen neuronit ovat kytkeytyneet seuraavan kerroksen neuroneihin (Aggarwal 2018).



Kuvio 2. Esimerkki myötäkytketystä neuroverkosta. Neuronit A, B ja C edustavat syötekerroksen neuroneita, jotka ovat kytketty piilokerroksen neuroniin Y. Jokaisella kytkennällä on paino W, joka määrittelee syöteneuronin vaikutuksen vastaanottavaan neuroniin. Neuroni Z edustaa neuroverkon ulostulokerrosta (Alpaydin 2016).

Toinen yleinen neuroverkkotyyppi on konvoluutioverkot (engl. convolutional neural networks), joita käytetään erityisesti kuvantunnistuksen yhteydessä. Konvoluutioverkot muodostuvat myötäkytketyn verkon tapaan syöte- ja ulostulokerroksesta, mutta ne eroavat myötäkytketystä verkosta siten, että konvoluutioverkkoon kuuluu yksi tai useampia konvoluutiokerroksia. Lisäksi konvoluutioverkot mahdollistavat matriisien käytön syötteenä, kun taas myötäkytketty verkko käsittelee vektoreita (Skansi 2018).

Nimensä mukaisesti konvoluutioverkot hyödyntävät toiminnassaan matemaattista lineaarista operaatiota, konvoluutiota. Yleisimmillään konvoluutio voidaan määritellä kahden funktion väliseksi operaatioksi, jossa käytetään reaaliarvoisia parametreja. Konvoluutioverkkojen avulla voidaan saavuttaa useita eri hyötyjä. Niiden avulla voidaan muun muassa käyttää samaa parametria (engl. parameter sharing) useampaan kertaan neuroverkon eri kohdissa. Näin ollen voidaan vähentää muistin tarvetta datan prosessoinnissa. Lisäksi konvoluutioverkoilla voidaan vähentää interaktioiden määrää datan käsittelyssä. Tämä tarkoittaa sitä, että lopputulos on mahdollista saavuttaa pienemmällä määrällä interaktioita ja sen avulla voidaan parantaa datan käsittelyn tehokkuutta (Ian Goodfellow 2016).

Myötäkytketyn ja konvoluutioverkkojen ohella toistuva neuroverkko (engl. recurrent neural network) on yleisesti käytetty neuroverkkotyyppi. Nimensä mukaisesti toistuvassa neuroverkossa ideana on, että neuroniin on mahdollista palata takaisin. Tällöin neuroverkkoon muodostuu ikään kuin silmukkamainen etenemistapa, joka eroaa aiemmista neuroverkkotyypeistä. Toistuvat neuroverkot soveltuvat erityisesti sekvenssisen datan käsittelyyn, kuten lauseiden muodostamiseen. Lauseiden käsittelyssä toistuvien neuroverkkojen avulla lauseet jaetaan pienemmiksi osiin eli yksittäisiksi sanoiksi. Tämän jälkeen neuroverkko lähtee liittämään sanoja toisiinsa yrittäen muodostaa mahdollisimman todennäköisen sanojen yhdistelmän (Skansi 2018).

3 Vahvistettu oppiminen

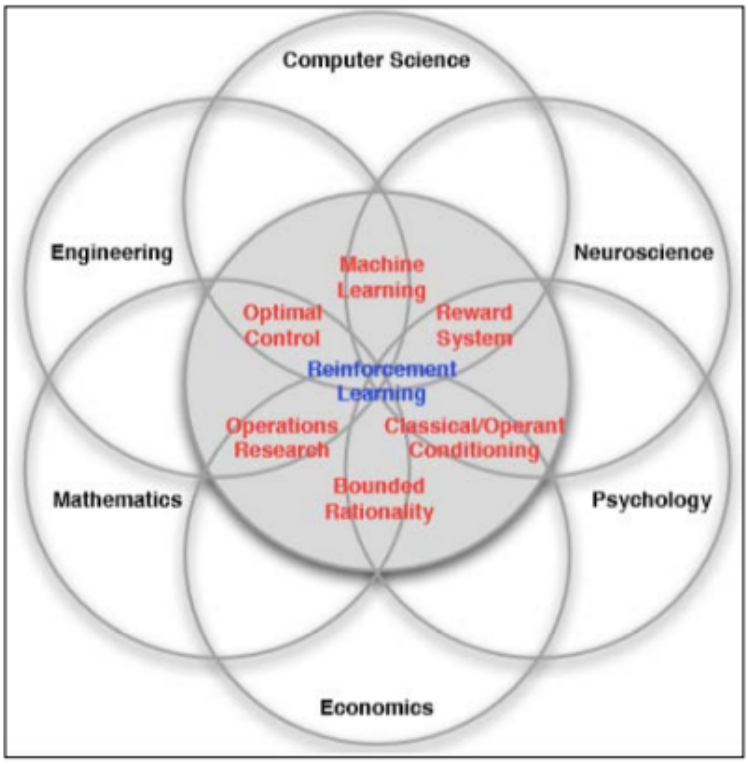
Tässä luvussa perehdytään tarkemmin tämän pro gradu -tutkielman kannalta oleellisimpaan koneoppimisen suuntaukseen, vahvistettuun oppimiseen. Ensimmäisessä alaluvussa käsitellään vahvistettua oppimista yleisellä tasolla, minkä jälkeen perehdytään vahvistetun oppimisen toimintaperiaatteeseen sekä sen taustalla vaikuttavaan Markovin päätöksentekoprosessiin. Kolmannessa, neljännessä ja viidennessä alaluvussa perehdytään vahvistetun oppimisen kannalta tärkeisiin käsitteisiin: palkkioon, toiminta- ja havaintoavaruuteen (engl. action space, observation space) sekä agentin toimintaperiaatteeseen (engl. policy). Kuudes alaluku käsittelee moniagenttista vahvistettua oppimista. Luvun lopuksi keskitytään vahvistetun syväoppimisen käsitteeseen.

3.1 Yleistä

Vahvistettu oppiminen on koneoppimisen suuntaus, jonka ajatuksena on mukailla ihmisellekin tyypillistä kokemusperäistä oppimista (Aggarwal 2018). Kun ohjatussa oppimisessa ideana on opettaa tietokonetta valmiiksi jäsennellyn syöte-tulos –parien avulla, vahvistetun oppimisen päämääränä on oppia kokemuksen perusteella tekemään oikeat ratkaisut ilman, että sille suoraan kerrotaan, mitä tulee tehdä. Koska suoraa ratkaisua ongelmiin ei kerrota, oppiminen tapahtuu ihmisellekin tyypillisesti yritys-erehdys -oppimiseen kautta. Oleellisena piirteenä vahvistetussa oppimisessa pidetään tavoitetta saavuttaa mahdollisimman suuri palkkio ympäristöltä saadun palautteen avulla (Sutton ja Barto 2018; Kulkarni 2012). Koska vahvistetussa oppimisessa oppiminen ei perustu vahvasti opetusdataan, se sopii tilanteisiin, jotka sisältävät epävarman ja ennalta vaikeasti määriteltävän tekijän, kuten muuttuvan ympäristön.

Vahvistettua oppimista ei myöskään voida pitää ohjaamattoman oppimisen suuntauksena, sillä vahvistettua oppimista määrittelee selkeästi idea maksimaalisesta palkkiosta. Ohjaamaton oppiminen kun taas pyrkii löytämään datan rakenteesta säännönmukaisuuksia. Kokeilemalla oppimista ja suurimman yhteenlasketun palkkion tavoittelua voidaan pitää vahvistetun oppimisen tunnusmerkkeinä (Sutton ja Barto 2018).

Vahvistetun oppimisen taustalla toimivasta ajatusmallista on nähtävissä yhteneväisyyksiä moniin eri tieteen aloihin. Tämä voidaan nähdä Kuviosta 3. Kaikista selvin vahvistetun oppimisen soveltaminen on jo tässäkin pro gradu -tutkielmassa todettu koneoppimisen osa-alue, joka on osa tietojenkäsittelytiedettä. Muita tieteenaloja, joilla on yhteistä vahvistetun oppimisen kanssa ovat neurotiede, psykologia, taloustiede, matematiikka sekä insinööritieteet (Lapan 2018).



Kuvio 3. Vahvistetun oppimisen taustalla vaikuttaa eri tieteen alat (Lapan 2018).

Esimerkiksi neurotieteissä on huomattu, että aivojen toiminta muistuttaa melko läheisesti vahvistetun oppimisen mallia. Psykologiassa taas tutkitaan ihmisen käyttäytymistä erilaisissa olosuhteissa. Kuten Kuviosta 3 nähdään, psykologiasta tuttu klassinen ehdollistuminen sivuaa vahvistetun oppimisen taustalla olevaa ajatusmallia. Taloustieteessä taas rajoitettu rationaalisuus (engl. bounded rationality) sivuaa vahvistettua oppimista yksilön päätöksenteon näkökulmasta (Lapan 2018).

Vahvistetussa oppimisessa haasteena on tasapainoilu, milloin on kannattavaa tutkia uusia mahdollisuuksia ja milloin taas on järkevää toimia jo opitun tiedon perusteella parhaimmalta

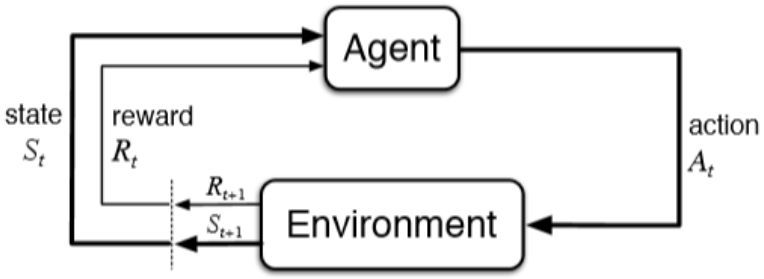
vaikuttavan vaihtoehdon mukaan (engl. exploration-exploitation dilemma) (François-Lavet ym. 2018). Tällainen tilanne voi tulla esiin esimerkiksi, kun roskia keräilevän akkukäyttöisen robotin tulee tehdä päätös, astuuko se vielä uuteen huoneeseen etsimään roskia vai päättääkö se palata takaisin akun lataamispisteelle. Robotti tekee päätöksensä jäljellä olevan akun varauksen perusteella sekä aiemmin opitun tiedon pohjalta, kuinka nopeasti ja helposti se löytää takaisin akun lataamispisteelle (Sutton ja Barto 2018).

3.2 Toimintaperiaate

Vahvistettuun oppimiseen kuuluu tyypillisesti malli ympäristön kanssa vuorovaikutuksessa toimivista agenteista. Agentilla tarkoitetaan oppijaa ja päätöksentekijää, kun taas ympäristöksi määritellään kaikki muu paitsi agentit (Sutton ja Barto 2018). Kyseisen mallin mukaan määriteltäessä aiemmin mainittu roskia keräilevä robotti edustaa oppijaa, kun taas ympäristöön luetaan kuuluvaksi itse robotin toimintaympäristö sekä kaikki muu data, kuten robotin jäljellä oleva akun varaus sekä arvio siitä, kuinka nopeasti robotti löytää akun lataamispisteelle. Vahvistetussa oppimisessa agenttien päämääränä on oppia, mikä toimintojen sarja saavuttaa kumulatiivisesti suurimman palkkion (François-Lavet ym. 2018).

Ympäristö ja agentti kommunikoivat toistensa kanssa jatkuvasti, sillä kuten kuviosta 4 huomataan agentin tehdessä toiminnon (engl. action) ympäristön tila (engl. state) muuttuu ja ympäristö antaa agentille palkkion (engl. reward) sen perusteella, kuinka kannattava toiminto oli. Tarkemmin ottaen jokaisella ajan hetkellä t= 0, 1, 2, 3,... agentti tekee toiminnon $A_t \in A$ ympäristön ollessa tilassa $S_t \in S$. Seuraavaa ajan hetkeä myöhemmin agentti vastaanottaa numeerisen palkkion $R_{t+1} \in R \subset \mathbb{R}$ ja ympäristön tilaksi muuttuu S_{t+1} (Sutton ja Barto 2018.)

Käytännössä tätä voidaan havainnollistaa esimerkiksi tilanteessa, jossa roskia keräilevä robotti päättää poimia esineen toimintaympäristössään. Tällöin ympäristön tila muuttuu ja ympäristö on tietoinen, että onko robotti poiminut roskaan vai jonkun muun esineen, ja mikä on akun varauksen tila toiminnon jälkeen. Mikäli robotin poimima esine osoittautuu roskaksi, ympäristö palkitsee sen onnistumisesta. Jos taas robotti on poiminut jonkun muun esineen kuin roskan, ympäristö antaa palautteen epäonnistuneesta toiminnosta.



Kuvio 4. Ympäristön ja agentin välinen vuorovaikutus vahvistetussa oppimisessa (Sutton ja Barto 2018).

3.3 Markovin päätöksentekoprosessi ja Markovin ominaisuus

Agenttien päätöksenteon taustalla vaikuttaa vahvasti Markovin päätöksentekoprosessi (engl. Markov decision process). Se on perinteinen mallintamiskeino sekvenssisestä päätöksenteosta, jossa ei haeta suurinta välitöntä palkkiota vaan maksimaalista yhteenlaskettua hyötyä (Sutton ja Barto 2018). Markovin päätöksentekoprosessia käytetään mallintamaan muuttuvia ympäristöjä, joissa päätöksentekijän tulee tehdä joukko toimintoja tietämättä niistä saatavaa palkkiota. Mikäli toiminto on kannattava, siitä seuraa positiivinen palkkio, kun taas epäsuotuisasta toiminnosta seuraa rangaistus (Ibe 2013).

Markovin päätöksentekoprosessi voidaan esittää matemaattisesti viisikkona (S, A, T, γ , R), jossa S on tila-avaruus, A on toimintojen avaruus, T : S $\times$ A $\times$ S $\rightarrow$ [0,1] on siirtymäfunktio (engl. transition function), $\gamma \in$ [0, 1] on vähennystekijä (engl. discount factor) ja R : S $\times$ A $\times$ S $\rightarrow \mathbb{R}$ on palkkiofunktio (engl. reward function) (Schwartz 2014). Mikäli tilojen, toimintojen ja palkkioiden joukot sisältävät äärellisen määrän jäseniä, päätöksentekoprosessia kutsutaan äärelliseksi Markovin päätöksentekoprosessiksi (Sutton ja Barto 2018).

Tarkemmin ottaen äärellisessä Markovin päätöksentekoprosessissa S voidaan määritellä äärelliseksi joukoksi ympäristön tiloja $s^1, ..., s^N$, jossa N on tila-avaruuden koko. Toisin sanoen voidaan todeta |S| = N. Joukko toimintoja A taas voidaan määritellä äärelliseksi joukoksi $a^1, ..., a^K$, jossa K on toimintojen avaruuden koko. Myös toiminnoille voidaan todeta, että |A| = K (Otterlo 2012). Siirtymäfunktiolla tarkoitetaan ehdollista joukkoa eri tilojen todennäköisyyksistä (François-Lavet ym. 2018). Alaluvuissa 3.3 perehdytään tarkemmin palkkiofunktion määritelmään.

Jotta päätöksentekojärjestelmän voidaan sanoa pohjautuvan Markovin päätöksentekoprosessiin, sen täytyy toteuttaa Markovin ominaisuus (engl. Markov property). Tällä tarkoitetaan tilannetta, että päätöksentekoon tarvitaan ainoastaan tieto nykyisestä tilasta. Markovin ominaisuus voidaan esittää muodossa:

$$P(S_t = s_t | S_{t-1} = s_{t-1}, ..., S_0 = s_0) = P(S_t = s_t | S_{t-1} = s_{t-1})$$

Tällöin tila S_t riippuu vain edellisestä tilasta S_{t-1} ja näin ollen voidaan sanoa, että järjestelmä täyttää Markovin ominaisuuden. Tila sisältää kuitenkin tiedot oleellisista toiminnoista, joita menneisyydessä tehty, mutta tulevaisuuden toiminnot eivät ole riippuvaisia niistä (Schwartz 2014; Kulkarni 2012).

3.4 Palkkio

Vahvistetussa oppimisessa palkkio ohjaa merkittävästi agentin toimintaa. Epämuodollisesti voidaan todeta, että agentin päämääränä on saavuttaa maksimaalisen suuri palkkio toiminnoistaan, joita se suorittaa. Maksimaalisella palkkiolla ei kuitenkaan tarkoiteta yksittäisen toiminnon jälkeistä välitöntä palkkiota vaan kaikkien toimintojen yhteenlaskettua palkkiota (Sutton ja Barto 2018; Kulkarni 2012).

Palkkiota voidaan pitää yhtenä vahvistetun oppimisen tunnusmerkkinä, sillä sen avulla pystytään ohjaamaan agentin oppimista. Ympäristöltä saatava palkkio kertoo agentille, oliko valittu toiminto kannattava. Esimerkiksi jos halutaan opettaa robottia keräämään roskia niin robottia voidaan palkita +1 kertoimella jokaisesta onnistuneesta roskan keräämisestä. Suurimmaksi osaksi aikaa tällaisessa tilanteessa robotti saa kuitenkin toiminnoistaan palkkioksi arvon 0, sillä aikaa kuluu liikkumiseen, joka taas ei ole tavoiteltu lopputulos. Mikäli robotti törmää esineeseen tai epäonnistuu muulla tavalla, sille voidaan antaa negatiivinen palkkio -1 merkiksi huonosta valinnasta.

Käytännössä agentin palkitseminen tapahtuu palkkiofunktion avulla. Tarkalleen ottaen palkkiofunktio $R : S \times A \times S \rightarrow \mathbb{R}$ määrittelee palkkion, joka annetaan tilojen muuttumisen perusteella. Yksinkertaisimmillaan palkkiofunktio voi olla kerroin 1, 0 tai -1, kuten yllä ole-

vasta esimerkistä voidaan havaita (Sutton ja Barto 2018; Otterlo 2012). Palkkiofunktio tulee kuitenkin määrittää ongelmakohtaisesti, sillä palkkiofunktion avulla voidaan epävirallisesti määrittää oppimisen päämäärä. Näin ollen kaikkiin ongelmiin ei sovi yksinkertaisin ja sama palkkiofunktio (Otterlo 2012).

Kirjallisuudessa käytetään palkkiofunktiosta myös nimitystä kustannusfunktio (engl. cost function). Tällöin tavoite maksimaalisen palkkion tavoittelusta muuttuukin päinvastaiseksi ja tavoitteena on saavuttaa mahdollisimman pieni kulu. Näin ollen kulufunktiolla yritetään opettaa agentti tekemään päätöksiä, joiden kulut on mahdollisimman pienet (Otterlo 2012).

3.5 Toiminta- ja havaintoavaruus

Agenttien tekemät toiminnot ja ympäristön palauttama informaatio ovat oleellinen osa vahvistetun oppimisen ympäristöä. Agenttien tekemät toiminnot muuttavat ympäristön tilaa. Tarkemmin tarkasteltuna nämä agentin tekemät toiminnat muodostavat toiminta-avaruuden. Toiminnot voidaan jakaa epäjatkuviin ja jatkuviin toimintoihin. Esimerkki epäjatkuvasta toiminnosta on agentin liikkuminen vasemmalle tai oikealle esimerkiksi shakin pelaamisessa (Lapan 2018). Epäjatkuvat toiminnot ovat melko harvinaisia tosielämän vahvistetun oppimisen sovelluksissa, sillä ne tarjoavat melko yksinkertaisia toimintoja (Zhu, Wu ja Zhao 2021).

Jatkuvat toiminnot taas ovat yleisempiä toimintoja vahvistetussa oppimisessa. Ne tarjoavat mahdollisuuden määritellä monimutkaisempia toimintoja (Zhu, Wu ja Zhao 2021). Jatkuvaksi määritellyissä toiminnoissa toimintoihin liittyy jokin arvo. Esimerkiksi auton ohjaamisessa tarvitaan tieto, mihin suuntaan ja missä kulmassa auton renkaiden tulisi kääntyä. Tässä huomataan ero epäjatkuviin toimintoihin, sillä niiden mukaan määritelty toiminto ‘käännä renkaiden suuntaa’ ei ole riittävän tarkka ohje, jotta voitaisiin saavuttaa haluttu lopputulos auton ohjaamisessa (Lapan 2018.)

Palkkion lisäksi ympäristö viestii agentin kanssa havaintojen avulla. Vaikka palkkiot ovat pääasiallinen viestintätapa agentin ja ympäristön välillä ja ne ohjaavat agentin oppimista kaikkein eniten, havaintojen avulla ympäristö viestii agentille, mitä ympäristössä tapahtuu. Käytännössä havainnot voivat olla yksinkertaisimmillaan joukko lukuarvoja tai toisaalta taas monimutkaisimmillaan esimerkiksi kuvainformaatiota sisältäviä tensoreita. Näiden havain-

tojen joukkoa kutsutaan havaintoavaruudeksi (Lapan 2018.)

3.6 Arvofunktio ja agentin toimintaperiaate

Vahvistetussa oppimisessa agentin tulee osata arvioida, kuinka optimaalisessa tilassa ympäristö on, kun tilan optimaalisuus määritellään odotettavissa olevan palkkion mukaan (Kulkarni 2012). Lähes kaikki vahvistetun oppimisen algoritmit hyödyntävät ympäristön tilan arvioinnissa arvofunktioita (engl. value function). Arvofunktio kertoo, mikä valinta tuottaa optimaalisen lopputuloksen pidemmän aikavälin näkökulmasta. Jotta arvofunktion avulla saadaan tietoa pidemmän aikavälin näkökulmasta, funktio huomioi tilat ja palkkiot, jotka todennäköisesti seuraa valitusta toiminnosta. On esimerkiksi mahdollista, että arvofunktion mukaan toiminnoksi kannattaa valita sellainen toiminto, joka tuottaa pienen välittömän palkkion, mutta tuottaa tämän jälkeen suurempia palkkioita tulevaisuudessa (Sutton ja Barto 2018). Tällainen tilanne voidaan havaita esimerkiksi, kun roskia keräilevä robotti liikkuu toimintaympäristössään kohti roskaa, mutta ei kuitenkaan vielä saavuta sitä. Tällöin robotti voidaan palkita liikkumisesta kertoimella 0, mutta saavutettuaan roskan robotti saa suuremman palkkion.

Vahvistetussa oppimisessa agentin päätöksenteon taustalla vaikuttaa aina jokin toimintaperiaate (engl. policy). Se määrittää agentin käytöksen tietyllä ajan hetkellä t. Käytännössä agentin toimintaperiaatteella tarkoitetaan sitä, että agentti kartoittaa tilat ja todennäköisyydet jokaiselle tarjolla olevalle toiminnolle (Schwartz 2014; Sutton ja Barto 2018). Roskia keräilevä robotti siis kartoittaa ennen seuraavan liikkeen ottamista, mikä siirtymä todennäköisesti muuttaisi ympäristön optimaaliseen tilaan. Tarkemmin tämän voi määritellä: jos ajan hetkellä t agentin toimintaperiaate on π, niin $\pi(a|s)$ on todennäköisyys $A_t = a$, jos $S_t = s$ (Schwartz 2014; Sutton ja Barto 2018).

Vahvistetussa oppimisessa agentin tulee oppia optimaalinen toimintatapa (engl. optimal policy). Optimaalisella toimintatavalla tarkoitetaan sitä toimintatapaa, joka on parempi tai yhtävertainen kaikkiin tarjolla oleviin toimintatapoihin nähden. Ei ole kuitenkaan rajattu, että optimaalisia toimintatapoja olisi vain yksi.

Mikäli agentti huomaa, ettei sen valitsema toiminto ollutkaan kannattava, se voi muuttaa toi-

mintatapaansa siten, että vastaavassa tilanteessa tulevaisuudessa se suorittaakin eri toiminnon (Kulkarni 2012). Esimerkiksi, jos roskia keräilevä robotti päättää palata akun lataamispisteelle huoneen x kautta, mutta huomaa, että reitti olisi ollut nopeampi huoneen y kautta, se voi muuttaa toimintatapaansa kulkea ensi kerralla huoneen y kautta palatessaan latauspisteelle.

3.7 Moniagenttinen vahvistettu oppiminen

Vahvistetun oppimisen ongelmia on mahdollista mallintaa siten, että ympäristössä toimii joko yksi tai useampia agentteja. Ympäristöä, jossa toimii yksi agentti, kutsutaan yhden agentin (engl. single-agent) vahvistetuksi oppimiseksi, kun taas useamman agentin mallia kutsutaan moniagenttiseksi (engl. multi-agent) vahvistetuksi oppimiseksi. Aiemmat alaluvut käsittelivät vahvistettua oppimista yhden agentin näkökulmasta, kun taas tämä alaluku perehtyy moniagenttiseen vahvistettuun oppimiseen.

Markovin päätöksentekoprosessin yleistämistä moniagenttiseksi tapaukseksi kutsutaan stokastiseksi peliksi. Sillä tarkoitetaan määritelmää, jossa agentteja toimii n kappaletta. Kun yhden agentin ympäristössä agentti tekee itsenäisiä toimintoja, moniagenttisessa ympäristössä agentit voivat tehdä yhteisiä toimintoja (engl. joint action). Lisäksi moniagenttisessa ympäristössä voidaan muodostaa agenttien yhteinen toimintaperiaate (engl. joint policy), jonka mukaan ne tekevät toimintoja. Agenttien on mahdollista tehdä yhteistyötä tai toisaalta ne voivat kilpailla toisiaan vastaan (Buşoniu, Babuška ja De Schutter 2010). Esimerkiksi roskia keräilevät robotit voivat joko kerätä roskia yhteistyössä tai ne voivat kilpailla toisiaan vastaan, kuka roboteista saavuttaa suurimman palkkion.

Moniagenttisessa vahvistetussa oppimisessa voidaan nähdä useita hyötyjä. Koska agentteja on useampi, ne oppivat samankaltaiset tehtävät nopeammin. Edistyneimmissä tapauksissa toiset agentit voivat toimia ympäristössä opettajina toisille agenteille. Näin voidaan saavuttaa parempi suorituskyky tehtävien tekemisessä. Lisäksi useamman agentin vahvistetussa oppimisessa voidaan toipua paremmin agentin epäonnistumisesta, koska toinen agentti voi hoitaa epäonnistuneen agentin tehtävän.

Hyötyjen lisäksi moniagenttisessa oppimisessa on haasteita. Haasteet liittyvät erityisesti las-

kennalliseen kompleksisuuteen, sillä tilojen ja toimintojen avaruus kasvaa moniagenttisessa oppimisessa eksponentiaalisesti. Useat vahvistetun oppimisen algoritmit hyödyntävät tilojen ja toimintojen avaruutta, minkä vuoksi ongelmasta tulee laskennallisesti vaikea. Näin ollen moniagenttisen vahvistetun oppimisen soveltaminen käytännön tilanteisiin voi osoittautua haastavaksi (Buşoniu, Babuška ja De Schutter 2010).

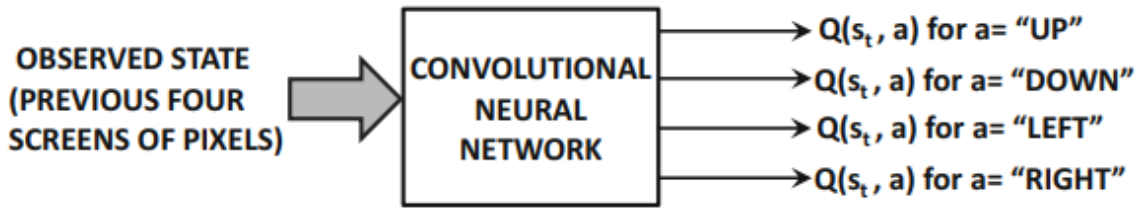
3.8 Vahvistettu syväoppiminen

Vahvistetun oppimisen ongelmia voidaan ratkoa syväoppimisen avulla. Tällöin vahvistettua oppimista kutsutaan vahvistetuksi syväoppimiseksi (engl. deep reinforcement learning). Vahvistettu syväoppiminen perustuu samoihin periaatteisiin kuin tavallinen vahvistettu oppiminen, mutta se hyödyntää toimintatavassaan neuroverkkoja. Vaikka useisiin vahvistetun oppimisen ongelmiin voidaan löytää ratkaisu perinteisten algoritmien avulla, osa ongelmista osoittautuu sen verran kompleksisiksi, että niissä todetaan paremmaksi lähestymistavaksi hyödyntää syväoppimista. Neuroverkkojen hyödyntäminen auttaa perinteisen vahvistetun oppimisen haasteeseen, jossa toiminta- ja tila-avaruudet kasvavat helposti todella suuriksi. Neuroverkkojen avulla tällaiset moniulotteiset ongelmat ovat helpommin ratkaistavissa (Zhu, Wu ja Zhao 2021; Aggarwal 2018).

Vahvistettu syväoppiminen on viime vuosina ollut melko suosittu tapa tutkia erilaisia päätöksentekoon liittyviä ongelmia. Vahvistettuun syväoppimiseen liittyen on tehty erilaisia tieteellisiä tutkimuksia ja innovaatioita, kuten vahvistetun svyäoppimisen hyödyntämistä robotin ohjaamisessa, älykkäässä kuljetuksessa (engl. intelligent transportation) sekä itseohjautuvien autojen ohjauksessa (Zhu, Wu ja Zhao 2021).

Videopelit ovat taas melko perinteinen esimerkki vahvistetun syväoppimisen soveltamisalueesta. Klassisena tapauksena voidaan pitää Atari 2600 -konsolia, joka tarjoaa alustan useille videopeleille. Yksinkertaistettuna Atari-pelien tapauksessa neuroverkon syötteenä toimii näytön pikselit joltakin ajan hetkeltä t. Vahvistetun oppimisen algoritmit taas ennustavat pikselien perusteella, mikä toiminto olisi pelin etenemisen kannalta kannattavaa tehdä.

Tarkemmin ottaen Atari-pelien esimerkissä ympäristön tila s_t voidaan esimerkiksi määritellä koostumaan neljän edeltävän näytön pikseleistä. Kuten kuviosta 5 voidaan huomata,



Kuvio 5. Q-verkko (engl. Q-network) Atari-videopelien tapaukseen (Aggarwal 2018).

pikselien käsittelemiseksi voidaan rakentaa konvoluutioverkko. Syötteenä toimiva joukko pikseleitä voidaan merkitä $\overline{X}_t$. Neuroverkko palauttaa ulostuloon $Q(s_t, a)$ jokaisen mahdollisen toiminnon a toimintojen joukosta A. Kuten kuviosta 5 nähdään, Atari-pelien tapauksissa nämä ulostulot ovat liikkuminen ylös, alas, vasemmalle ja oikealle. Konvoluutioverkkoa hyödynnetään konvertoimaan pikselit Q-arvoiksi (engl. Q-value). Q-oppimisesta (engl. Q-learning) on yksi vahvistetun oppimisen algoritmi, joka hyödyntää Q-arvoja. Tällaista verkkoa kutsutaan Q-verkoksi. Q-oppimista käsitellään tarkemmin luvussa 4.1. (Aggarwal 2018).

4 Vahvistetun oppimisen tunnettuja algoritmeja

Tässä luvussa perehdytään erilaisiin vahvistetun oppimisen algoritmeihin. Luvussa esitellään yksi tunnetuimmista vahvistetun oppimisen algoritmeista, Q-oppiminen. Tämän jälkeen käydään läpi muita vahvistetun oppimisen algoritmeja, jotka ovat oleellisia tämän pro gradu -tutkielman teknisen toteutuksen kannalta.

4.1 Q-oppiminen

Yksi merkityksellisimmistä tapahtumista vahvistetun oppimisen kehityksessä oli Q-oppimisen keksiminen (Sutton ja Barto 2018). Q-oppiminen on vahvistetun oppimisen algoritmi, jonka ideana on oppia Q-funktio (engl. Q-function) $Q : S \times A \rightarrow \mathbb{R}$, joka ennustaa jokaisesta toiminnosta $a \in A$ seuraavan palkkion ympäristön ollessa tilassa $S_t \in S$. Q-funktion päämääränä on löytää tapa, jolla saavutetaan mahdollisimman suuri yhteenlaskettu palkkio (Rummery ja Niranjan 1994). Q-oppimisen kehitti Chris Watkins vuonna 1989 (Watkins ja Dayan 1992). Oppiakseen optimaalisen tavan ennustaa palkkiota, Q-oppimisessa talletetaan arvot $Q(s,a)$ hakutaulukkoon siten, että jokaista tila-toiminto -paria vastaa yksi alkio. Tätä taulukkoa kutsutaan myös nimellä Q-taulukko (François-Lavet ym. 2018). Q-oppiminen voidaan esittää kaavana:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{(t+1)} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)]$$

(Sutton ja Barto 2018)

Q-funktion arvo päivitetään aina tilassa S_t jokaisen toiminnon A_t jälkeen. Q-funktiolle annetaan funktion nykyisen hetken arvo, johon lisätään yhteenlaskettu palkkio $R_{(t+1)}$ sekä kyseisen ajanhetken maksimaalinen palkkio $\max_a Q(S_{t+1},a)$. α arvolla tarkoitetaan vakiota, jonka avulla oppimisnopeutta voidaan säädellä. Vakioarvo γ viittaa vakioon, jolla vaikutetaan maksimaalisen palkkion painoarvoon (Sutton ja Barto 2018).

Q-oppiminen voidaan esittää algoritmina:

Valitse algoritmin parametrit: $\alpha, \gamma \in [0,1]$

Alustetaan Q-taulukko $Q(s,a)$ kaikille $s \in S, a \in A(s)$

Toista jokaiselle oppimisiteraatiolle:

Alustetaan tila $S_0,\ t = 0$

Toista kunnes S_{t+1} on lopputila

Valitaan Q-taulukon avulla toiminto A_t tilassa S_t

Suoritetaan toiminto $A_t,$ havainnoidaan S_{t+1}, R_{t+1}

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1},a') - Q(S_t,A_t)\right]$$

$t \leftarrow t+1$

Kuvio 6. Q-oppimisen algoritmi

Q-oppimisen vahvuutena on, ettei algoritmin hyödyntämiseen tarvita mallia (engl. model-free), kuten neuroverkkoa. Q-oppimisen haasteena on kuitenkin nopeasti kasvava Q-taulukon koko, mikäli tila-avaruus on suuri (Shi 2011).

4.2 Syvä Q-verkko

Syvä Q-verkko (engl. Deep Q-network, DQN) on vahvistetun oppimisen algoritmi, joka yhdistää Q-oppimisen ja syvät konvoluutioneuroverkot. Syvän Q-verkon avulla voidaan ratkaista monimutkaisia vahvistetun oppimisen ongelmia, sillä syvä Q-verkko pystyy oppimaan optimaalisen toimintatavan monimutkaisesta syötedatasta huolimatta. Tällaista monimutkaista syötedataa on esimerkiksi kuvien ja puheen tuottama sensoridata (Mnih ym. 2013). Syvän Q-verkon algoritmi kuuluu arvopohjaisiin algoritmeihin, jotka keskittyvät optimaalisen arvofunktion löytämiseen (Lapan 2018).

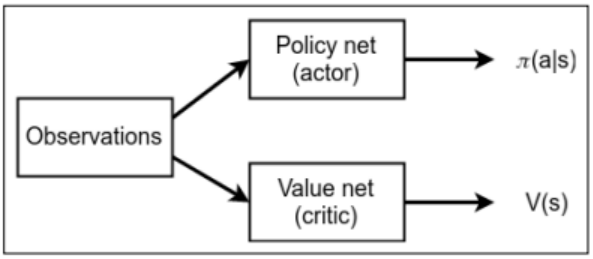
Esimerkki syvän Q-verkon soveltamisesta on tutkimus, josta syvän Q-verkon algoritmi tuli tunnetuksi. Tutkimuksessa haluttiin saada agentti oppimaan mahdollisimman monta Atari

2600 -peliä syvän Q-verkon avulla (Mnih ym. 2013). Tutkimuksen lopputuloksena agentti pystyi oppimaan syvää Q-verkkoa hyödyntäen useat pelit yhtä hyvin kuin ammattilaistason ihmispelaaja, vaikka agentille annettiin syötteeksi ainoastaan videopelin pikselit sekä pelin pistemäärät (Mnih ym. 2015). Tämä tutkimus osoittaa sen, kuinka syvä Q-verkko sopii moniulotteisen syötedatan sisältämien ongelmien ratkaisuun.

4.3 A2C

A2C (engl. Advantage Actor Critic) on synkroninen vahvistetun oppimisen algoritmi, joka kuuluu toimija-kriitikko -metodeihin. A2C on siten erilainen algoritmi kuin perinteiset vahvistetun oppimisen algoritmit, sillä se ei perustu pelkästään arvofunktioon. A2C-algoritmi yhdistää arvofunktion hyödyntämistä sekä agentin toimintaperiaatteen laskutavassaan (Sutton ja Barto 2018).

Toimija-kriitikko -metodeissa vahvistetun oppimisen ongelma jaetaan toimija- ja kriitikkokomponentteihin. Toimijalla viitataan agentin opittuun toimintatapaan, kun taas kriitikolla viitataan opittuun arvofunktioon. Toimija-kriitikko -metodit approksimoivat kumpaakin edellä mainittua toimintatapaa (Lapan 2018).



Kuvio 7. A2C-algoritmin taustalla oleva arkkitehtuuri (Lapan 2018).

Kuviossa 7 havainnollistetaan A2C-algoritmin taustalla olevaa arkkitehtuuria. Toimintaperiaatteihin perustuva komponentti (policy net) viittaa toimija-kriitikko -metodin toimija osuuteen. Toimintaperiaate palauttaa toimintoihin perustuvan todennäköisyysjakauman, kun taas arvoihin perustuva (value net) kriitikko palauttaa tiedon, kuinka kannattava toiminto oli (Lapan 2018). A2C-algoritmista on olemassa myös asynkroninen versio A3C (engl. Asynchronous Advantage Actor Critic), jossa voidaan harjoittaa agenttia samanaikaisesti useammassa

ympäristössä (Mnih ym. 2016).

4.4 PPO

PPO (engl. Proximal Policy Optimization) on vahvistetun oppimisen algoritmi, joka pohjautuu agentin toimintaperiaatteen optimointiin (Schulman ym. 2017). PPO eroaa aiemmin esitetyistä vahvistetun oppimisen algoritmeista siten, että se kuuluu gradienttityyppisten metodien joukkoon, kun taas esimerkiksi Q-oppiminen kuuluu arvopohjaisten algoritmien joukkoon (Aggarwal 2018).

Gradienttimetodeissa agentin toimintaperiaate parametrisoidaan neuroverkoksi, jonka painoja muokkaamalla voidaan ohjailla oppimista. Toisin sanoen neuroverkko tuottaa jokaiselle toiminnolle a, ympäristön tilassa $\overline{X}_t$ parametrilla $\overline{W}$ todennäköisyyden $P(\overline{X}_t, \overline{W}, a)$, jonka perusteella toiminto tulisi suorittaa. Jos toimintaperiaate osoittautuu huonoksi, myös tehty toiminto ja siitä seurannut palkkio on todennäköisesti huono. Neuroverkon painovektori $\overline{W}$ päivitetään jokaisella iteraatiolla sen perusteella, oliko toiminto optimaalinen (Aggarwal 2018).

PPO:n ajatuksena on ratkaista ongelma siitä, kuinka suurella toimintaperiaatteen muutoksella voidaan saavuttaa optimaalinen toimintaperiaate. Mikäli toimintaperiaatetta muutetaan liikaa, on vaarana, että opittua tietoa katoaa ja algoritmin tehokkuus kärsii. PPO on yksinkertaisempi metodi kuin TRPO (engl. Trust Region Policy Optimization), josta PPO on ottanut vaikutteita. PPO:n avulla voidaan ratkaista yksinkertaisemmin vahvistetun oppimisen ongelmia samalla hyödyntäen TRPO-metodin etuja, kuten vakautta ja luotettavuutta (Schulman ym. 2017).

5 Teoreettinen viitekehys

Tässä pro gradu -tutkielmassa päämääränä on tutkia, voidaanko kotihoitajien reitinkulkua optimoida vahvistetun oppimisen algoritmien avulla. Tässä luvussa määritellään teoreettinen malli, jonka mukaan reitin optimointia lähdetään käytännössä toteuttamaan.

5.1 Taustaa

Tutkielman päämääränä on mallintaa kotihoitajien reitinkulku vastaamaan mahdollisimman todenmukaisesti kotihoitajien reittioptimointia. Optimoinnin tarkoituksena on laskea hoitajalle paras mahdollinen reitti potilaskohteisiin päivän aikana huomioiden mm. hoitajan ammatillinen pätevyys, hänen käytössä oleva kulkuväline sekä mahdolliset potilaan asettamat rajoitteet.

Jotta tutkimusongelmaa voidaan lähteä selvittämään, optimointiongelma tulee ensin mallintaa. Yksi keino lähestyä optimointiongelma on mallintaa se töidenjärjestelyongelmaksi (engl. job shop scheduling problem). Töidenjärjestelyongelmassa työnkulku määritellään koostuvan työvoimasta (engl. resource), kuten koneista, ja töistä (engl. jobs). Ideana on aikatauluttaa työt koneille, siten että jokainen kone voi työskennellä vain yhden työn parissa kerrallaan, mutta työt jaetaan koneille mahdollisimman optimaalisesti esimerkiksi suoritusajan kannalta (Komm 2016).

Tarkemmin töidenjärjestelyongelma voidaan määritellä sisältävän m konetta $M_1, ..., M_m$ ja N työtä $J_1, ..., J_N$. Työt n_j koostuvat operaatioista O_{ij} (i =1, ..., n), jotka täytyy suorittaa järjestyksessä $O_{1j} \rightarrow O_{2j} \rightarrow ... \rightarrow O_{n_j,j}$. Lisäksi operaatio O_{ij} täytyy prosessoida $p_{ij} > 0$ aikayksikössä ilman keskeytyksiä saman koneen $\mu_{ij} \in M_1, ..., M_m$ parissa. Aikataulutus $S = (S_{ij})$ on määritelty operaatioiden O_{ij} aloitusaikojen mukaan. Aikataulua voidaan pitää toteuttamiskelpoisena, jos

- $S_{ij} + p_{ij} \leqslant S_{i+1,j}$ kaikille töille $j = 1, ..., N$ ja $i = 1, ..., n_j - 1$ ja
- $S_{ij} + p_{ij} \leqslant S_{u,v}$ tai $S_{u,v} + p_{u,v} \leqslant S_{ij}$ kaikille pareille operaatioita O_{ij}, O_{uv}, kun $\mu_{ij} = \mu_{uv}$

Toisin sanoen aikataulutuksen tulee siis noudattaa järjestystä, joka töille on määritelty sekä

jokainen kone voi prosessoida ainoastaan yhtä työtä kerrallaan (Brucker 2012).

Töidenjärjestelyongelma voidaan määritellä yksiagenttiseksi Markovin päätöksentekoprosessiksi. Tällöin ympäristön tila $s(t) \in S$ käsittää tiedon kaikkien työvoimien tilasta sekä, mikä on niiden töiden tila. Lisäksi $a(t) \in A$ kertoo, mitkä työt on päätetty käsitellä seuraavaksi. Yleisesti ottaen voidaan sanoa, että töidenjärjestelyn päämääränä on löytää toimintatapa $\pi^\star$, joka minimoi työstä syntyvät kulut pidemmällä aikavälillä (Gabel ja Riedmiller 2008):

$$\pi^\star := min \sum_{t=1}^{C_{max}} c(s,a,t)$$

Töidenjärjestely ei kuitenkaan ole ainoa tapa mallintaa kotihoitajien reitinkulkua. Toinen tapa lähestyä ongelmaa olisi mallintaa kotihoitajien työnkulku ajoneuvon reititysongelmana (engl. vehicle routing problem). Koska kyseessä on ongelma, jossa hoitajan täytyy käydä useassa eri paikassa sijaitsevassa potilaskohteessa päivän aikana, ongelman mallintamiskeinoksi sopii myös ajoneuvon reititys -tyyppinen malli.

Ajoneuvonreititysongelma on yksi tunnetuimpia optimointiongelmia, jonka päämääränä on selvittää optimaalinen reitti kulkuneuvolle eri solmujen kautta (Toth ja Vigo 2002). Yksinkertaisimmillaan ajoneuvonreititysongelmassa ajoneuvon tulee kuljettaa tavaroita useaan eri asiakaskohteeseen mahdollisimman optimaalisessa järjestyksessä ja palata kierroksen jälkeen takaisin lähtöpisteeseensä (Nazari ym. 2018). Näin ollen hoitajien reitinoptimoinnissa sekä kulkuneuvon reittiongelmassa voidaan havaita yhtäläisyyksiä.

Sekä ajoneuvonreititys että töidenjärjestelyongelma ovat molemmat kombinatorisia optimointiongelmia, jotka ovat luokiteltu NP-koviksi (engl. NP-hard) laskentaongelmiksi (Nazari ym. 2018; Gabel ja Riedmiller 2008). NP-kovalla laskentaongelmalla tarkoitetaan ongelmaa, jota ei voida ratkaista polynomisessa ajassa (Komm 2016). Molempia mallintamiskeinoja on hyödynnetty kirjallisuudessa vahvistetun oppimisen tutkimisen yhteydessä.

5.2 Kotihoitajien reitinoptimointi töidenjärjestelyongelmana

Tässä pro gradu -tutkielmassa kotihoitajien työnkulun mallintamiseen valitaan tavaksi töidenjärjestely, sillä siitä löytyy kattavasti tieteellisiä lähteitä ja niiden perusteella voidaan olettaa töidenjärjestelyn soveltuvan hyvin ongelman määrittämiseen.

Kotihoitajien työnkulku mallinnetaan töidenjärjestelyongelmana siten, että yksi hoitaja edustaa aina yhtä työvoimaa ja yksi potilaskäynti taas yhtä työtä. Näin ollen hoitajille saadaan jaettua päivän ajalle rajoitettu määrä töitä, jotka heidän tulee suorittaa. Jokaisen potilaskäynnin väliin asetetaan siirtymäaika, jolloin hoitaja siirtyy potilaskäynniltä toiseen. Kotihoitajien reitinkulussa päämääränä on käydä mahdollisimman monessa potilaskohteessa päivän aikana, minkä vuoksi kulujen minimointi on oleellinen osa optimointiongelmaa.
Määritelmä 5.2.1. Lähtötiedot tapauksesta voidaan määritellä kolmikkona $X = \langle \mathrm{N}, \mathrm{P}, \mathrm{R} \rangle$, jossa

- $N : \{N_1, ..., N_r\}$ on äärellinen joukko hoitajia,
- $P : \{P_1, ..., P_n\}$ on äärellinen joukko potilaskohteita,
- $R : \{R_{P_1}, ..., R_{P_n}\}$ on joukko rajoitteita

Hoitajista tiedetään heidän ammatillinen pätevyys eli ovatko he koulutukseltaan lähihoitajia vai sairaanhoitajia. Hoitajan ammatillinen pätevyys on oleellista tietää, sillä se määrittelee, millaisia sairaanhoidollisia hoitotoimenpiteitä hän voi tehdä ja näin ollen, mihin potilaskohteeseen hänen ammatilliset taitonsa soveltuvat. Lisäksi hoitajista on tiedossa, mikä hänen maantieteellinen sijaintinsa on koordinaateilla sekä millä kulkuvälineellä hän liikkuu ja kuinka nopeasti. Kulkuneuvovaihtoehtoja ovat kävellen ja autolla. Hoitajan sukupuoli täytyy myös olla tiedossa, sillä on mahdollista, että potilas on määritellyt rajoitteen, jonka mukaan hän haluaa esimerkiksi kotikäynnilleen vain naishoitajan.

Myös potilaskohteesta on tiedossa sen maantieteellinen sijainti koordinaattien avulla. Sen lisäksi potilaskohteisiin sisältyy ennalta määritelty tieto kotikäynnin kestosta. Potilaskohteella voi olla erilaisia rajoitteita. Joissain tapauksissa hoitotoimenpide voi vaatia tietyn ammatillisen pätevyyden, kuten sairaanhoitajan tutkinnon. Tällöin potilaskohteeseen tulee lähettää hoitaja, jonka ammatillinen pätevyys on riittävä hoitotoimenpiteeseen. Tämän lisäksi potilas voi määritellä vaatimuksen kotihoitajan sukupuolesta, kuten edellisessä kappaleessa todet-

tiin.

5.3 Kotihoitajien reitinoptimointi Markovin päätöksentekoprosessina

Kotihoitajien reitinkulun optimointi voidaan mallintaa Markovin päätöksentekoprosessina, sillä agenttien tekemät toiminnot ovat riippumattomia siitä, mikä aiempi toiminto on ollut ja missä tilassa ympäristö on aiemmin ollut. Näin ollen ongelma toteuttaa Markovin ominaisuuden ja ongelma voidaan mallintaa Markovin päätöksentekoprosessina.

Määritelmä 5.3.1. MDP Markovin päätöksentekoprosessi voidaan määritellä viisikkona $M = \langle \mathrm{D}, \mathrm{S}, \mathrm{A}, \mathrm{T}, \mathrm{R} \rangle$, jossa

- D on ympäristössä toimiva agentti,
- $S : S_1 \times ... \times S_n$ on äärellinen joukko tiloja,
- $A : A_1 \times ... \times A_n$ on joukko toimintoja,
- $T : S_m \times A_m \times S_m \rightarrow [0, 1]$ on siirtymien todennäköisyysfunktiot,
- $R : S \times A \rightarrow \mathbb{R}$ on välitön palkkiofunktio.

Markovin päätöksentekoprosessin näkökulmasta hoitajat edustavat ympäristössä toimivia agentteja.

Määritelmä 5.3.2. Agentti D_i voidaa määritellä parina $D_i = \langle \mathrm{C}, \mathrm{s} \rangle$, jossa

- $C : \{c_1, ..., c_r\}$ on joukko ammatillisia taitoja,
- $s : D_i \rightarrow \mathbb{R}$ on agentin keskimääräinen siirtymänopeus potilaskohteiden välillä liukulukuna

Kotihoitajien reitin optimoinnin kannalta tila S sisältää kaikki tarvittavat tiedot, joiden perusteella voidaan valita seuraava potilaskohde. Käytännössä tilaan sisältyy tieto potilaskohteen sijainnista, potilaskäynnin kestosta, rajoitteista sekä tieto, onko kotikäynti tehty jo.

Määritelmä 5.3.3. Tila S_i voidaa määritellä viisikkona $S_i = \langle \mathrm{lat}, \mathrm{lon}, \mathrm{d}, \mathrm{C}, \mathrm{h} \rangle$, jossa

- $lat : S \rightarrow \mathbb{R}$ on sijainnin leveyskoordinaatti liukulukuna,
- $lon : S \rightarrow \mathbb{R}$ on sijainnin pituuskoordinaatti liukulukuna,
- $d : S \rightarrow \mathbb{N}$ on potilaskäynnin kesto kokonaislukuna,
- $C : \{c_1, ..., c_k\}$ on joukko rajoitteita,

- $h : S_i \rightarrow \{True, False\}$ True, jos kotikäynti on jo tehty. Muissa tapauksissa False.

Toiminto $A_i j$ on saatavilla agentille D_k jos ja vain jos, agentilla ei ole vielä potilaskohdetta työn alla, agentin ammatilliset taidot täyttävät potilaskohteen rajoitteet ja potilaskohteessa ei ole vielä vierailtu.

5.4 Kirjallisuus käytännön toteutuksen tukena

Vahvistettu syväoppiminen on noussut tieteen saralla suosituksi tavaksi tutkia kombinatorisia ongelmia viime vuosina. Tutkimukset ovat keskittyneet kuitenkin enemmän perinteisiin ongelmiin, kuten kauppamatkustajaongelmaan tai graafiseen optimointiin. Vahvistettu syväoppiminen on melko uusi tapa ratkaista töidenjärjestelyongelmia (Tassel, Gebser ja Schekotihin 2021).

Tassel, Gebser ja Schekotihin (2021) ovat kehittäneet julkaisussaan 'A Reinforcement Learning Environment For Job-Shop Scheduling' OpenAI Gym -pohjaisen vahvistetun syväoppimisen ympäristön, jota käytetään työjärjestelyongelman ratkaisemiseksi. Julkaisussa on esitelty optimoitu ympäristö, jolla on pyritty mallintamaan teollisuusolosuhteita. Julkaisussa kehitetyssä ympäristössä tarkoituksen on vahvistetun syväoppimisen avulla oppia jakamaan työt aikarajoitetussa skenaariossa.

Tassel, Gebser ja Schekotihin (2021) esittelemässä vahvistetun syväoppimisen ympäristön toteutuksessa työjärjestelyongelma on mallinnettu yksiagenttisena ongelmana, jossa agentti toimii ikään kuin 'lähettinä,' joka valitsee kunkin työn alle otettavan työn tietyllä ajanhetkellä. Toiminta-avaruus on määritelty diskreettinä joukkona töitä $A = \{J_0, ..., J_{|J|-1}, No-Op\}$, jossa No-Op on yksi ylimääräinen toiminto siltä varalta, että koneelle ei ole valittavissa työtä. Jotta agentti ei valitsisi No-Op vaihtoehtoa sellaisissa tilanteissa, joissa työtä olisi tarjolla, merkitään toiminnot joko sallituiksi tai ei-sallituiksi. Mikäli agentti valitsee ei-sallitun toiminnon, neuroverkon palauttama arvo muutetaan mahdollisimman pieneksi negatiiviseksi numeroksi.

Tassel, Gebser ja Schekotihin (2021) esittelemässä tutkimuksessa palkkiofunktio esitettiin töille allokoidun keston ja koneiden tyhjän odotusajan erotuksena. Toisin sanoen tämä voi-

daan esittää kaavana:

$$R(s,a) = p_{aj} - \sum_{m \in M} empty_m(s,s')$$

jossa s edustaa nykyistä ympäristön tilaa ja s' edustavat seuraavaa tilaa, a edustaa operaatiota j^{th} operaatioista J_a prosessointiajalla p_{aj}. Funktio $empty_m(s,s')$ palauttaa arvon siitä, kuinka kauan kone m on ollut tyhjäkäynnillä siirryttäessä tilasta s tilaan s' .

Tassel, Gebser ja Schekotihin (2021) toteuttamassa tutkimuksessa agenttia harjoitettiin luvussa 4.4 esitellyllä gradienttimetodeihin kuuluvalla PPO-algoritmia. Tutkimustuloksia verrattiin kirjallisuudesta tunnettuihin FIFO- (engl. First in first out) ja MWKR (engl. Most Work Remaining) -menetelmiin. Tutkimuksen lopputuloksena havaittiin, että tutkimuksessa rakennettu optimoitu ympäristö suoriutui töidenjärjestelyongelmasta paremmin kuin tunnetut FIFO- ja MWKR -menetelmät.

6 Toteutus

Tässä luvussa käydään läpi, kuinka tekninen toteutus kotihoitajien reitinoptimointiongelmaan on tehty. Aluksi luvussa käsitellään kotihoitajien reitin optimointiongelman teknistä toteutusta ja ongelman mallintamista. Tämän jälkeen käydään läpi käyttötapaus, jota on lähestytty kahdesta eri näkökulmasta, ja lopuksi analysoidaan, millaisia lopputuloksia tutkimuksesta saatiin.

6.1 Tekninen toteutus

Kotihoitajien reittioptimointiongelma on mallinnettu Python-ohjelmointikielellä käyttäen Googlen Colaboratory -alustaa. Toteutuksessa on hyödynnetty datan käsittelyyn ja laskuoperaatioihin erilaisia Python-kirjastoja, kuten pandas- ja numpy -kirjastoja, sekä datan havainnollistamiseen kartalla on käytetty folium-kirjastoa.

Oleellisessa osassa kotihoitajien reitinoptimointiongelman toteutuksessa on ollut Gymnasium-rajapinta. Gymnasium on Farama Foundationin omistama vahvistetun oppimisen työkaluja tarjoava rajapinta. Ennen Farama Foundationin omistukseen siirtymistä Gymnasium tunnettiin nimellä OpenAI Gym. Gymnasium tarjoaa monipuoliset työkalut erilaisten vahvistetun oppimisen ongelmien mallintamiseen. Se tarjoaa sekä valmiita vahvistetun oppimisen ympäristöjä että itse mukautettavia ympäristöjä, joihin vaativammat ongelmat on mahdollista itse rakentaa (*Gymnasium*). Tässä pro gradu -tutkielmassa kotihoitajien reittioptimointiongelma on rakennettu mukautettuna ympäristönä, joka toteuttaa Gymnasiumin tarjoaman rajapinnan.

Toinen oleellinen osa kotihoitajien reitin optimointiongelman teknisessä toteutuksessa on ollut Stable baselines 3 -kirjaston algoritmit. Stable baselines 3 on avoimen lähdekoodin ratkaisu, joka tarjoaa erilaisia vahvistetun oppimisen algoritmeja (Raffin ym. 2021). Kotihoitajien reitin optimointiongelmassa agenttien kouluttamiseen on käytetty Stable baselines 3:n tarjoamia algoritmeja, kuten DQN, A2C ja PPO.

6.2 Malli

Kotihoitajien reitin optimointiongelma on mallinnettu olio-ohjelmointipohjaisesti muodostaen omat luokat ympäristölle, asiakkaalle ja agentille. Tulevissa alaluvuissa perehdytään tarkemmin kunkin luokan malliin.

6.2.1 Ympäristö

Kotihoitajien toimintaympäristö on mallinnettu "HomecareEnvironment"nimisessä luokassa. Kyseinen luokka on mukautettu Gymnasium-ympäristö, joka toteuttaa Gymnasium-rajapinnan. Gymnasium-rajapinta voidaan määritellä:

```
class gymnasium.Env:
    def step(self, action: ActType) ->
        tuple[ObsType, SupportsFloat, bool, bool, dict[str, Any]])
    def reset(self, *, seed: int | None = None,
        options: dict[str, Any] | None = None) ->
        tuple[ObsType, dict[str, Any]]
    def render(self) -> RenderFrame | list[RenderFrame] | None
```

Mukautetun toimintaympäristön avulla voitiin hyödyntää Gymnaisum-rajapinnan tarjoamia funktioita ympäristön luomiseen sekä agentin harjoittamiseen. Käytännössä ympäristöön on rakennettu init-, step-, reset- ja render-funktiot. Init-funktio alustaa ongelman sekä määrittelee ongelman toiminto- ja havaintoavaruudet (engl. observation space). Kotihoitajien reittiongelmassa ympäristössä on agentille mahdollista toteuttaa viisi erilaista toimintoa. Näihin toimintoihin tutustutaan tarkemmin alaluvussa 6.3.

Step-funktio kuvastaa agentin ajanhetkellä t tekemää toimintoa. Step-funktiossa valitaan agentin seuraava potilaskohde, lasketaan palkkio sekä määritellään havainto. Potilaskohde valitaan sen perusteella, mikä toiminto agentille on valittu. Reset-funktiota käytetään tyhjentämään optimointiongelman arvot lähtötilanteeseen. Render-funktion avulla taas tulostetaan havainnot agenttien toiminnasta konsoliin.

6.2.2 Asiakas

Asiakkaaseen liittyvä tietosisältö on mallinnettu ”Customer” nimisessä luokassa. Asiakkaalla tarkoitetaan tässä yhteydessä kotihoitajan asiakkaana toimivaa potilasta. Asiakkaaseen liittyvään tietosisältöön kuuluu:

- Asiakkaan yksilöivä tunnus
- Totuusarvo, onko asiakaskohteessa jo käyty
- Asiakaskohteen maantieteellinen sijainti koordinaatteina
- Asiakkaaseen liittyvät rajoitteet
- Tapaamisen kesto

6.2.3 Agentti

Agenttiin liittyvä tietosisältö on mallinnettu ‘’Agent” nimiseen luokkaan. Tässä tapauksessa agentilla tarkoitetaan hoitajaa. Agent-luokan ominaisuuksiin kuuluu:

- Agentin yksilöivä tunnus
- Ammatillinen pätevyys
- Agentin liikkumisnopeus
- Agentin maantieteellinen sijainti koordinaatteina
- Tieto agentin käytössä olevasta kulkuvälineestä
- Tieto reiteistä ja etäisyyksistä
- Agentin lähtösijainnin koordinaatit
- Tieto asiakkaan yksilöintitunnuksesta, kenen luona agentti on vierailemassa

6.3 Käyttötapauksen läpikäynti

Tässä alaluvussa käydään läpi teknisen toteutuksen käyttötapaus. Käyttötapausta on lähestytty kahden eri lähestymistavan mukaan hieman erilaisten havaintoavaruuksien ja palkkiofunktioiden näkökulmasta. Aluksi käydään läpi kummallekin lähestymistavalla yhteiset taustatiedot, minkä jälkeen käydään kumpikin lähestymistapa tarkemmin läpi.

6.3.1 Taustatiedot

Kummassakin lähestymistavassa ohjelmakoodin alussa ladataan tarvittavat kirjastot käyttöön sekä haetaan taustadataa käyttötapauksen läpiviemiseksi. Taustadataan kuuluu joukko osoitetietoja, jotka asetetaan potilaskohteiden osoitetiedoiksi. Osoitetiedot on haettu käyttämällä Google Mapsin tarjoamaa rajapintaa, joka palauttaa katuosoitteita sekä niiden koordinaatit. Tässä pro gradu -tutkielmassa osoitetiedoiksi on haettu joukko Jyväskylän kaupungin ja sen lähialueiden satunnaisia katuosoitteita. Joukon koko tässä käyttötapauksessa on 100 potilaskohdetta.

Kirjastojen lataamisen ja taustadatan keräämisen jälkeen ohjelmakoodi alustaa optimointiongelmassa käytetyn ympäristön. Alustettaessa ympäristö asettaa optimointiongelman kooksi 100 potilaskohdetta sekä pohjustaa asiakkaaseen liittyvän datan. Asiakasdata asetetaan taulukkoon, johon asiakkaan liittyvä tietosisältö lisätään. Asiakkaasta on tiedossa siis potilaskohteenyksilöivä tunnus, totuusarvo onko asiakaskohteessa jo käyty, maantieteellinen sijainti, rajoitteet sekä tapaamisen kesto. Tapaamisen kestoksi määritellään satunnainen arvo 10-30 minuutin väliltä.

Asiakkaan datan alustamisen jälkeen ohjelmakoodi alustaa agentteihin liittyvän datan. Agenttien määräksi asetetaan vakioarvo 5. Jokaiselle agentille määritellään sama maantieteellinen lähtöpiste. Tämän avulla mallinnetaan sitä, että kotihoitajan lähtisivät joka päivä samasta lähtöpisteestä, kuten toimistolta liikkeelle. Agentin kulkuvälineeksi valitaan satunnaisesti joko auto tai polkupyörä. Autolla liikkuessaan agentin nopeus on 50 km/h ja pyörällä liikkuessaan 20 km/h.

6.3.2 Lähestymistapa 1

Ensimmäisessä lähestymistavassa havaintoavaruus on rakennettu Box-tietorakenteeseen, jonka arvojen alarajaksi asetetaan 0 ja ylärajaksi 1000. Havaintoavaruus on 7-ulotteinen ja sen sisältämät arvot ovat liukulukuja. Havaintoavaruus perustuu agentin matkustusaikaan, läpimenoaikaan sekä asiakaskäynnin pituuteen ja ne talletetaan havaintotaulukkoon. Ympäristön alustamisen jälkeen Stable baselines 3 -kirjaston make_vec_env -funktio muodostaa vektroisoidun ympäristön.

Ohjelmakoodin step-funktiossa simuloidaan agentin tekemää toimintoa, jonka jälkeen ympäristö siirtyy uuteen tilaan. Step-funktiossa käydään jokainen agentti läpi silmukassa. Ensiksi agentille haetaan seuraava potilaskohde. Ensimmäisessä lähestymistavassa ohjelmakoodiin on toteutettu 5 erilaista laskentatapaa potilaskohteen valitsemiseen. Nämä viisi erilaista laskentatapaa ovat:

- 0 = Valitaan maantieteellisesti lähin potilaskohde
- 1 = Valitaan potilaskohde, jonka suoritusaika (matkustusaika + asiakaskäynti) on lyhyin
- 2 = Valitaan potilaskohde, jonka asiakaskäynti on ajallisesti lyhyin
- 3 = Valitaan potilaskohde satunnaisesti viidestä maantieteellisesti lähimmästä kohteesta
- 4 = Valitaan potilaskohde, jonka maantieteellinen sijainti on kaikkein kauimpana

Potilaskohteen valitsemistapa on mahdollista asettaa joko manuaalisesti tai ympäristön malliin pohjautuvalla Stable baselines 3 -kirjaston predict-funktiolla. Tässä tutkielmassa kuvatussa tapauksessa potilaskohde valitaan predict-funktion tuottaman arvon perusteella. Potilaskohteen maantieteellisiä etäisyyksiä laskettaessa hyödynnetään Haversine-algoritmia, joka laskee etäisyyden eri koordinaattien välillä. Haversine-algoritmia hyödynnettäessä välimatka lasketaan isoympyräreittiin (engl. great-circle distance) perustuen, joten välimatka ei ota kantaa tieverkoston kautta lyhimpään etäisyyteen.

Kun potilaskohde on saatu valittua, päivitetään agenttiin liittyvä data. Agentin kulkema matka, sen sijainnin koordinaatit, matkustusaika sekä potilaskohteen kokonaissuoritusaika. Lisäksi päivitetään totuusarvo, että potilaskohteessa on käyty.

Step-funktion lopulla siirrytään ympäristön tilan muutoksen logiikkaan. Agentti saa ympäristöltä palkkion sen mukaan, kuinka kannattavan potilaskohteen se onnistui valitsemaan. Palkkion laskenta perustuu käyttötapauksen ensimmäisessä lähestymistavassa agentin potilaskohteen läpimenoaikaan. Läpimenoaika voidaan esittää kaavana:

Kokonaispalkkio = (Agentin nopeus * etäisyys + seuraavan asiakaskäynnin kesto) - rangaistus

Kokonaispalkkio muuttuu aina agentin tekemän toiminnon jälkeen. Tässä lähestymistavassa päädyttiin lisäämään palkkion laskentaan rangaistusfunktio agentille, mikäli se valitsee liian läheisen potilaskohteen toisen agentin kanssa. Tällä pyrittiin vähentämään sitä, että eri agentit valitsisivat toisiaan lähellä olevia potilaskohteita. Rangaistus lasketaan siten, että ensiksi agenteista muodostetaan parit, minkä jälkeen parien maantieteellinen etäisyys kilometreinä laitetaan ''distances" nimiseen listaan. Tämän jälkeen lista etäisyyksistä korotetaan toiseen potenssiin. Jokainen listan jäsen iteroidaan läpi ja lasketaan summa alla olevan laskutavan mukaan. Lopuksi rangaistukseksi palautetaan maksimiarvo listasta, jonka arvot ovat 1 ja aiemmin laskettu summa. Rangaistuksen laskutapa voidaan esittää Python-koodina:

```
distances = numpy.power(distances, 2)
total = sum([1/(100*x) if x != 0 else 1/(100 * (0.001 ** 2)) for x
  in distances])
return max([1, total])
```

Toinen skenaario, jossa agenttia rangaistaan huonosta valinnasta, on tapaus, jolloin agentti rikkoo potilaaseen liittyviä rajoitteita. Tällöin agentti valitsee esimerkiksi potilaskohteen, johon agentin ammatillinen pätevyys ei ole riittävä. Rajoitusten rikkomisesta annetaan rangaistukseksi aina vakioarvo -2. Tällaisessa tapauksessa aiemmin esiteltyä rangaistusfunktiota ei haluttu käyttää, sillä etäisyyksiin perustuva funktio ei olisi ollut tarkoitukseen sopiva. Jos rajoitusten rikkomiseen olisi käytetty samaa rangaistusfunktiota kuin valitessa liian läheisen potilaskohteen toisen agentin kanssa, olisi ollut mahdollista, että valittu potilaskohde olisi ollut etäisyydeltään optimaalinen. Tällöin rangaistus olisi jäänyt pieneksi, mikä taas ei olisi viestinyt agentille valinnan olleen huono. Lisäksi haluttiin antaa kohtalaisen suuri rangaistus, jotta agentti oppisi melko nopeasti, ettei rajoitusten rikkominen ole toivottu toiminto.

Palkkion lisäksi step-funktion lopussa palautetaan havainto ympäristön tilasta. Havaintotaulukkoon asetetaan arvoiksi lyhyin, pisin sekä keskiarvo matkustus- ja läpimenoajoista. Lisäksi havaintotaulukkoon lisätään arvo summa kaikista asiakaskäyntien kestosta, joka jaetaan summalla kaikista läpimenoajoista.

Kun mukautettu ympäristö on saatu rakennettua, voidaan aloittaa agentin harjoittaminen. Tässä pro gradu -tutkielmassa agenttien harjoittamiseen on käytetty Stable baselines 3 -

kirjaston valmiita vahvistetun oppimisen algoritmeja. Valmiita algoritmeja on mahdollista kutsua suoraan, sillä Stable baselines 3 tukee mukautettuja Gymnasium-ympäristöjä.

```
#Agentin harjoittaminen eri algoritmeilla
model = PPO('MlpPolicy', env, verbose=1)
model.learn(total_timesteps=100000)

model = DQN('MlpPolicy', env, verbose=1)
model.learn(total_timesteps=100000)

model = A2C('MlpPolicy', env, verbose=1)
model.learn(total_timesteps=100000)
```

Yllä olevassa esimerkissä on nähtävillä, kuinka agentin harjoittamista kutsutaan. Algoritmille annetaan parametreiksi MlpPolicy, mukautettu Gymnaisum-ympäristö sekä tieto siitä, miten paljon palautetta mallin luomisesta halutaan palautettavan. Verbose=1 arvolla tarkoitetaan, että ohjelmakoodi palauttaa info-tasoiset viestit konsoliin. Kun malli on muodostettu, voidaan sitä kutsua learn-funktiolla, joka harjoittaa mallia parametrissa olevan aikamääreen ajan. Jokaista mallia harjoitettiin 100 000 kertaa.

6.3.3 Lähestymistapa 2

Toisessa lähestymistavassa kotihoitajien reittioptimointiongelmaa lähdettiin ratkaisemaan hieman eri näkökulmasta. Kun ensimmäisessä lähestymistavassa havaintoavaruus perustui erilaisiin aikamääreisiin, nyt havaintoavaruutta lähdettiin rakentamaan agentin sijainnin ja vierailtujen potilaskohteiden avulla. Havaintoavaruus määritellään Box-tietorakenteeseen, johon asetetaan tiedot agentin nykyisestä sijainnista sekä totuusarvo, onko potilaskohteessa vierailtu. Havaintoavaruuden kooksi asetetaan summa agenttien ja potilaskohteiden määrästä. Toimintoavaruus määritellään Discrete-tyyppiseksi tietorakenteeksi, joka perustuu potilaskohteisiin. Käytännössä tässä lähestymistavassa toiminnoksi määritellään aina potilaskohde.

Kun ensimmäisessä lähestymistavassa step-funktio kävi jokaisen agentin silmukassa vuo-

rollaan läpi, tässä toteutuksessa hyödynnetään agentin valitsemiseen LIFO-periaatetta (engl. last in first out), jossa käsiteltävä agentti poistetaan listalta käsittelyn ajaksi ja potilaskohteen valinnan ja palkkion laskemisen jälkeen agentti palautetaan takaisin listan viimeiseksi jäseneksi. Jokaiselle listan agentille asetetaan kuitenkin ensimmäinen potilaskohde ennen kuin siirrytään toisen potilaskohteen valitsemiseen. Tällöin potilaskohteen valitseminen tapahtuu aina lista kerrallaan. Potilaskohteen valinta tapahtuu käytännön toteutuksessa Stable baselines 3 -kirjaston predict-funktion avulla, sillä se valitsee ympäristölle toiminnon.

Palkkiofunktio on toteutettu tässä lähestymistavassa perustumaan agentin ja potilaskohteen väliseen sijaintiin. Palkkiofunktio on eksponentiaalisesti vähenevä funktio, joka painottaa agentin ja potilaskohteen maantieteellistä läheisyyttä. Jos potilaskohteeksi on valittu kohde, joka on mahdollisimman lähellä agentin nykyistä sijaintia, annetaan suurempi palkkio. Jos taas potilaskohteeksi valikoituu hyvin kaukainen potilaskohde suhteessa agentin sen hetkiseen sijaintiin, palkkio on pienempi. Tällä pyritään säätelemään sitä, että agentit pysyisivät melko samalla alueella eivätkä hajaantuisi laajalle alueelle tarpeettomasti.

Palkkiofunktion laskeminen voidaan esittää Python-koodina, jossa a_lat ja a _lng edustavat agentin maantieteellisiä koordinaatteja ja c_lat ja c_lng taas edustavat potilaskohteen maantieteellisiä koordinaatteja:

```
def get_reward(self, a_lat, a_lng, c_lat, c_lng):
    distance = haversine((c_lat, c_lng), (a_lat, a_lng), unit='km')
    # Hyödynnetään eksponentiaalista vähenemistä
    return distance, (5.5 * np.exp(-1.25 * distance))
```

Palkkiofunktiossa potilaskohteen ja agentin välinen etäisyys lasketaan Haversine-algoritmilla. Mitä pienempi etäisyys agentilla ja potilaskohteella on, sitä suurempi palkkio eksponenttifunktiosta saadaan.

Tässä lähestymistavassa haluttiin hyödyntää luvussa 5.4 esitellyn tutkimuksen ajatusmallia sallittujen ja ei-sallittujen toimintojen luokittelusta. Tässä pro gradu -tutkielmassa kotihoitajien reitinoptimoinnissa potilaskohteisiin voi liittyä rajoitteita. Lisäksi potilaskohteissa halutaan vierailla aina vain yhden kerran. Tästä syystä haluttiin toteuttaa maski, jonka avulla pystyttiin luokittelemaan sallitut ja ei-sallitut toiminnot samaan tapaan kuin Tassel, Gebser ja

Schekotihin (2021) ovat luokitelleet tutkimuksessaan. Koska toimintoavaruus perustuu potilaskohteisiin, pystytään maskille antamaan tieto, onko kohteessa jo vierailtu ja kuuluuko siihen rajoitteita. Toiminnot luokiteltiin käymällä läpi potilaskohteet ja agentit, minkä jälkeen suodatettiin ei-sallitut kohteet pois.

Maski rakennettiin hyödyntämällä Stable baselines 3:n tarjoamaa valmista 'Maskable PPO'-luokkaa, jossa PPO-algoritmia hyödyntävän mallin toiminnot voidaan ajaa maskin läpi. Tällöin maski suodattaa ei-sallitut kohteet pois jo ennen kuin malli valitsee potilaskohteita. Tällä pystyttiin estämään, ettei agentti valitse ei-sallittuja potilaskohteita. Mallia harjoitettiin Python-koodilla:

```
model = MaskablePPO(MaskableActorCriticPolicy, env, verbose=1)
model.learn(total_timesteps=100000)
```

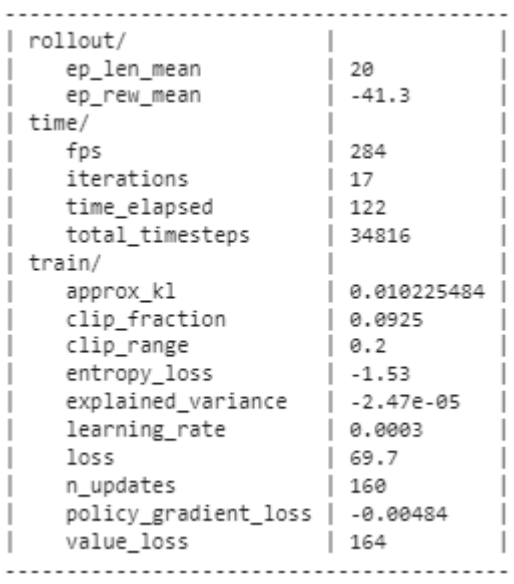
Mallille annettiin parametriksi Stable baselines 3:n 'MaskableActorCriticPolicy'-luokka, mukautettu Gymnasium-ympäristö ja verbose-arvo, jolla määritellään konsolin tulosteiden tarkkuus. Mallia harjoitettiin 100 000 kertaa.

6.4 Lopputulokset

Tässä alaluvussa käsitellään käyttötapauksen lopputuloksia. Aluksi käydään läpi ensimmäisen lähestymistavan lopputulokset, jossa agenttia harjoitettiin kolmella eri algoritmilla: PPO-, DQN- ja A2C-algoritmeilla. Tämän jälkeen käydään toisen lähestymistavan lopputulokset, joissa agenttia on harjoitettu maskitetulla PPO-algoritmilla.

6.4.1 Lähestymistapa 1

Ensimmäisessä lähestymistavassa mallin harjoittaminen aloitettiin luvusta 4.4 tutulla gradienttimetodeihin pohjautuvalla PPO-algoritmilla. Kun mallia lähdetään harjoittamaan PPO-algoritmilla, potilaskohteiden valitsemistavaksi valikoitui tapa, jossa valitaan asiakaskäynnin kestoltaan lyhyin potilaskohde. Kun mallia harjoitetaan, huomataan pian, että oppimistulokset eivät ole optimaalisia. Stable baselines 3 tulostaa info-tasoiset tiedot konsoliin, joten niistä nähdään jo, että algoritmi ei optimoi agenttien reitinvalintaa halutulla tavalla.

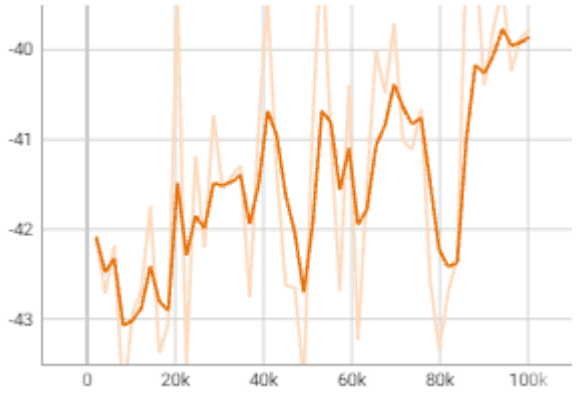


Kuvio 8. Esimerkki tulosteesta, joka saadaan PPO-algoritmin harjoittamisen yhteydessä

Kuviosta 8 huomataan, että PPO-algoritmilla harjoitetun mallin tulokset eivät ole optimaaliset. Erityisen olennaisia arvoja oppimisen kannalta ovat ep_rew_mean ja explained_variance. Ep_rew_mean kuvaa palkkion keskiarvoa ja optimaalisessa tuloksessa arvon tulisi olla nouseva harjoitusmäärien kasvaessa. Explained_variance taas viittaa arvofunktion palauttamaan varianssiin, joka optimaalisessa tuloksessa olisi mahdollisimman lähellä lukuarvoa 1. Jos arvo on lähellä nollaa tai sitä pienempi, voidaan arvoa pitää huonona.

Ep_rew_mean arvon kehitystä harjoitusmäärien suhteen on havainnollistettu kuviossa 9. Kuvio on luotu Tensorboard-integraation avulla, joka mahdollistaa oppimisdatan havainnollistamisen Stable baselines 3:n tuottaman datan perusteella. Y-akselilla esitetään palkkion arvo ja x-akselilla harjoituskertojen määrää. Kuviosta 9 nähdään, että palkkion keskiarvo on melko vaihteleva. Palkkion vähenemiset johtuvat todennäköisesti rangaistusfunktion käytöstä. Mallin harjoittamisessa huomattiin, että agentit eivät oppineet noudattamaan rajoitteita.

PPO-algoritmilla laskettujen agenttien sijoittumista on havainnollistettu kartalle kuviossa 10. Jokainen merkki kartalla edustaa potilaskohdetta, jossa agentti on käynyt. Jokaiselle agentille on määritelty oma väri, jolla agentit erotetaan toisistaan. Toisin sanoen esimerkiksi kaikki punaisella värillä merkityt potilaskohteet ovat saman agentin vierailemia potilaskohteita. Kuviosta nähdään, että reitinvalinta ei ole ollut optimaalinen, sillä kaikki agentit ovat jakau-

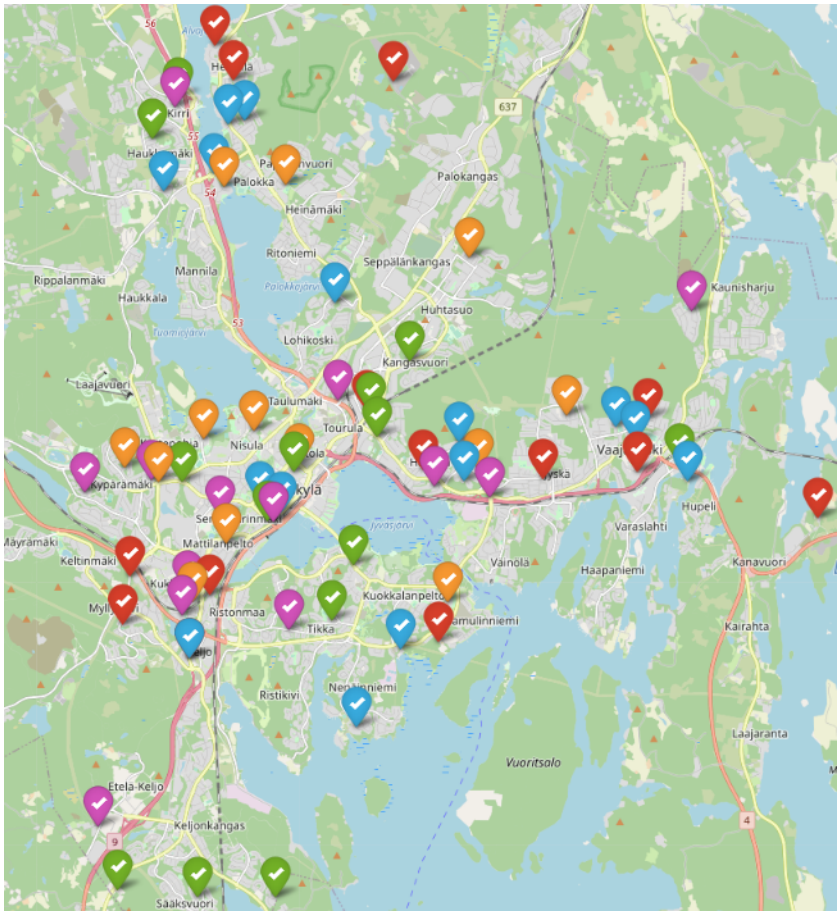


Kuvio 9. Ep_rew_mean arvon kehittyminen PPO-algoritmilla harjoitettaessa

tuneet laajalle alueelle, ja ne ovat valinneet toisia lähellä olevia potilaskohteita.

Toisena algoritmina mallin harjoittamiseen käytettiin luvussa 4.2 käsiteltyä DQN-algoritmia. DQN-algoritmia käytettäessä potilaskohteita valittiin kahdella eri tavalla. Potilaskohde valittiin joko maantieteellisesti lähimmän sijainnin perusteella tai potilaskohde valittiin satunnaisesti viidestä maantieteellisesti lähimmästä kohteesta. Mallin harjoittamisesta huomattiin jälleen nopeasti, ettei DQN-algoritmi myöskään optimoinut reitinvalintaa halutulla tavalla. Konsoliin tulostettavat tiedot osoittivat Explained_variance -arvon jäävän kauaksi optimaalisesta arvosta eikä palkkion keskiarvo ollut nouseva. Palkkion keskiarvoa on havainnollistettu kuviossa 11. Kuviosta nähdään selkeästi, että palkkion keskiarvossa on suuria vaihteluita. Myöskään DQN-algoritmin avulla ei onnistuttu luomaan mallia, joka oppisi noudattamaan rajoitteita. Rajoitteiden rikkominen laskee aina palkkion keskiarvoa.

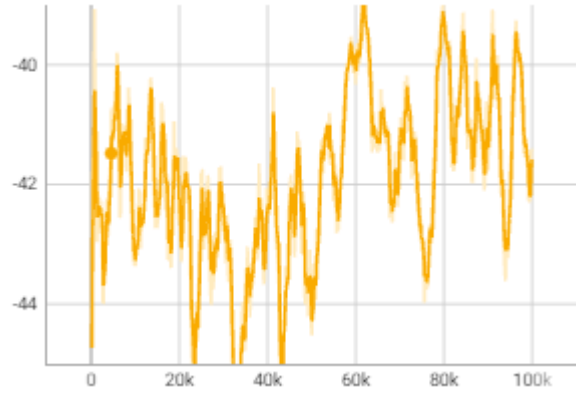
Viimeinen tässä lähestymistavassa käytetty algoritmi oli toimija-kriitikko -metodeihin kuuluva A2C. A2C-algoritmin kohdalla potilaskohteet valittiin siten, että kohteeksi valikoitui aina maantieteellisesti lähin potilaskohde. Myös A2C-algoritmia käytettäessä huomattiin nopeasti, että mallin harjoittaminen ei tuota toivottuja tuloksia. Mallin harjoittamisesta saatavasta tulosteesta kävi ilmi, että palkkion keskiarvo oli laskeva eikä explained_variance noussut kuin suurimmillaan nollaan. Palkkion keskiarvoa on havainnollistettu kuviossa 12, josta huomataan palkkion keskiarvon laskevan harjoituskertojen kasvaessa. Näiden tietojen perusteella voidaan päätellä, ettei A2C-algoritmi ollut sopiva lähestymistapa kotihoitajien reitinoptimointiin.



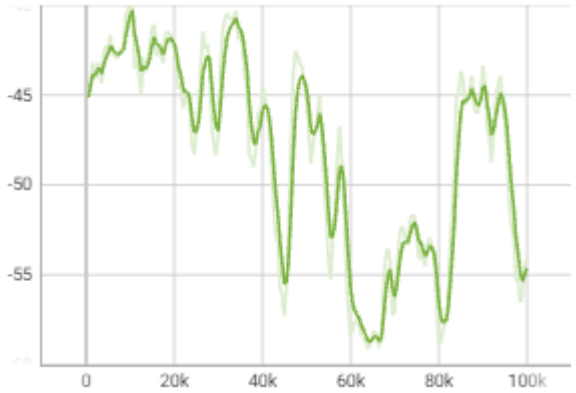
Kuvio 10. Potilaskohteet ja agentit sijoitettuna kartalle PPO-algoritmilla opetetussa mallissa

Lopputuloksena voidaan todeta, että ensimmäisessä lähestymistavassa käytetyt PPO-, DQN- ja A2C-algoritmit eivät optimoineet agenttien reitinvalintaa toivotulla tavalla. Jokaisella algoritmilla tulokset jäivät heikoiksi eikä agentit sijoittuneet toivotulla tavalla potilaskohteisiin. Potilaskohteen valintatavalla tai agentin harjoittamiseen käytetyillä aikamäärillä ei ollut merkittävää vaikutusta lopputulokseen. Myöskään PPO-, DQN- ja A2C-algoritmien välillä ei ollut suurta eroa lopputuloksen kannalta.

Syynä heikolle optimointitulokselle voi olla tapa, jolla havaintoavaruus on muodostettu. On mahdollista, että havaintojen välillä ei ole merkittävää eroa, jolloin ne ovat niin samankaltaiset, että optimaalisen toimintatavan oppiminen voi olla vaikeaa. Toinen mahdollinen syy heikolle optimointitulokselle voi olla sopimaton palkkiofunktio. On mahdollista, että palkkiofunktiossa ei annettu tarpeeksi suurta rangaistusta ei-sallituista toiminnoista, jolloin malli ei pystynyt oppimaan sallittuja ja ei-sallittuja valintoja.



Kuvio 11. Ep_rew_mean arvon kehittyminen DQN-algoritmilla harjoitettaessa



Kuvio 12. Ep_rew_mean arvon kehittyminen A2C-algoritmilla harjoitettaessa

6.4.2 Lähestymistapa 2

Toisessa lähestymistavassa mallia harjoitettiin maskitetulla PPO-algoritmilla. Maskin avulla saatiin suodatettua ei-sallitut potilaskohteet pois potilaskohteen valinnasta. Kun mallia lähdettiin harjoittamaan, Stable baselines 3:n tulostamasta info-tiedoista huomataan pian, että oppimistulokset ovat parempia kuin aiemmassa lähestymistavassa.

Tulostetta on havainnollistettu kuviossa 13. Tulosteesta tarkastellaan samoja arvoja kuin ensimmäisessä lähestymistavassa, eli oleellisia tietoja ovat ep_rew_mean ja explained_variance.

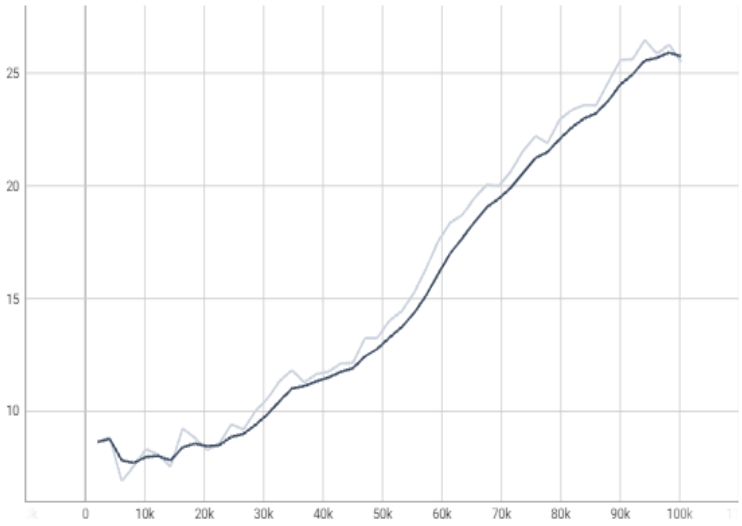
```
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 102         |
|    ep_rew_mean          | 25.5        |
| time/                   |             |
|    fps                  | 127         |
|    iterations           | 49          |
|    time_elapsed         | 786         |
|    total_timesteps      | 100352      |
| train/                  |             |
|    approx_kl            | 0.015259212 |
|    clip_fraction        | 0.166       |
|    clip_range           | 0.2         |
|    entropy_loss         | -2.75       |
|    explained_variance   | 0.856       |
|    learning_rate        | 0.0003      |
|    loss                 | 0.84        |
|    n_updates            | 480         |
|    policy_gradient_loss | -0.043      |
|    value_loss           | 2.1         |
-----------------------------------------
```

Kuvio 13. Esimerkki tulosteesta, joka saadaan maskitetun PPO-algoritmin harjoittamisesta.

Kuten kuviosta 13 nähdään, explained_variance arvo on lähellä arvoa 1, kuten optimaalisissa tuloksissa odotetaan. Lisäksi palkkion keskiarvoa kuvaava ep_rew_mean -arvo kasvaa harjoitusmäärien kasvaessa. Tätä on havainnollistettu kuviossa 14. Kuviosta nähdään selvästi, että palkkion keskiarvon kehittyminen on nousujohteista harjoitusmäärien kasvaessa.

Kuviossa 15 on havainnollistettu agenttien sijoittumista kartalle samaan tapaan kuin ensimmäisessä lähestymistavassa. Kuviosta nähdään, että agentit ovat sijoittuneet melko laajalle alueelle eikä silmämääräisesti voida nähdä, onko reitinvalinta parempi kuin ensimmäisessä lähestymistavassa.

Tässä lähestymistavassa haluttiin havainnollistaa reitinvalintaa vielä tarkemmin yhden agen-

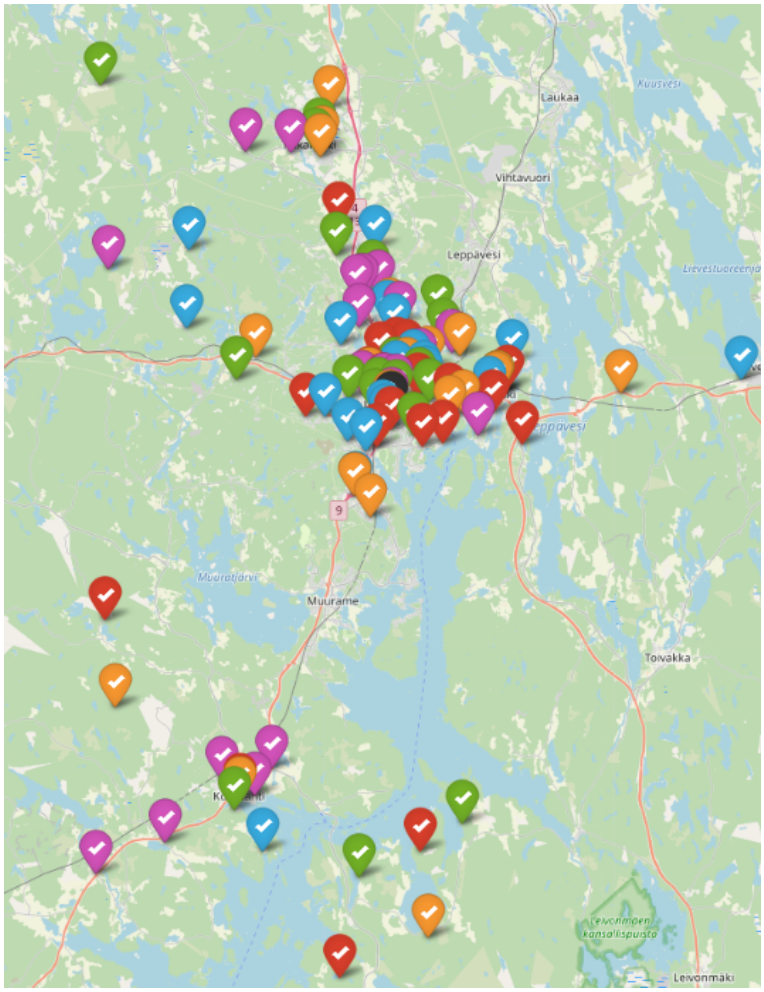


Kuvio 14. Palkkion keskiarvon kehitys mallia harjoitettaessa

tin näkökulmasta. Yhden agentin kulkema reitti on nähtävillä kuviossa 16. Reittiä on pyritty havainnollistamaan lisäämällä katkoviivat kuvaamaan agentin kulkemaa reittiä eri potilaskohteiden välillä. Kuviosta 16 voidaan huomata, että agentin potilaskohteet ovat jakautuneet melko laajalle alueelle ja osa potilaskohteiden välisistä etäisyyksistä on suuria.

Yhden agentin kulkemaa reittiä on havainnollistettu vielä tarkemmin kuviossa 17. Kuviossa agentin reitti on keskitetty lähelle mustalla ikonilla merkittyä agentin lähtöpaikkaa. Jokaiselle agentille on määritelty sama lähtöpaikka ja jokainen agentti lähtee liikkeelle aamulla kello 8.00. Kuviossa on havainnollistettu myös jokaiseen potilaskohteeseen liittyviä tietoja. Ensimmäisenä tietona on kohdenumero. Tällä tarkoitetaan järjestystä, jossa agentti kulkee potilaskohteita. Tämän lisäksi kartalla on havainnollistettu teksti ‘Valmis,’ jolla tarkoitetaan kellonaikaa, jolloin agentti on saanut potilaskohteen toimenpiteet suoritettua. Tähän kellonaikaan sisältyy agentin matkustusaika potilaskohteeseen ja potilaskäynnille määritelty kesto. Kuviossa 17 havainnollistetun agentin kulkuvälineenä oli auto, joka liikkui 50km/h nopeudella. Agentin matkustusaika laskettiin Haversine-algoritmin palauttaman etäisyyden ja agentin nopeuden perusteella.

Agentin reittiin liittyviä aikatietoja on havainnollistettu tarkemmin kuviossa 18. Kuviossa on esitetty taulukkona jokaiseen potilaskohteeseen liittyvät matkustusajat, potilaskäyntien kestot sekä kellonaika, jolloin agentti on ollut valmis suoritettuaan potilaskäynnin toimenpiteet.



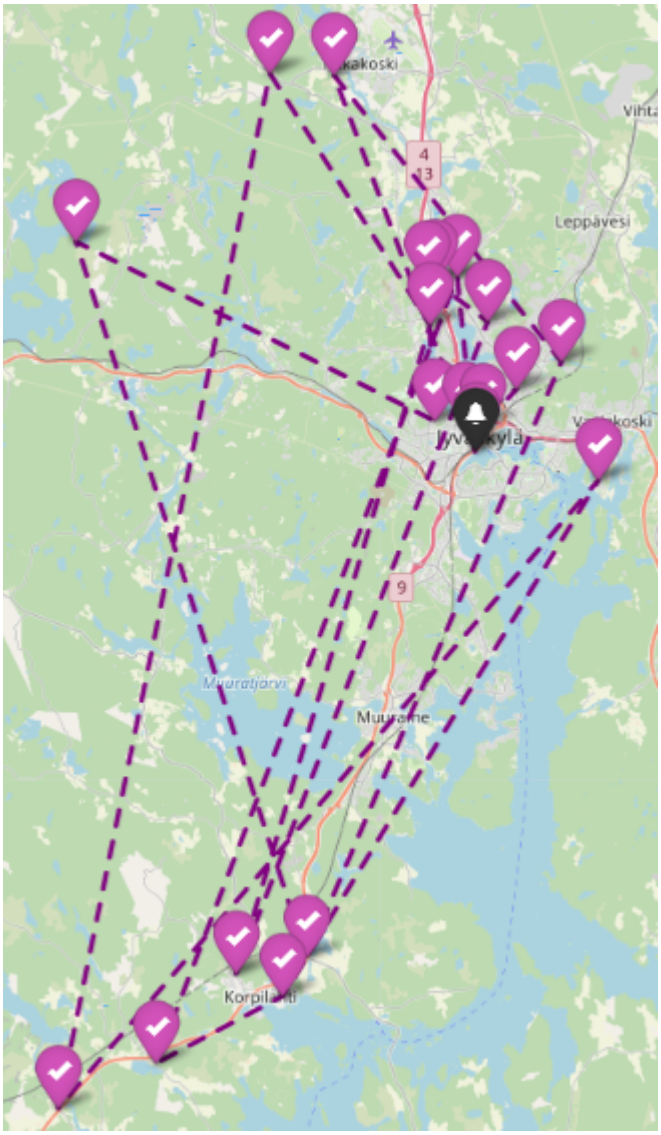
Kuvio 15. Agentit sijoitettuna kartalle maskitetussa PPO-algoritmissa

Taulukkoon on merkitty 0. kohde, jolla tarkoitetaan agentin lähtöpaikkaa. Taulukosta voidaan huomata, että matkustusajat vaihtelevat melko paljon. Osa potilaskohteista on lähellä toisiaan, jolloin matkustusaikakin on lyhyt. Osa potilaskohteista sijaitsi saman kadun varrella, jolloin matkustusaika oli jopa alle minuutin. Matkustusajoista nähdään myös, että osa potilaskohteista oli melko kaukana, jolloin liikkumiseen kului paljon aikaa.

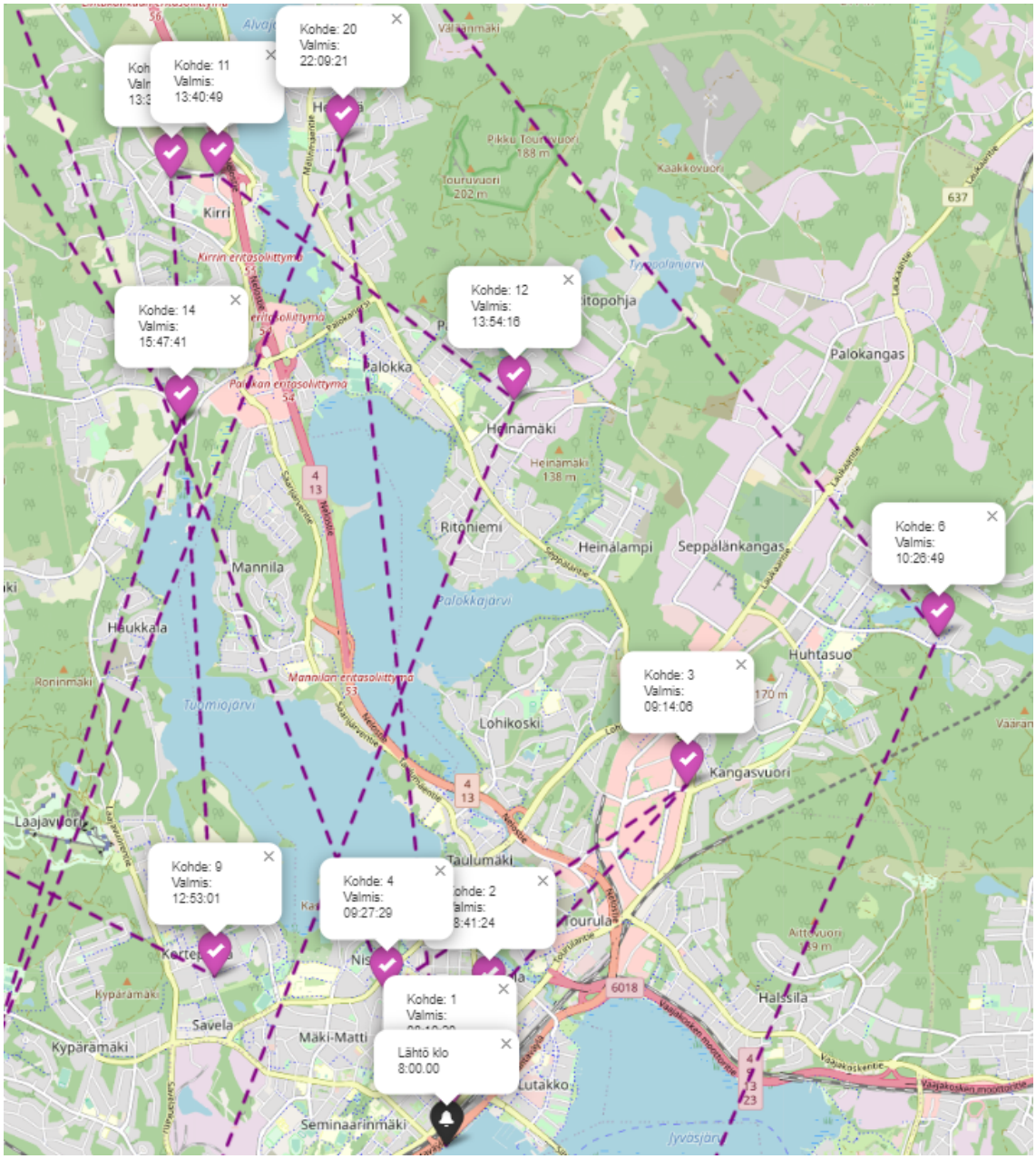
Yhdestäkään edellä mainituista kuvioista ei voida selkeästi nähdä, että lopputulos olisi parempi kuin aiemmassa lähestymistavassa. Oppimisdatan perusteella voidaan kuitenkin todeta mallin pystyvän ratkaisevan ongelman melko hyvin lopputuloksin. Oppimisdatan explained_variance -arvo on lähellä lukuarvoa 1, jota voidaan pitää optimaalisena tuloksena. Lisäksi ep_rew_mean -arvo kasvaa selkeästi harjoitusmäärien kasvaessa, mistä voidaan pää-

tellä mallin oppivan. Erona aiempaan lähestymistapaan, tässä lähestymistavassa pystyttiin myös estämään maskin avulla ei-sallittujen toimintojen valitseminen. Näiden tietojen perusteella lopputulosta voidaan pitää parempana kuin aiemmassa lähestymistavassa.

Ratkaisevana tekijänä parempiin lopputuloksiin on mahdollisesti erilainen tapa muodostaa havaintoavaruutta. Nyt havaintovektoreiden arvot poikkeavat selkeämmin toisistaan, sillä ne perustuvat potilaskohteen dataan aikamääreen sijaan. Toinen vaihtoehto paremmalle lopputulokselle voi olla erilainen palkkiofunktio ja maskin käyttö, minkä vuoksi ei-sallittuja kohteita ei valikoitunut agentin reitille.



Kuvio 16. Yhden agentin kulkema reitti



Kuvio 17. Tarkennettu kuva agentin kulkemasta reitistä

Kohde	Matkustusaika (min)	Potilaskäynnin kesto (min)	Kellonaika
0	0	0	8:00:00
1	0.34	10	08:10:20
2	1.06	30	08:41:24
3	2.71	30	09:14:06
4	3.37	10	09:27:29
5	19.85	10	09:57:20
6	19.49	10	10:26:49
7	34.60	20	11:21:25
8	40.21	10	12:11:38
9	21.38	20	12:53:01
10	7.38	30	13:30:24
11	0.43	10	13:40:49
12	3.45	10	13:54:16
13	37.29	20	14:51:34
14	36.13	20	15:47:41
15	15.77	30	16:33:27
16	56.39	30	17:59:51
17	44.40	30	19:14:15
18	32.41	30	20:16:39
19	7.66	30	20:54:19
20	45.04	30	22:09:21

Kuvio 18. Agentin reitin aikatiedot

7 Johtopäätökset

Tässä pro gradu -tutkielmassa tutkittiin vahvistettua oppimista kotihoitajien reittioptimoinnissa. Tutkielman tarkoituksena oli selvittää, voiko vahvistetun oppimisen algoritmeilla optimoida kotihoitajien reitinkulkua. Tutkielman alussa perehdyttiin kirjallisuuskatsauksen muodossa koneoppimiseen ja syväoppimiseen. Näillä aiheilla pohjustettiin tämän tutkielman kannalta oleellisinta koneoppimisen suuntausta, vahvistettua oppimista. Tutkielmassa perehdyttiin vahvistetun oppimisen perusperiaatteisiin ja sen taustalla vaikuttavaan Markovin päätöksentekoprosessiin. Lisäksi tutkielmassa esiteltiin neljä tunnettua vahvistetun oppimisen algoritmia: Q-oppiminen, syvä Q-verkko, A2C ja PPO.

Vahvistetun oppimisen teorian jälkeen keskityttiin tutkimuskysymyksen kannalta oleelliseen teoreettiseen viitekehykseen. Teoreettisessa viitekehyksessä esiteltiin, miten kotihoitajien reitinkulkua voidaan mallintaa kirjallisuudesta tutun töidenjärjestelyongelman avulla ja Markovin päätöksentekoprosessina. Teoreettisen viitekehyksen lopulla käytiin läpi tieteellistä julkaisua, josta saatiin ajatusmalli käytännön toteutuksen tueksi.

Kirjallisuuskatsauksen jälkeen keskityttiin tässä pro gradu -tutkielmassa tehtyyn tapaustutkimukseen. Tapaustutkimuksessa rakennettiin Gymnaisum-ympäristö, johon mallinnettiin kotihoitajien reitinoptimointiympäristö. Kotihoitajien reitinkulkua harjoitettiin vahvistetun oppimisen algoritmeilla ja tutkittiin optimoivatko algoritmit reittiä. Tapaustutkimusta lähestyttiin kahdesta eri näkökulmasta, joissa havaintoavaruudet ja palkkiofunktio muodostettiin eri lailla. Ensimmäisessä lähestymistavassa havaintoavaruus ja palkkiofunktio perustuivat agentin suorittamien potilaskäynteihin liittyviin aikamääreisiin, kun taas toisessa lähestymistavassa havaintoavaruus perustui agentin sijaintiin ja potilaskohteen vierailutietoon. Jälkimmäisessä lähestymistavassa palkkiofunktio perustui agentin ja potilaskohteen väliseen sijaintiin. Mitä lähempi potilaskohde valittiin omaan sijaintiin nähden, sitä suurempi palkkio saatiin. Lopputulokseksi saatiin, että jälkimmäinen malleista pystyi oppimaan kotihoitajien reitinoptimoinnin. Reittioptimoinnissa käytettiin maskitettua PPO-algoritmia. Tästä voidaan todeta, että vahvistetun oppimisen algoritmeilla on mahdollista optimoida kotihoitajien reitinoptimointia.

Koska vahvistettu oppiminen on suosittu tutkimusaihe ja töidenjärjestelyongelmien näkökulmasta vielä melko uusi tutkimusaihe, jatkotutkimusmahdollisuuksia aiheelle olisi. Kotihoidon toimialalla voitaisiin esimerkiksi tutkia mahdollisuuksia havainnoida potilaiden terveydentilaa koneoppimisen menetelmien avulla. Voitaisiin esimerkiksi tutkia, onko potilasdatasta mahdollista ennustaa terveydentilan muutoksia. Toinen jatkotutkimuksen aihe voisi taas liittyä kotihoidon asiakkaiden yksinäisyyden lievittämiseen. Koneoppimisen ja robotiikan avulla voitiaisiin tutkia, voiko robotin kanssa keskustelu lievittää yksinäisyyttä. Robottia voisi opettaa keskustelemaan erilaisten koneoppimisen menetelmien avulla. Lisäksi nykyisessä yhteiskunnallisessa tilanteessa, jossa työntekijöistä on pulaa useilla eri aloilla, töidenjärjestelyongelmat sekä vahvistetun oppimisen sovellukset voisivat tuoda kaivattuja lisäresursseja eri toimialoille. Toisaalta yhteiskunnan toimivuuden kannalta kriittisillä aloilla, kuten terveydenhuollon alalla, tekoälyn hyödyntäminen tuo aina esiin myös eettisiä kysymyksiä, esimerkiksi koneiden luotettavuuden suhteen tai sosiaalisen kontaktin puuttumisen vuoksi.

Lähteet

Aggarwal, Charu C. 2018. *Neural Networks and Deep Learning A Textbook.* 497. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-94463-0.

Alpaydin, Ethem. 2016. *Machine Learning : The New AI.* United States of America: MIT Press.

Brucker, Peter. 2012. *Complex Scheduling.* 342. GOR-Publications. Berlin, Heidelberg: Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-23929-8.

Buşoniu, Lucian, Robert Babuška ja Bart De Schutter. 2010. "Multi-agent reinforcement learning: An overview". Teoksessa *Innovations in multi-agent systems and applications-1,* 183–221. Springer.

Chapelle, Olivier, Bernhard Schölkopf ja Alexander Zien. 2006. *Semi-Supervised Learning.* Toimittanut Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, Alexander Zien, Bernhard Schölkopf ja Bernhard Schölkopf. MIT Press. https://sfx.finna.fi/nelli09?url_ver=Z39.88-2004&ctx_ver=Z39.88-2004&ctx_enc=info:ofi/enc:UTF-8&rfr_id=info:sid/sfxit.com:opac_856&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx&sfx.ignore_date_threshold=1&rft.object_id=1000000000466006&svc_val_fmt=info:ofi/fmt:kev:mtx:sch_svc&.

François-Lavet, Vincent, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau ym. 2018. "An introduction to deep reinforcement learning". *Foundations and Trends® in Machine Learning* 11 (3-4): 219–354.

Gabel, Thomas, ja Martin Riedmiller. 2008. "Adaptive reactive job-shop scheduling with reinforcement learning agents". *International Journal of Information Technology and Intelligent Computing* 24 (4): 14–18.

Gymnasium. https://gymnasium.farama.org/.

Hervé Abdi, Dominique Valentin, ja Betty Edelman. 1999. *Neural Networks.* London: SAGE Publications.

Ian Goodfellow, Aaron Courville, Yoshua Bengio. 2016. *Deep learning.* MIT Press.

Ibe, Oliver C. 1947. 2013. *Markov processes for stochastic modeling.* Second edition. 494. Elsevier insights. London: Elsevier. http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=516132.

Komm, Dennis. 2016. *An Introduction to Online Computation : Determinism, Randomization, Advice.* 349. Texts in Theoretical Computer Science. An EATCS Series. Cham: Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-42749-2.

Kulkarni, Parag. 2012. *Reinforcement and systemic machine learning for decision making.* Canada: Hoboken New Jersey : John Wiley / Sons.

Lapan, Maxim. 2018. *Deep reinforcement learning hands-on : apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more.* Packt Publishing. https://sfx.finna.fi/nelli09?url_ver=Z39.88-2004&ctx_ver=Z39.88-2004&ctx_enc=info:ofi/enc:UTF-8&rfr_id=info:sid/sfxit.com:opac_856&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx&sfx.ignore_date_threshold=1&rft.object_id=4100000004836426&svc_val_fmt=info:ofi/fmt:kev:mtx:sch_svc&.

Libbrecht, Maxwell W, ja William Stafford Noble. 2015. “Machine learning applications in genetics and genomics”. *Nature Reviews Genetics* 16 (6): 321–332.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver ja Koray Kavukcuoglu. 2016. “Asynchronous Methods for Deep Reinforcement Learning”. *CoRR* abs/1602.01783. arXiv: 1602.01783. http://arxiv.org/abs/1602.01783.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra ja Martin A. Riedmiller. 2013. “Playing Atari with Deep Reinforcement Learning”. *CoRR* abs/1312.5602. arXiv: 1312.5602. http://arxiv.org/abs/1312.5602.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski ym. 2015. “Human-level control through deep reinforcement learning”. *nature.*

Mohammed, Mohssen Khan, Muhammad Badruddin Bashier ja Eihab Bashier Mohammed. 2017. *Machine Learning : Algorithms and Applications.* Boca Raton : CRC Press.

Nazari, Mohammadreza, Afshin Oroojlooy, Lawrence Snyder ja Martin Takác. 2018. “Reinforcement learning for solving the vehicle routing problem”. Teoksessa *Advances in Neural Information Processing Systems,* 9839–9849.

Otterlo, Martijn van. toimittaja, toimittanut. 2012. *Reinforcement Learning : State-of-the-Art.* 638. Adaptation, Learning, and Optimization. Berlin, Heidelberg: Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-27645-3.

Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus ja Noah Dormann. 2021. “Stable-baselines3: Reliable reinforcement learning implementations”. *The Journal of Machine Learning Research* 22 (1): 12348–12355.

Rummery, Gavin A, ja Mahesan Niranjan. 1994. *On-line Q-learning using connectionist systems.* Nide 37. University of Cambridge, Department of Engineering Cambridge, UK.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford ja Oleg Klimov. 2017. “Proximal Policy Optimization Algorithms”. *CoRR* abs/1707.06347. arXiv: 1707.06347. http://arxiv.org/abs/1707.06347.

Schwartz, H. M. 2014. *Multi-Agent Machine Learning: A Reinforcement Approach.* John Wiley / Sons, Incorporated.

Sheikh, Aziz. 2018. *Key Advances in Clinical Informatics : Transforming Health Care through Health Information Technology.* Toimittanut David W. Bates, Kathrin Cresswell ja Adam Wright. Saint Louis: Academic Press. https://sfx.finna.fi/nelli09?url_ver=Z39.88-2004&ctx_ver=Z39.88-2004&ctx_enc=info:ofi/enc:UTF-8&rfr_id=info:sid/sfxit.com:opac_856&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx&sfx.ignore_date_threshold=1&rft.object_id=3710000001417593&svc_val_fmt=info:ofi/fmt:kev:mtx:sch_svc&.

Shi, Zhongzhi. 2011. *Advanced artificial intelligence.* Toimittanut Zhongzhi Shi. Series on intelligence science. Hackensack, N.J.: World Scientific. https://ebookcentral.proquest.com/lib/jyvaskyla-ebooks/detail.action?docID=840558.

Skansi, Sandro. 2018. *Introduction to Deep Learning: from logical calculus to artificial intelligence.* Springer.

Sutton, Richard S, ja Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

Tassel, Pierre, Martin Gebser ja Konstantin Schekotihin. 2021. *A Reinforcement Learning Environment For Job-Shop Scheduling.* arXiv: 2104.03760 [cs.LG].

Toth, Paolo, ja Daniele Vigo. 2002. *The vehicle routing problem.* SIAM.

Van Engelen, Jesper E, ja Holger H Hoos. 2020. "A survey on semi-supervised learning". *Machine learning* 109 (2): 373–440.

Watkins, Christopher JCH, ja Peter Dayan. 1992. "Q-learning". *Machine learning* 8:279–292.

Zhu, Jie, Fengge Wu ja Junsuo Zhao. 2021. "An Overview of the Action Space for Deep Reinforcement Learning". Teoksessa *2021 4th International Conference on Algorithms, Computing and Artificial Intelligence,* 1–10.