Tampereen yliopisto

Patrik Tohni

RUBIKIN KUUTION RATKAISEMINEN VAHVISTAVAN KONEOPPIMISEN AVULLA

Teknis-luonnontieteellinen tiedekunta
Kandidaatintyö
Maaliskuu 2019

TIIVISTELMÄ

Patrik Tohni: Rubikin kuution ratkaiseminen vahvistavan koneoppimisen avulla
Kandidaatintyö
Tampereen yliopisto
Teknis-luonnontieteellinen tutkinto-ohjelma
Maaliskuu 2019

Tässä työssä tutkittiin vahvistavalla oppimisella koulutetun tiheän neuroverkon kykyä oppia ratkaisemaan Rubikin kuutio -pulmalelu. Kuutiolle rakennettiin matemaattinen malli, jonka perusteella koodattiin kuutiota simuloiva ympäristö koneoppiagentin käytettäväksi. Työssä selvitettiin agentin pohjana olevan deep Q-learning -neuroverkkotekniikan toimintaperiaatteita ja parametrejä. Lisäksi tutkittiin muita oppimiseen vaikuttavia tekniikoita: one hot encoding, replay memory ja monimutkaisuuden vaiheittainen lisääminen. Toteutettu agentti saavutti tavoitteensa, joka oli kolmella liikkeellä sekoitetun kuution ratkaiseminen.

Avainsanat: Rubikin kuutio, vahvistava oppiminen, deep Q-network, Q-Learning

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

a_i^+	kuution i:nnen tahkon 90° käännös myötäpäivään
a_i^-	kuution i:nnen tahkon 90° käännös vastapäivään
α	käännösjono
a_i	käännösjonon i:s käännös
A	käännösjoukko
$b_{i,j}$	neuroverkon i:nnen kerroksen j:nnen solmun vakio
β	oppimiskerroin
f_a	aktivaatiofunktio
f_v	virhefunktio
γ	pisteiden periytymisen vähennyskerroin
$Q(s, a)$	tilassa s tehdyn käännöksen a palauttama kokonaispalkkio
R	agentille käännöksestä annettava tulevaisuuden palkkio
r	agentille käännöksestä annettava välitön palkkio
s	kuution tilan esittävä jono
S	kuution tilat käsittävä tilajoukko
$\mathbf{t_e}$	neuroverkon tuloskerrokselta saatava ennustevektori
$\mathbf{t_h}$	neuroverkon tuloskerrokselta haluttu kontrollivektori
$w_{i,j,k}$	neuroverkon i:nnen kerroksen j:nnen solmun k:nnen liitoksen painokerroin
$x_{i,j}$	neuroverkon i:nnen kerroksen j:nnen solmun arvo
Adam	Adaptive Moment Estimation -optimointialgoritmi
agentti	koneoppimisella tuotettu toimija
DeepCube	McAleer et al. tuottama Rubikin kuution ratkaiseva agentti
DQN	DeepMindin kehittämä deep Q-network -koneoppiagentti
MSE	Mean Squared Error
ReLU	Rectified Linear Unit -aktivaatiofunktio
[120, 50, 20]	neuroverkon piilotettujen kerrosten solmujen määrä

1 JOHDANTO

Rubikin kuutio on erittäin tunnettu pulmalelu. Sen on kehittänyt unkarilainen arkkitehtuurin professori Ernő Rubik vuonna 1974 auttaakseen oppilaitaan ymmärtämään kolmiulotteisia ongelmia. Rubik nimitti keksintöään aluksi taikakuutioksi, mutta lelun muodostuessa hittituotteeksi se uudelleenbrändättiin Rubikin kuutioksi. Seitsemän vuotta kuution keksimisen jälkeen sitä myytiin jo miljoonia kappaleita ympäri maailman. Alkuperäinen kuutio on $3 \times 3 \times 3$ kokoinen, mutta siitä on myöhemmin valmistettu ja myyty myös pienempiä ja suurempia kokoja. [18]

Rubikin kuutiota on jo pitkään ratkaistu erilaisin ohjelmallisin keinoin. Toimivimmiksi todetut ratkaisualgoritmit käyvät kuution tiloja järjestelmällisesti läpi tai hyödyntävät kuution ryhmäteoreettisia ominaisuuksia. Tämän kaltaisten ongelmien ratkaiseminen myös koneoppimisen tekniikoin on jo pitkään kiinnostanut alan asiantuntijoita. McAleer et al. onnistuivat toukokuussa 2018 ensimmäisen kerran luomaan täysin koneoppimiseen pohjautuvan Rubikin kuution ratkaisijan, joka kykenee ratkaisemaan mielivaltaisen monta kertaa sekoitettuja kuutioita 100 % onnistumisvarmuudella. Aiemmat vastaavat yritykset ovat saaneet ratkaistua vain noin viisi kertaa sekoitettuja kuutioita, minkä jälkeen niiden ratkaisuprosentti on laskenut huomattavasti [5]. [6]

Tässä työssä pyrittiin vahvistavan koneoppimisen keinoilla toteuttamaan itseoppiva Rubikin kuution ratkaisija. Työn tavoitelaajuuden ja kirjoittajan aiemman alan osaamisen perusteella ratkaisukyvylle asetettiin maltillinen muutaman sekoituksen tavoite. Työn päätavoite olikin tutkia eri parametrien vaikutusta oppimisen tehokkuuteen ja tuloksiin keskittyen deep Q-network -neuroverkon kaltaisen tekniikan käyttöön [8]. Luvussa 2 määritellään matemaattisesti Rubikin kuutio ja siihen liittyvät työssä käytettävät termit. Kolmannessa luvussa esitellään kuution ratkaisemista eri keinoin ja neljännessä tarkastellaan työssä käytettävää neuroverkkoa sekä sen toimintaan vaikuttavia parametreja. Lopuksi viidennessä luvussa esitetään eri parametrien vaikutukset oppimisprosessiin.

2 MATEMAATTINEN MALLI

Rubikin kuutio rakentuu $3 \times 3 \times 3$ pienemmästä kuutiosta eli periaatteessa 27 palasta, mutta kuution sisällä oleva keskimmäinen pala ei koskaan näy, joten kuutiossa on 26 merkitsevää palaa. Palat koostuvat 6 keskipalasta, 12 reunapalasta ja 8 nurkkapalasta. Keskipalojen paikat eivät ikinä muutu suhteessa toisiinsa ja niissä on vain yksi näkyvä sivu. Reunapaloissa on aina kahta eri väriä ja nurkkapaloissa kolmea.

Kuva 2.1. *Perinteinen* $3 \times 3 \times 3$ *Rubikin kuutio, jossa kolme ratkaistua tahkoa [18].*

Kuution *tila* s voidaan esittää kiinnittimällä kuution keskipalat tiettyyn asentoon ja luettelemalla muiden ruutujen värit sovitussa järjestyksessä 48-alkioisena jonona. Jonon jokainen alkio ilmaisee kyseistä indeksiä vastaavan ruudun värin kokonaisluvuksi koodattuna:

$$s = (v_1, v_2, v_3, \ldots, v_{48}), \quad v_i \in [1, 6].$$

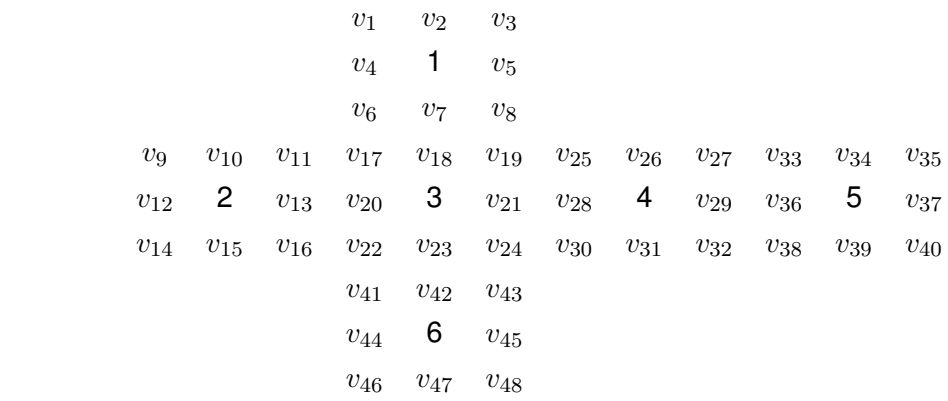
Jokaiselle kuudelle värille c joukko $\{v_i \mid v_i = c\}$ sisältää täsmälleen kahdeksan alkiota. Ruudut v_i on kiinnitetty kuvan 2.2 osoittamalla tavalla. Kaikki tilat s muodostavat kokonaisuudessaan tilojen joukon $S = \{s\}$.

Tilasta toiseen voidaan siirtyä alkeistoimintojen

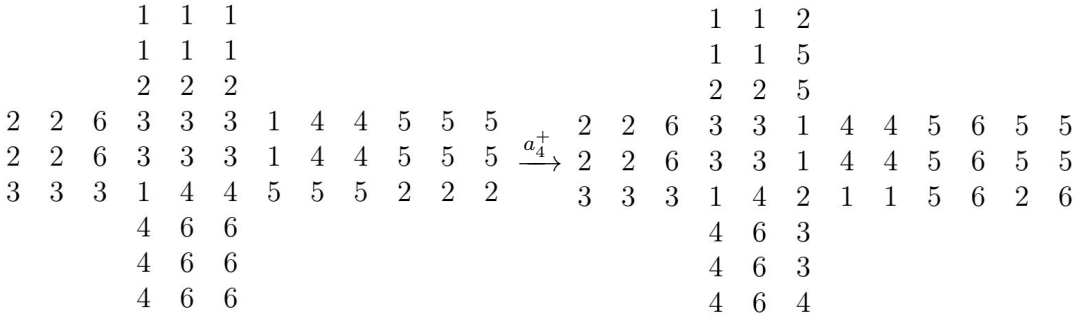
$$A^+ = \left\{a_i^+ \mid i \in [1, 6]\right\}$$

avulla. Jokainen toiminto a_i^+ kuvaa indeksiä i vastaavan kuution tahkon t_i kääntämistä myötäpäivään 90 astetta. Kuvassa 2.3 esitellään esimerkkinä toiminnon a_4^+ vaikutus kuution tilaan. Alkeistoimintojen joukkoa A^+ laajennetaan rinnastamalla kolmen saman liikkeen sarja yhdeksi vastapäivään tehtäväksi neljäsosakierroksen käännökseksi

$$a_i^- = a_i^+ \cdot a_i^+ \cdot a_i^+.$$



Kuva 2.2. *Kuution ruutujen indeksit ja kiinnitettyjen keskipalojen värit.*



Kuva 2.3. *Esimerkki tahkon t_4 myötäpäivään kääntämisestä toiminnolla a_4^+.*

Tämä on perusteltua, koska liikkeiden kokonaisvaikutukset kuutioon ovat samat. Merkitään näitä vastapäivään tehtäviä liikkeitä joukkona

$$A^- = \left\{ a_i^- \mid i \in [1,6] \right\}$$

ja kirjoitetaan laajennettu toimintojoukko

$$A = A^+ \cup A^-.$$

Joukkoon A kuuluvia toimintoja a kutsutaan tästä eteenpäin *käännöksiksi*. Käännökset ovat siis eräänlaisia funktioita

$$a \ : \ S \to S,$$

jotka muuttavat kuution tilaa. Käännökset voidaan ilmaista yhtälössä kirjoittamalla

$$s_{i+1} = s_i \cdot a,$$

missä funktion a argumentti luetaan aina merkin $\cdot$ vasemmalta puolelta. Käännöksiä voidaan ketjuttaa seuraavasti:

$$s_{i+n} = s_i \cdot a_1 \cdot a_2 \cdot \ldots \cdot a_n.$$

Ketjutetut käännökset voidaan kirjoittaa *käännösjonona*

$$\alpha = (a_1, a_2, \ldots, a_n) = a_1 \cdot a_2 \cdot \ldots \cdot a_n,$$

jolla voidaan operoida kuution tilaa samalla tavalla kuin yksittäisellä käännöksellä:

$$s_{i+1} = s_i \cdot \alpha.$$

Käännökset a ovat bijektiivisia, joten vastakkaissuuntaisilla käännöksillä peräkkäin operoitaessa kuutio päätyy takaisin alkuperäiseen tilaansa:

$$(s \cdot a_i^+) \cdot a_i^- = s.$$

Käännökset a_i^+ ja a_i^- ovat siis toistensa käänteisoperaatioita.

Tilat s voidaan määritellä myös käännösten a avulla, mikäli ensin määritellään täsmällisesti kuution *ratkaistu tila* s_0. Ratkaistussa tilassa kuution kaikilla sivuilla on vain kyseisen sivun keskipalan värisiä ruutuja:

$$s_0 = (1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, \ldots, 6, 6).$$

Ratkaistulle tilalle käännöksiä suorittamalla voidaan saavuttaa noin $4,3 \cdot 10^{19}$ erilaista tilaa s [4]. Tällä tavoin saavutettavien tilojen joukko S' on kuitenkin vain koko tilajoukon S osajoukko, eli $S' \subset S$. Joukko S' voidaan määritellä käännösten avulla:

$$S' = \{s \mid \exists\, \alpha : s = s_0 \cdot \alpha\}.$$

Kaikkien tässä työssä käsiteltävien kuutioiden tilat kuuluvat joukkoon S'. Kuutiota ratkaistaessa pyritään löytämään käännösjono, jolla operoimalla kuutio saadaan siirrettyä sekoitetusta tilasta ratkaistuun tilaan: $s \xrightarrow{\alpha} s_0$. Käännösjonoa α, jossa on mahdollisimman vähän alkioita a kutsutaan tässä työssä *optimaaliseksi* tai *lyhyimmäksi ratkaisuksi*.

Määriteltyjen ominaisuuksien pohjalta voidaan muodostaa seuraava Rubikin kuutiolle pätevä lause.

Lause 2.1. *Jos ja vain jos* $s_1 \cdot \alpha = s_3 \ \wedge \ s_2 \cdot \alpha = s_3$ *niin* $s_1 = s_2$.

Eli, jos tietty käännösjono α siirtää kuution tilasta s_1 tilaan s_3 ja sama käännösjono muuttaa tilan s_2 tilaksi s_3, niin tilojen s_1 ja s_2 täytyy olla täsmälleen samat.

On syytä huomioida myös, että erilaisilla käännösten a kombinaatioilla voidaan päätyä samasta alkutilasta s_i samaan lopputilaan s_j. Sama pätee käänteisesti Rubikin kuutiota ratkaistaessa; tilasta s_0 sallittujen käännösten mielivaltaisten kombinaatioiden avulla muodostetusta tilasta $s \in S'$ voidaan päästä takaisin ratkaistuun tilaan erilaisia strategioita tai käännösjonoja käyttäen.

3 RUBIKIN KUUTION RATKAISEMINEN

Rubikin kuutiota voidaan tutkia useista eri lähtökohdista, mutta yleisimmin yritetään etsiä kuution ratkaiseva käännösjono. Ratkaisemiselle voidaan asettaa erilaisia tavoitteita; useimmat haluavat vain päästä maaliin helpoiten hahmotettavaa ja löydettävää reittiä pitkin, kokeneemmat harrastajat yrittävät ratkaista kuution ajallisesti mahdollisimman nopeasti ja jotkut pyrkivät etsimään lyhyimmän mahdollisen ratkaisevan käännösjonon. Kaksi viimeksi mainittua risteävät osittain merkityksiltään, mutta eron voi määritellä siten, että ihminen pyrkii ratkaisemaan kuution nopeasti, kun taas kone pyrkii etsimään lyhyimmän mahdollisen reitin. Lyhyimmän reitin hahmottaminen voi olla todella vaikeaa ja sen vaatimat käännösjonot voivat olla fyysisesti suoritettuina hitaampia kuin hieman pidemmän ratkaisureitin jonot.

Rubikin kuutio on alusta asti kiinnostanut pulmapelien ystävien lisäksi matemaatikkoja ja tietojenkäsittelijöitä. Erityisen kiinnostava kysymys pitkään oli maksimipituus kuution optimaaliselle ratkaisureitille, eli miten monen askeleen päähän ratkaistusta tilasta kuutio voidaan sekoittaa. Vuonna 2014 tämän niin sanotun Jumalan luvun *(God's number)* osoitettiin olevan 20, mikäli puolen kierroksen käännökset lasketaan yhdeksi liikkeeksi, tai 26, mikäli ne lasketaan kahdeksi liikkeeksi. Tuloksen saaminen vei yhteensä noin miljardi sekuntia prosessointiaikaa Googlen palvelimilla. [11]

3.1 Ratkaisijana ihminen

Rubikilta itseltään kului yli kuukauden verran aikaa ratkaista kuutio ensimmäisen kerran. Ratkaisemisesta alettiin järjestää nopeuskilpailuja vuonna 1982 kuution levittyä suuren yleisön tietoisuuteen. Nykyinen ennätys viiden kuution ratkaisuajan keskiarvolle on 5,80 sekuntia. Viidestä ratkaisusta jätetään huomioimatta nopein ja hitain. [12]

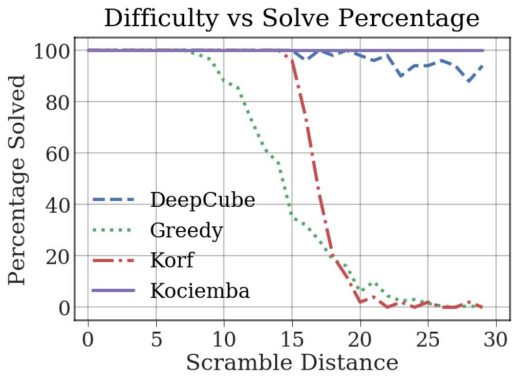
Ihmisen käyttöön suuniteltuja ratkaisutapoja on useita. Nopeuskilpailuissa suosituimmaksi on noussut edistyneempi Fridrich-metodi *(advanced Fridrich (CFOP))*, jonka toteuttamiseksi ratkaisijan tulee opetella jopa 78 erilaista algoritmeiksi kutsuttua käännösjonoa eri tilanteita varten. Toinen kilpailuissa käytettävä metodi, joka perustuu vähemmän ulkoaopeteltaviin algoritmeihin ja enemmän päättelyyn, on nimeltään Roux. Aloittelijalle suositeltavin lähestymistapa on helpotettu Fridrich-menetelmä, jonka opettelu ei vaadi kuin viiden algoritmin muistamisen ja hieman yksinkertaista päättelyä. [14]

3.2 Ratkaisijana tietokone

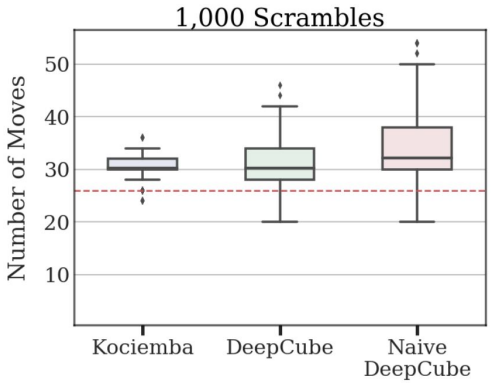
Tietokoneen laskentatehon ansiosta sillä voidaan tarkastella kuution tiloja paljon ihmistä laajemmin. Ainoa keino optimaalisen ratkaisun vahvistamiselle lienee käydä puumaisesti läpi kaikki löydettyä ratkaisua mahdollisesti lyhyemmät käännösjonot. Tunnetuimman tällaisen A*-hakuun perustuvan algoritmin kirjoitti Richard Korf vuonna 1998. Korfin algoritmin suoritusajalle ei ole laskettu ylärajaa, ja sen ajaminen voi viedä jopa useita tunteja.

Toinen, paljon nopeampi Kociemba-ratkaisija hyödyntää kuution ryhmäteoreettisia ominaisuuksia. Tässä algoritmissa on kaksi vaihetta, jotka vievät enimmillään puolen kierroksen käännökset sallittaessa 12 + 18 siirtoa, eli 10 enemmän kuin pisin optimaalinen reitti. Algoritmi siirtää kuution tilan ensin tiettyyn tilojen osajoukkoon, josta lopulliseen ratkaisuun pääsemiseksi tarvitsee käyttää käännöksiä vain tietystä käännösten osajoukosta. Kociemban ajoaika on noin 1 sekunti, joten se on paljon käyttökelpoisempi kuin Korf. Algoritmin koodi ja matemattinen selitys löytyvät Kociemban ylläpitämältä sivustolta. [2]

Näiden järjestelmällisempien ratkaisualgoritmien lisäksi monet ovat yrittäneet kehittää koneoppimiseen perustuvia ratkaisuja. Rubikin kuutio on kuitenkin osoittautunut sellaiseksi ongelmaksi, että useimmat koneoppipohjaiset ratkaisijat eivät onnistu luotettavasti ratkaisemaan yli viittä kertaa sekoitettuja kuutioita [5] [15]. Tämä muuttui, kun McAleer et al. kehittivät DeepCube-nimisen neuroverkkotekniikan, joka onnistuu ratkaisemaan 100 % sekoitetuista kuutioista. DeepCube oppi kuution ratkaisemisen säännöt ja tekniikat itsenäisesti vain tietyistä suorituksista saatavien palkkioiden avulla, joten koneoppitekniikkana se kuuluu vahvistavan oppimisen piiriin. Neuroverkon harjoittamiseen kului 44 tuntia aikaa 32-ytimisellä suorittimella ja kolmella ammattitason grafiikkaprosessorilla. [6]



Kuva 3.1. *Eri ratkaisijoiden onnistumisprosentit, kun algoritmeille annettiin 30 minuuttia aikaa kuution ratkaisemiseen [6].*



Kuva 3.2. *Kociemban ja DeepCuben antamien käännösjonojen pituudet 1000 kertaa sekoitetuilla kuutioilla [6].*

Kuvasta 3.1 nähdään, miten Kociemba onnistuu ratkaisemaan kuutiot aina, ja kuinka Korf-algoritmin laskenta-aika kasvaa jo 17 sekoituksen jälkeen puolituntiseksi. DeepCube onnistuu löytämään keskimäärin yhtä lyhyitä ratkaisuja neljäosakierroksen säännöillä kuin Kociemba, mutta 10 minuutin keskimääräisellä ratkaisuajalla se on vielä paljon Kociembaa hitaampi. Kuvassa 3.2 punaisen katkoviivan kohdalla on optimaalisen reitin vaatima siirtojen enimmäismäärä, niin sanottu Jumalan luku 26.

4 AGENTIN OPETTAMINEN

Luvussa 2 rakennetun mallin tila $s \in S'$ esitettiin ohjelmallisesti tietorakenteena ja käännökset $a \in A$ sitä muokkaavina funktioina, jotta *agentilla*, eli koneoppimisen tuloksena saatavalla toimijalla, olisi ympäristö, jonka rajoissa harjoitella ratkaisemista. Kuution esittämiseen ja hallintaan on kehitetty useita erilaisia tietorakenteita, jotka optimoivat erilaisten käyttötarkoitusten tavoitteita. Howard A. Peelle on esitellyt ja vertaillut toimivimpia rakenteita julkaisussaan Representing rubik's cube in APL (1984) [9].

Kuution tietorakenteessa päätettiin pyrkiä tilan ymmärrettävyyden ja seuraamisen helppouteen muistin tai laskenta-ajan optimoinnin sijasta. Mallissa kuutio esitettiin todellisuutta läheisesti vastaavana kolmiulotteisena $6 \times 3 \times 3$ matriisina. Tällöin visualisointi ja rakenteen ymmärtäminen oli helppoa, koska kuution sivuja esittävät matriisit ja kuution käännökset voitiin tulostaa avattua kuutiota esittävään muotoon kuvan 2.3 mukaisesti.

Ratkaisemisen opettamiseen mietittiin erilaisia keinoja. Ohjattuna oppimisena *(supervised learning)* agentille voitaisiin syöttää sekoitettuja kuutioita ja ne ratkaiseva sarja käännöksiä. Tällöin sekoitettua kuutiota ratkaistaessaan agentti vertaisi kuution tilaa aiemmin näkemiinsä, etsisi jonkin säännön mukaan uutta tilaa lähimpänä olevan jo opitun tilan ja suorittaisi sekoitetulle kuutiolle opittua tilaa vastaavan käännösjonon. Tilojen läheisyyden määritelmä riippuisi toteutuksesta. Lauseen 2.1 perusteella agentin täytyisi taulukoida kuution jokainen tila. Agentilla ei siis olisi edes mahdollisuutta oppia ratkaisemaan kuutioita, joiden ratkaisua sille ei olisi kerrottu. Perinteiset ohjatun oppimisen keinot, jotka eivät perustu neuroverkkoihin, olisivat siis suurelta osin vain ratkaisujen taulukointia. Rubikin kuution tilojen valtavan määrän takia näitä keinoja olisi mahdoton skaalata sekoitusmäärien lisääntyessä.

4.1 Q-Learning

Toinen kirjoittajalle ennestään tuttu koneoppimismetodi oli vahvistavan oppimisen *(reinforcement learning)* Q-Learning-algoritmi. Siinä agentti oppii kuution ratkaisemisen itsenäisesti, eli agentille ei tarvitse antaa kuutioiden ratkaisemiseen tarvittavia käännösjonoja. Q-Learning-algoritmi rakentaa taulukkoa, jossa jokaisen tilan mahdolliset käännökset on pisteytetty. Ratkaisuvaiheessa agentti valitsee taulukosta korkeimmat pisteet antavan käännöksen joka tilanteessa ja saavuttaa siten ratkaistun tilan.

Algoritmi oppii optimaalisen käännöksen kussakin tilanteessa pelaamalla peliä useita ker-

toja. Aluksi agentti valitsee seuraavan käännöksen täysin satunnaisesti, koska se ei vielä tiedä pelaamansa pelin sääntöjä tai tavoitteita. Aikanaan agentti onnistuu sattumalta tekemään jotakin, josta algoritmi palkitsee sen pisteillä. Rubikin kuution tapauksessa pisteitä saa yksinkertaisimmillaan vain ratkaisemalla kuution, mutta pisteitä voitaisiin jakaa myös ennalta määriteltyjen kuutiota edistävien tilojen saavuttamisesta.

Kun agentti on kerran saanut pisteitä tietystä käännöksestä tietyssä tilassa, se osaa myöhemminkin tehdä saman käännöksen kyseisessä tilassa. Ratkaistuun tilaan pääsemisestä saatavat pisteet jaetaan myös reitin edeltävillä tiloilla tehdyille liikkeille, jotta agentti osaisi tulevaisuudessa seurata samaa reittiä pitkin ratkaisuun. Näin opittu reitti ei kuitenkaan välttämättä ole optimaalinen, joten agentin kannattaa tutkia harjoitteluvaiheessa myös muita käännösvaihtoehtoja lyhyemmän reitin löytämiseksi. Ratkaisusta periytyvät pisteet R vähenevät jokaisella askeleella liikuttaessa kauemmas ratkaisusta, joten lyhyemmälle reitille johtavien käännösten pisteet jäävät korkeammiksi kuin pidemmän reitin. Pisteiden oppiminen tapahtuu yhtälön

$$Q(s,a)_{i+1} = Q(s,a)_i + \beta \left[r + \gamma \max_{a'} Q(s \cdot a, a')_i - Q(s,a)_i \right]$$

avulla. Funktio $Q(s,a)$ on taulukko, joka palauttaa tietyssä tilassa tehtävän tietyn käännöksen palkkion. Funktio $R = \gamma \max_{a'} Q(s \cdot a, a')$ palauttaa valitusta käännöksestä a aiheutuneen tilan $s \cdot a$ skaalatun maksimipalkkion. Tietyssä tilassa tehtävän käännöksen pisteet riippuvat siis välittömästi saatavista pisteistä r ja kyseistä reittiä pitkin tulevaisuudessa saatavista korkeimmista mahdollisista pisteistä R. Kerroin $\beta \leq 1$ vaikuttaa oppimisprosessin nopeuteen ja $\gamma < 1$ säätelee periytyvien pisteiden vähenemistä. Funktion Q oppiminen onkin iteratiivinen prosessi. [19]

Agentille annetaan välittömänä palkkiona ratkaistuun tilaan johtaneesta käännöksestä 10 pistettä ja jokaisesta muusta käännöksestä yksi miinuspiste.

$$r = \begin{cases} 10, & \text{jos } s_{i+1} = s_i \cdot a = s_0 \\ -1, & \text{jos } s_{i+1} = s_i \cdot a \neq s_0 \end{cases} \tag{4.1}$$

Nämä palkkioiden suuruudet valittiin yksinkertaisiksi, jotta harjoitteluvaiheessa niiden summana saatavien kokonaispisteiden tarkastelu olisi helppoa. Palkkioiden suuruuksia valittaessa oletettiin algoritmin toiminnan kannalta riittäväksi, että ratkaisuun johtavasta käännöksestä saa enemmän pisteitä kuin muista. Moniosaisemmassa palkkiojärjestelmässä eri palkkioiden suuruuksien suhteilla voisi olla isompi merkitys.

Yleinen tapa ohjata agentin päätöksentekoa on antaa sen aluksi valita toimintoja satunnaisesti, jotta se oppii ympäristönsä säännöt. Satunnaisuuden osuutta vähennetään harjoittelun edetessä, jolloin agentti hyödyntää enemmän aiemmin oppimaansa ja pääsee siten tehokkaammin kohti tavoitetta. Tästä satunnaisuuden ja jo opitun käyttämisen tasapainotuksesta käytetään termiä *exploration and exploitation*.

4.2 Deep Q-learning

Myös Q-learning joutuu taulukoimaan suurimman osan tiloista, jotta sen avulla voidaan ratkaista rubikin kuutio. Sekään ei siis voi skaalautua tehokkaasti sekoitusmäärien lisääntyessä. Q-learningin käyttäminen onkin käytännöllisempää tapauksissa, joissa kaikki mahdolliset tilat ja niiden ratkaisut voidaan taulukoida, mutta joissa taulukointi manuaalisesti olisi algoritmin ajamista työläämpää. Tästä esimerkkinä Q-learningin "Hello World"-tehtävä taxi-v2 [3].

Esitellyn DeepCube-ratkaisijan ja DeepMindin Atari-peleissä saavuttaman menestyksen takia työssä päätettiin kokeilla deep Q-learning -neuroverkkotekniikkaa [7] [6]. Toteutettava neuroverkko muistuttaa DeepMindin kehittämää DQN-agenttia, mutta sitä on paikoittain yksinkertaistettu [8]. Eroavaisuutena Q-learning-algoritmiin seuraavaa askelta ei katsotakaan taulukosta, vaan neuroverkko pyrkii ennustamaan ratkaisua kohti vievän käännöksen. Algoritmin kirjoittamisessa käytettiin apuna netissä julkaistuja artikkeleita [16] [10] ja Python-ohjelmointikielellä käytettävää TensorFlow-kirjastoa.

Käytettävä neuroverkko koostuu kuution tilasta arvonsa saavasta syötekerroksesta, täysin yhdistetyistä piilotetuista kerroksista *(hidden layers)* ja lopulta tuloskerroksesta. Täysin yhdistetyssä tai tiheässä *(dense)* kerroksessa jokainen solmu *(node)* saa syötteen $x_{i,j}$ edellisen kerroksen jokaisesta solmusta. Opetusvaiheessa verkko pyrkii oppimaan haluttuun lopputulokseen johtavat liitoskohtaiset painokertoimet $w_{i,j,k}$ ja solmukohtaiset vakiot $b_{i,j}$. Indeksoinnissa muuttujat i, j ja k numeroivat neuroverkon kerrosta, solmua ja liitosta.

Jokaisessa solmussa kerrotaan jokaiselta edellisen kerroksen solmulta saatu syöte $x_{i-1,k}$ liitosta vastaavalla painotuksella $w_{i,j,k}$, summataan tulot, lisätään vakio $b_{i,j}$ ja skaalataan tulos aktivaatiofunktiolla f_a.

$$x_{i,j} = f_a\left(b_{i,j} + \sum_{k=1}^{n} x_{i-1,k} w_{i,j,k}\right)$$

Tässä työssä aktivaatiofunktioksi valittiin syvissä verkoissa suosittu ReLU (Rectified Linear Unit), joka muuttaa negatiiviset arvot nollaksi:

$$f_a(y) = \max(0, y).$$

Toinen yleinen aktivaatiofunktio on Sigmoid -funktio, joka skaalaa arvon reaalilukuvälille $x_{i,j} \in [0, 1]$. [1]

Kerroksen jokainen solmu syöttää saamansa tuloksen $x_{i,j}$ seuraavan kerroksen kaikille solmuille, joissa toimenpide toistetaan. Lopulta saavutetaan viimeinen kerros, jossa on yleensä yksi solmu jokaista mahdollista tulosta kohti. Suurimman arvon antavaa solmua vastaava tulos valitaan verkon lopulliseksi ennusteeksi. Tässä työssä sallitaan 12 eri käännöstä a, joten neuroverkon tuloskerroksessa on aina 12 solmua.

Painotuskertoimet $w_{i,j,k}$ opitaan *backpropagation*-menetelmällä, jossa käytetään *gradient*

descent -optimointialgoritmeja. Tämän työn neuroverkossa optimointiin käytetään *Adam*-algoritmia, joka perustuu gradienttimenetelmään. [13] Harjoitteluvaiheessa neuroverkon tuloskerrokselta saadaan ennustevektori $\mathbf{t_e}$, jota verrataan haluttuun tulosvektoriin $\mathbf{t_h}$ virhefunktiolla *(loss funtion)* f_v. Tässä työssä käytetään neliöllisen virheen keskiarvoa *(Mean Squared Error, MSE)*

$$f_v(\mathbf{t_e}, \mathbf{t_h}) = \frac{1}{n}\sum_{i=1}^{n}(t_{e_i} - t_{h_i})^2.$$

Työn verkko ennustaa tilassa s tehtävistä käännöksistä a saatavaa palkkiota

$$t_{e_i} = r_{a_i} + \gamma \max_{a'} Q(s \cdot a_i, a').$$

Virhefunktion haluttuja arvoja $\mathbf{t_h}$ laskettaessa välitön palkkio r tiedetään varmaksi ja periytyvä osa R saadaan verkon ennusteesta. Painoja säädettäessä virhefunktiolle lasketaan gradientti verkon jokaisen solmun suhteen ja painoja muutetaan hieman negatiivisen gradientin suuntaan virhefunktion minimoimiseksi.

Työssä kokeiltiin verkon koon vaikutusta muuttamalla piilotettujen kerrosten sekä niissä olevien solmujen määriä. Jatkossa verkon koosta puhuttaessa verkon piilotettujen kerrosten koko ja määrä voidaan ilmoittaa merkinnällä [120, 50, 30]. Tätä luetaan siten, että ensimmäisessä piilotetussa kerroksessa on 120 solmua, seuraavassa 50 ja viimeisessä 30.

4.3 Tehostavat tekniikat

Rubikin kuution ratkaiseminen satunnaisesti reittiä etsimällä on tehoton tapa jo pienilläkin sekoitusmäärillä, koska hyvin harva käännösjono päätyy ratkaistuun tilaan. Opettamisessa kokeiltiin tehtävän vaikeustason nostamista asteittain. Ensin agentille syötettiin kerran käännettyjä kuutioita, minkä ansiosta se oppi nopeasti valitsemaan viimeisen käännöksen oikein. Seuraavaksi ratkaistavia kuutioita oli sekoitettu kahteen kertaan, jolloin agentille riitti löytää lyhin tie sen jo tuntemalle, yhden siirron päässä olevalle tilalle. Samaa jatkettiin, kunnes neuroverkko ei enää kyennyt ratkaisemaan suurinta osaa testikuutioista muutaman minuutin harjoittelun jälkeen. Opetellessaan agentti otti siis edelleen askeleita sattumanvaraisesti, mutta lähempänä tavoitetta olevien tilojen etukäteenopettelun toivottiin tehostavan ratkaisuun pääsemistä. Lisäksi ensimmäisen askeleen jälkeen satunnaisen käännöksen tekemisen todennäköisyyttä laskettiin huomattavasti, jotta agentti todella hyödyntäisi jo oppimaansa tietoa ratkaisuun pääsemiseksi. Ratkaisemiseen käytettävä käännösten maksimimäärä rajoitettiin, jotta agentti ei jäisi jumiin sen sekoittaessa kuutiota vain enemmän, koska agentin pääseminen takaisin tunnetuille tiloille pienenkin eksymisen jälkeen on epätodennäköistä.

Täysin satunnaisten syötteiden käyttäminen oletettiin tässä tapauksessa riittäväksi, mutta yleisesti olisi hyvä etsiä mahdollisia erittäin harvinaisia ongelmatyyppejä ja pakottaa

agentti opettelemaan ne. Satunnaisilla syötteillä nuo tilanteet saattavat jäädä huomioimatta, mikä voi aiheuttaa käyttövaiheessa odottamattomia virheitä. Agentille annettavista kuutioista ei myöskään rajattu mitään tiloja pois harjoitteluvaiheessa, joten testausvaiheessa se ratkaisi suurella todennäköisyydellä vain jo käsittelemiään tiloja. Tämä vaikeuttaa ylioppimisen *(overfitting)* tarkastelemista.

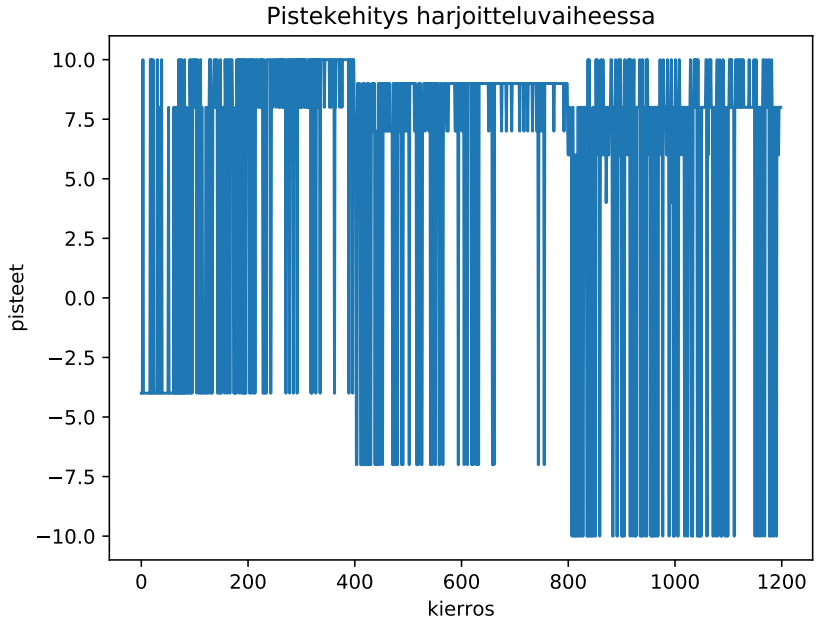
Kuution tilan esittämiseksi alkuperäinen tilamatriisi hajotettiin 54×6 matriisiksi, jossa rivit edustavat yksittäistä ulospäin näkyvää pikkupalan ruutua ja sarakkeet kuutta palalle mahdollista väriä. Jokaiselta riviltä asetettiin ruudun väriä vastaavan indeksin arvo ykköseksi ja muut jätettiin nollaksi. Lopuksi tämä matriisi vielä litistettiin yhdeksi pitkäksi vektoriksi. Tällä pyrittiin estämään isommilla luvuilla tapahtuvia oireiluja laskutoimituksissa, kuten yhden värin arvottaminen toista korkeammalle vain sitä kuvaavan suuremman numeerisen arvon takia. Tälle menettelylle on englanninkielinen termi *One Hot Encoding* ja yksinkertaisuudessaan sillä muokataan kategorisoiva data binääriseksi. Verkolle syötettävästä tilasta oltaisiin luultavasti voitu jättää keskipaloja vastaavat ruudut pois, koska niiden värit ovat aina samat.

Q-Learning-algoritmissä kuljetun reitin käsittely ja oppiminen suoritetaan usein heti ratkaisun löydyttyä. Tämän lähestymistavan lisäksi kouluttamisessa kokeiltiin myös tapahtumia tallentavaa muistia *(replay memory)*. Muistiin tallennetaan tehtyä käännöstä edeltävä tila, käännös, saatu välitön palkkio sekä uusi tila. Oppimisvaiheessa muistista haetaan suorituksia, joiden mukaisesti verkko säätää painotuskertoimiaan. Tällä keinolla pyritään ehkäisemään uuden oppiminen vanhan päälle toistamalla kerran suoritettuja liikkeitä myöhemminkin. Samalla neuroverkon painot suppenevat toiston ansiosta kohti tasapainopistettä kertoimen β hidastaessa oppimista yhtälössä 4.1.

Muistia voi käyttää painottaen eri asioita. Aluksi voidaan kerätä iso määrä testidataa, joka sisältää mahdollisesti hyvin erilaisissa ympäristöissä suoritettuja toimintoja. Tällöin estetään ympäristössä tapahtuvan suuren muutoksen aiheuttama aiemmin opitun unohtaminen. [8] Rubikin kuutiossa tällaista suurta muutosta ei tule, ja parempi esimerkki menetelmän hyödyistä olisikin jokin tasohyppelypeli, jossa esimerkiksi liikenopeuteen ja hyppimiseen vaikuttaisi suuresti pelattavan tason painovoima. Ennen opettelemista kerättävän datan määrää kannattaa tasapainotella, sillä agentti ei opi uutta ennen datan käsittelyä vaan jatkaa satunnaisten toimintojen suorittamista. Tämä ei välttämättä tuota parasta mahdollista dataa oppimisen kannalta.

5 TULOKSET ERI MENETELMISTÄ

Tarkastellaan aluksi satunnaisuuden ja jo opitun hyödyntämistä harjoittelun eri vaiheissa. Kuvassa 5.1 esitellään agentin saamat kierroskohtaiset kokonaispisteet harjoittelun edetessä. Kokonaispisteet muodostuvat kyseisen ratkaisureitin välittömien palkkioiden 4.1 summana. Jokaisen vaiheen alussa agentti on epäonnistunut suuressa osassa kuution ratkaisuja tehdessään satunnaisia käännösvalintoja. Vaiheen lopussa agentti hyödyntää ennestään oppimaansa, jolloin se onnistuu ratkaisemaan lähes kaikki kuutiot. Ensimmäisen ja toisen vaiheen loppupään epäonnistumiset johtuvat lähinnä olemassa olevasta pienestä satunnaisen käännöksen mahdollisuudesta, jolla algoritmi yrittää löytää optimaalisempia reittejä. Testausvaiheessa, jossa agentti ei tee ollenkaan satunnaisia käännöksiä, yhden ja kahden sekoituksen kuutiot ratkeavat > 99 % varmuudella.



Kuva 5.1. *Agentin saamat pisteet opetuksen eri vaiheissa.*

Jokaisessa vaiheessa esiintyvät optimaalista ratkaisua kahdella käännöksellä pidemmät ratkaisureitit, eli 8, 7, ja 6 pisteen kierrokset, ovat mielenkiintoisia, koska niitä vaikuttaa kuvaajan perusteella olevan melko paljon. Ylimääräiset askeleet voisivat aiheutua reitistä, jossa agentti oppisi jonkin yksittäisen käännöksen tekemisen sijaan kääntämään samaa tahkoa kolme kertaa eri suuntaan, jolloin lopputulema on tietenkin sama. Tällaista liikesarjaa algoritmi ei välttämättä optimoisi pois automaattisesti ennen kuin se kerran ko-

keilisi kiertää kuution tahkoa oikeaan suuntaan. Ylimääräiset käännökset voivat johtua myös satunnaisen käännöksen aiheuttamasta harha-askeleesta, josta agentti on onnistunut palautumaan. Testausvaiheessa agentti käytti epäoptimaalisia reittejä vain noin 1 % ratkaistuista tapauksista, joten kuvaajassa esiintyvät epäoptimaalisuudet lienevät satunnaisuuden aiheuttamia.

Kolmen sekoituksen vaiheessa nähtävät 10 pisteen suoritukset johtuvat sekoitusvaiheessa tehdyistä kahdesta toisensa kumoavasta käännöksestä, jolloin kuutio näyttää kerran sekoitetulta. Epäonnistuneiden yritysten määristä on pääteltävissä, että kolme kertaa sekoitetun kuution ratkaisemisen opetteluun tarvitaan jo paljon enemmän harjoittelukierroksia. Aiemmissa vaiheissa on sen sijaan käytetty ylimäärin resursseja ja opettamisen optimoinniksi niitä voisi vähentää. Agentin enimmäiskäännösmäärä oltaisiin myös voitu rajoittaa tiukemmin, koska agentti ei yleensä päässyt ratkaisuun otettuaan kaksikin harha-askelta. Tässä tapauksessa harjoitteluaika oli melko lyhyt, joten optimoinnilla ei olisi saavutettu merkittäviä aikasäästöjä. Yleisessä tapauksessa laskenta-ajan käyttöä on kuitenkin hyvä miettiä.

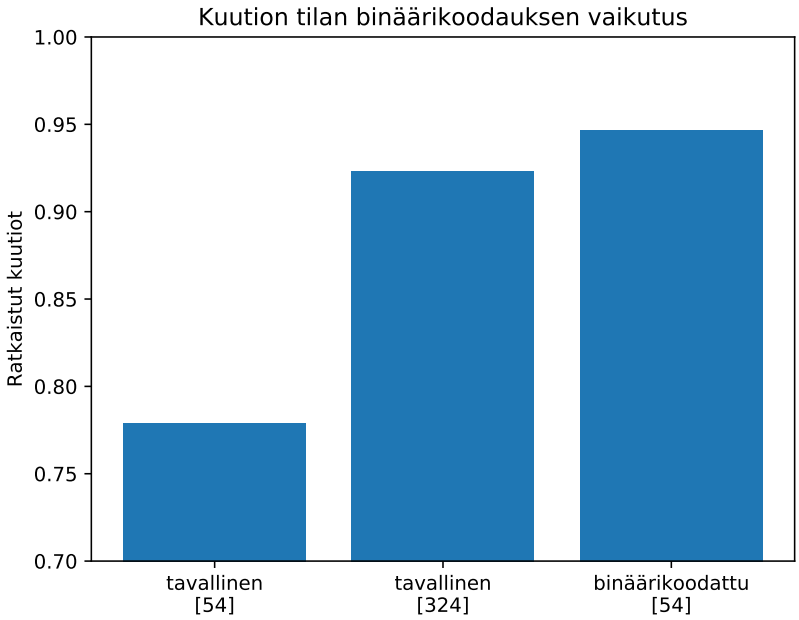
5.1 One hot encoding

Työssä kokeiltiin tilan koodauksen vaihtamista alkuperäisestä kategorisoivasta kokonaislukuesityksestä binäärivektorimuotoon. Seuraavassa kuvassa 5.2 esitellään kuution tilan esitystavan vaikutusta agentin oppimiskykyyn. Tulokset ovat kyseisillä asetuksilla saatujen eri opetus- ja testauskertojen ratkaisuprosenttien keskiarvoja. Ratkaisukykyä testattaessa agentin tuli yrittää ratkaista 1000 satunnaisesti kolmella käännöksellä sekoitettua kuutiota.

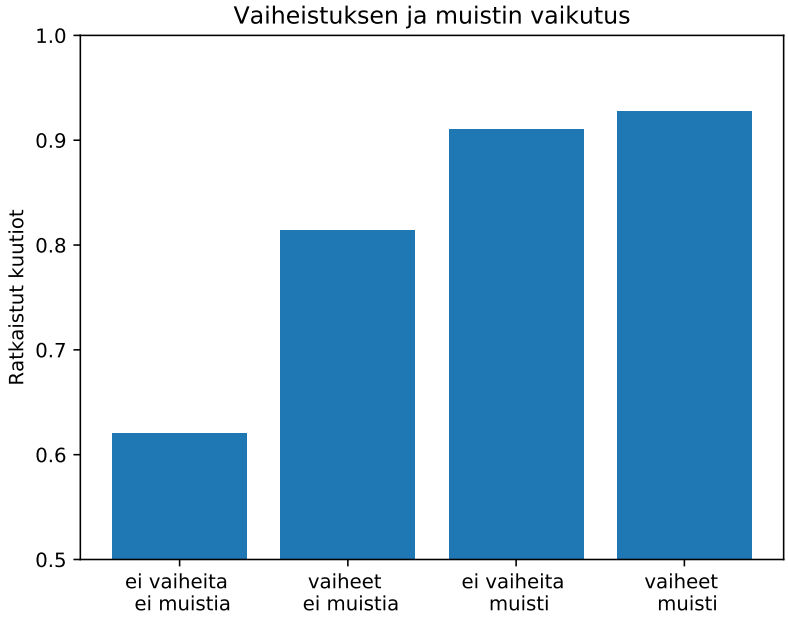
Kuution tilan esitystavan muuttaminen binäärivektorimuotoon lukujen 1-6 käyttämisen sijasta nosti ratkaisutuloksia 10-20 prosenttiyksikköä sekoitusmäärästä riippuen. Tulos ei kuitenkaan ole välttämättä täysin binääriesityksen paremmuuden ansiota, sillä se samalla kuusinkertaistaa neuroverkon ensimmäisen kerroksen liitäntöjen määrän. Jos liitosten määrä nostetaan samaan kasvattamalla neuroverkon ensimmäistä kerrosta 54 solmusta 324 solmuun, ovat tulokset lähes yhtä hyvät. Tällöin täysin yhdistetyn kerroksen parametrien määrä on molemmissa lähes sama. Piilotetun kerroksen solmujen määrän kasvattaminen toki nostaa parametrien määrää myös kerroksen toisella puolella.

5.2 Replay memory ja opetuksen vaiheistus

Työssä kokeiltiin myös muistin ja vaiheittaisen oppimisen vaikutusta agentin oppimiskykyyn. Tuloksia tarkasteltaessa on hyvä huomioida, että muistia käytettäessä jokainen agentin tekemä käännös hyödynnetään keskimäärin 100 kertaa harjoittelussa. Tämä aiheuttaa enemmän laskentaa kuin ilman muistia, mutta kerroin olisi luultavasti voitu valita paljon pienemmäksi ilman, että tulokset olisivat kärsineet.



Kuva 5.2. *Kuution tilan koodauksen vaikutus kolmesti sekoitetuissa kuutioissa. Hakasulkeissa neuroverkon solmujen määrä piilotetuissa kerroksissa.*



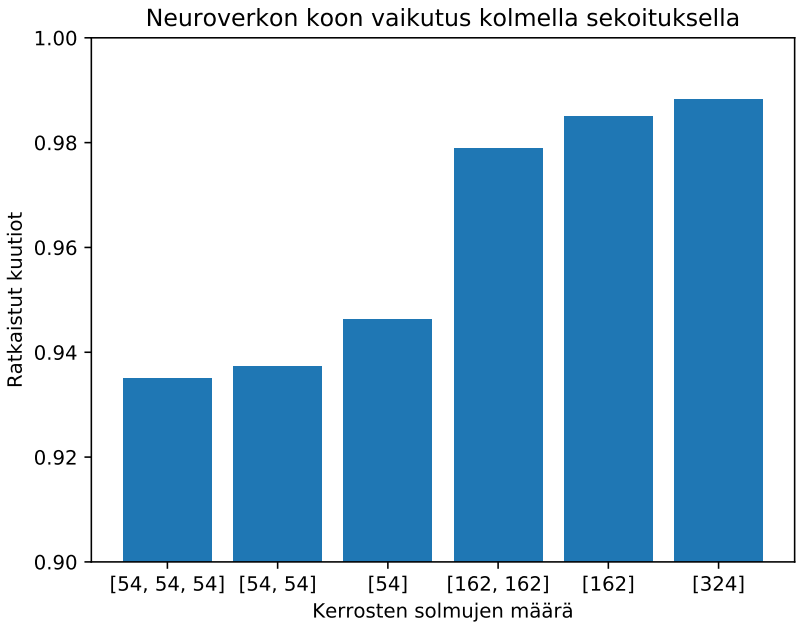
Kuva 5.3. *Muistin ja vaiheistuksen hyöty opettelussa.*

Kuvasta 5.3 nähdään opettamisen vaiheistuksella olevan merkittävä vaikutus oppimiskykyyn, jos muistia ei käytetä. Ilman muistia agentin on vaikea oppia kuution ratkaiseminen suoraan kolmesti sekoitetusta tilasta, koska se ei ensimmäisten askelten jälkeen tiedä vielä menevänsä oikeaan suuntaan. Tämä vähentää ratkaisuun päätyviä harjoittelukertoja ja hidastaa käännöksistä saatavien palkkioiden oppimista. Vaiheistetussa oppimisessa agentti osaa hyödyntää helpommassa ympäristössä aluksi oppimaansa tietoa.

Pienillä sekoitusmäärillä ja suurella muistinkäytön kertoimella pelkän muistin käyttäminen tuottaa lähes yhtä hyviä tuloksia kuin muistin ja vaiheistuksen käyttäminen yhdessä. Muistia käytettäessä tavoitteeseen ei tarvitse päästä niin usein tai aikaisessa vaiheessa, kuin ilman muistia, koska agentti pystyy harjoittelemaan kertaalleen tehdyllä suorituksella uudelleen. Tällöin epäonnistuneiden harjoituskertojen käännökset eivät mene täysin hukkaan, koska yhtälön 4.1 päivittäminen onnistuu yksittäisten käännösten uudelleentarkastelulla, kunhan seuraavan tilan pisteet $Q(s \cdot a, a')$ on jo opittu lähes todellisuutta vastaaviksi. Suuremmilla sekoitusasteilla vaiheistuksen rooli saattaisi korostua jälleen, koska agentin todennäköisyys päästä maaliin satunnaisilla käännöksillä pienenee sekoitusta lisättäessä.

5.3 Neuroverkon koon vaikutus ratkaisukykyyn

Työssä tarkasteltiin myös neuroverkon koon vaikutusta agentin ratkaisukykyyn. Kuvassa 5.4 on esiteltynä eri verkkokonfiguraatioilla saatuja tuloksia.



Kuva 5.4. *Neuroverkon koon vaikutus oppimiseen.*

Kuvaajasta havaitaan, että verkon kerrosten lisäämisellä ei ole juuri mitään vaikutusta tai vaikutus on jopa negatiivinen agentin ratkaisukykyyn, kun taas solmujen lisääminen kerrokseen kasvattaa ratkaisukykyä. Myös muita kuin tasasolmuisia verkkokonfiguraatioita kokeiltiin ja tulokset olivat aina esitetyn kaltaisia. Neljällä sekoituksella saatiin vastaavia tuloksia, mutta kerroksen solmujen määrää tuli edelleen nostaa samoihin ratkaisuprosentteihin pääsemiseksi.

Saadut tulokset ovat sinänsä tyydyttäviä, koska tuotettu agentti oppi ratkaisemaan kolme ja neljä kertaa sekoitettuja kuutioita. On kuitenkin hyvin mahdollista, että neuroverkko

onnistui suuren parametrimäärän avulla opettelemaan ratkaisuihin tarvittavat käännösjonot ulkoa. Harjoitteluvaiheeseen käytettiin 5000 kierrosta ja kolmella käännöksellä sekoitettaessa kuutio saattoi päätyä 1080 erilaiseen tilaan [17]. Agentti siis kohtasi varmasti lähes kaikki testausvaiheessa saamansa tilat jo harjoitellessaan. Myös verkon syvyyden minimaalinen vaikutus agentin ratkaisukykyyn saattaa viestiä siitä, ettei verkko oikeasti oppinut mitään älykästä kuution ominaisuuksista tai ratkaisutavoista.

6 YHTEENVETO

Työssä pyrittiin tuottamaan vahvistavalla koneoppimisella neuroverkko, joka kykeni ratkaisemaan kolmella neljäsosakierroksen käännöksellä sekoitetun Rubikin kuution. Opettamiseen valittiin täysin kytketyistä kerroksista koostuva deep Q-learning -niminen neuroverkkotekniikka, koska sen yksinkertainen versio Q-Learning oli kirjoittajalle jo ennestään tuttu ja sillä ajateltiin olevan edes jonkinlainen mahdollisuus skaalautua taulukointia paremmin. Työssä keskityttiin selvittämään valitun neuroverkon toimintaperiaatteita sekä erilaisten oppimisprosessiin vaikuttavien avustavien tekniikoiden ja parametrien vaikutusta lopulliseen ratkaisukykyyn.

Lopulta neuroverkko onnistui yhden ainoan piilotetun kerroksen solmujen määrästä riippuen ratkaisemaan noin 95-98 % satunnaisesti kolmella käännöksellä sekoitetuista kuutioista. Algoritmi siis oppi itsenäisesti ratkaisemaan Rubikin kuution, joten työn tavoite saavutettiin. Käytetyistä tekniikoista etenkin eri tilanteissa suoritettuja käännöksiä tallentava muisti osoittautui hyödylliseksi. Sen avulla agentti pystyi harjoittelemaan samoja tilanteita useaan kertaan, mikä auttoi vahvistamaan iteratiivisesti opittavia neuroverkon painokertoimia. Vastaavasti vaiheittainen opettelu helpoista tilanteista vaikeampiin siirtyen vaikutti tehostavan oppimista. Havaittiin myös, että neuroverkon kerrosten lisääminen ei nostanut agentin ratkaisukykyä ollenkaan, mikä aiheuttaa epäilyjä siitä, että agentti ei oppinut kuutiosta mitään erityistä vaan se saattoi onnistua muistamaan ratkaisureitit verkon parametrien avulla.

Rubikin kuutio ei ole ongelmana erityisen otollinen käytetyille menetelmille, koska sen tila-avaruus on diskreetti joukko ja yhden tilan ratkaisusta opitut liikkeet eivät sellaisenaan auta toisen tilan ratkaisemisessa. Tavoitetilat ovat myös todella vähälukuiset suhteessa kaikkiin tiloihin, jos agenttille annetaan palkkioita vain ratkaisuun pääsemisestä. Suuremmille sekoitusmäärille skaalautuakseen neuroverkon tulisi kyetä oppimaan jotain abstraktia kuution ratkaisemisesta ulkoaopettelun sijaan, kuten McAleer et al. onnistuivat tekemään työssään Solving the Rubik's Cube Without Human Knowledge [6]. Tilanteen haastavuus tiedostettiin jo työn alussa, koska aiemmat kokeilut olivat myös olleet vaikeuksissa jo muutamien sekoituskertojen jälkeen, vaikka niissä oli käytetty tässä työssä esiteltyjä menetelmiä edistyksellisempiä tekniikoita [5]. Työ ajoi kuitenkin pohtimaan erilaisia vahvistavaan oppimiseen ja neuroverkkoihin liittyviä kysymyksiä, joihin onnistuttiin löytämään opettavaisia ratkaisuja.

LÄHDELUETTELO

[1] *Activation functions*. https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html. luettu 28.1.2019.

[2] *Herbert Kociemba. Two-phase algorithm details.* http://kociemba.org/math/imptwophase.htm. luettu 18.12.2018.

[3] *Introduction to reinforcement learning and OpenAI Gym*. https://www.oreilly.com/learning/introduction-to-reinforcement-learning-and-openai-gym. luettu 13.2.2019.

[4] R. E. Korf. Finding optimal solutions to Rubik's cube using pattern databases. Teoksessa: *AAAI/IAAI*. 1997, 700–705.

[5] P. Lichodzijewski ja M. Heywood. The Rubik cube and GP temporal sequence learning: an initial study. Teoksessa: *Genetic Programming Theory and Practice VIII*. Springer, 2011, 35–54.

[6] S. McAleer, F. Agostinelli, A. Shmakov ja P. Baldi. Solving the Rubik's Cube Without Human Knowledge. *arXiv preprint arXiv:1805.07470* (2018).

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra ja M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al. Human-level control through deep reinforcement learning. *Nature* 518.7540 (2015), 529.

[9] H. A. Peelle. Representing rubik's cube in APL. Teoksessa: *ACM SIGAPL APL Quote Quad*. Vol. 14. 4. ACM. 1984, 255–262.

[10] *RL—DQN Deep Q-network*. https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4/. luettu 13.2.2019.

[11] T. Rokicki, H. Kociemba, M. Davidson ja J. Dethridge. The diameter of the Rubik's Cube group is twenty. *SIAM Review* 56.4 (2014), 645–670.

[12] *Rubik's Cube World Records*. http://www.recordholders.org/en/list/rubik.html. luettu 26.1.2019.

[13] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[14] *Ruwix Rubik's Cube Wiki*. https://ruwix.com/the-rubiks-cube/different-rubiks-cube-solving-methods/. luettu 18.12.2018.

[15] R. J. Smith, S. Kelly ja M. I. Heywood. Discovering Rubik's Cube Subgroups using Coevolutionary GP: A Five Twist Experiment. Teoksessa: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM. 2016, 789–796.

[16] *Solving Mountain Car with Q-Learning*. https://adventuresinmachinelearning.com/reinforcement-learning-tensorflow/. luettu 8.1.2019.

[17] *Tomas Rokicki. God's number is 26 in the quarter-turn metric.* `https://www.cube20.org/qtm/`. luettu 16.1.2019.

[18] *Viralliset Rubik's brandin sivut*. `https://rubiks.com/`. luettu 18.12.2018.

[19] C. J. Watkins ja P. Dayan. Q-learning. *Machine learning* 8.3-4 (1992), 279–292.