Hertta Leinonen

MuZero ja mallipohjainen vahvistusoppiminen

Tietotekniikan kandidaatintutkielma

11. toukokuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Hertta Leinonen

Yhteystiedot: hertta.a.leinonen@student.jyu.fi

Ohjaaja: Leevi Annala

Työn nimi: MuZero ja mallipohjainen vahvistusoppiminen

Title in English: MuZero and model-based reinforcement learning

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 26+0

Tiivistelmä: Tutkielmassa pyritään selvittämään, mitä mallipohjainen vahvistusoppiminen tarkoittaa, ja kuinka sitä hyödynnetään MuZero-nimisen tekoälyn algoritmissa. MuZeroa on testattu menestyksekkäästi sekä klassisissa lautapeleissä, että visuaalisesti monimutkaisissa Atari –peleissä. MuZero yhdistää toiminnassaan syvän mallipohjaisen vahvistusoppimisen, sekä Monte Carlo -puuhaun, saavuttaen kyvyn suoriutua keskenään hyvin erilaisista peleistä tuntematta niiden sääntöjä entuudestaan.

Avainsanat: MuZero, tekoäly, syväoppiminen, mallipohjainen vahvistusoppiminen, algoritmit, Monte Carlo -puuhaku, DeepMind

Abstract: The aim of this thesis is to find out what model-based reinforcement learning is and how it is utilized in MuZero's algorithm. MuZero has been successfully tested in both classic board games and visually complex Atari games. MuZero combines deep model-based reinforcement learning with Monte Carlo tree search, achieving the ability to play different games without knowing their rules.

Keywords: MuZero, artificial intelligence, deep learning, model-based reinforcement learning, algorithms, Monte Carlo tree search, DeepMind

Kuviot

Sisällys

1 Johdanto

Perinteisesti videopelin hallitseva tekoäly on tarkoittanut tietokoneshakkia. Tietokoneshakilla on pitkä historia, ja shakkialgoritmit ovat päihittäneet keskivertoiset ihmispelaajat jo vuosikymmenien ajan. Tämän päivän videopelit ovat kuitenkin yhä monimutkaisempia, ja muodostavat dynaamisia ympäristöjä, joiden mekaniikoista ei ole täsmällistä tietoa. Vaikka tietokoneen on vaikeampi navigoida tällaisissa ympäristöissä, myös monimutkaisiin peleihin on olemassa tehokkaita algoritmeja. Näiden ohjelmien algoritmit, samoin kuin historian ja nykypäivän shakkikoneiden algoritmit, hallitsevat täydellisesti vain yhden pelin, johon ne on erikoistettu.

Tekoälytutkimukseen keskittyvä yhtiö DeepMind julkaisi MuZero-nimisen tietokoneohjelman loppuvuodesta 2019. Ohjelman tarkoituksena on oppia pelaamaan erilaisia pelejä tuntematta niiden sääntöjä entuudestaan, käyttäen apunaan suunnittelussa koneoppimisen tekniikoita. MuZero pyrkii kohti yleiskäyttöistä algoritmia, joka voisi potentiaalisesti pelata mitä tahansa peliä shakista konsolipeleihin.

Suunnitteluominaisuuksilla varustetuilla tekoälyillä on tavallisesti valmiina tiedot pelin ympäristöstä ja säännöistä. Tällöin toimintojen suunnittelu tapahtuu aiemman tiedon perusteella, käyttäen apuna esimerkiksi puupohjaisia hakualgoritmeja optimaalisen siirron löytämiseksi. Puuhakuun perustuvat menetelmät ovat osoittautuneet tehokkaiksi shakin ja gon kaltaisissa peleissä, joissa on käytettävissä simulaattori, toisin sanoen jokaisen tulevan siirron seuraukset ovat laskettavissa (Schrittwieser ym. 2020). Nämä suunnittelualgoritmit kuitenkin tukeutuvat vahvasti tietoihin ympäristön dynamiikasta, eivätkä ole kovin helposti yleistettävissä muihin ongelmiin, ainakaan ilman suuria muutoksia. Mikäli ympäristön dynamiikka ei ole tiedossa, vaihtoehtoiset koneoppimisalgoritmit opettelevat ensin mallin ympäristöstä, kyetäkseen suunnitteluun sen pohjalta. Useimmat mallipohjaiset suunnittelualgoritmit eivät tosin selviydy visuaalisesti monimutkaisissa ympäristöissä, joissa suosituimmat menetelmät ovat perustuneet mallivapaaseen oppimiseen. Mallivapaat algoritmit arvioivat optimaalisen toimintatavan suoraan vuorovaikutuksesta ympäristön kanssa (Schrittwieser ym. 2020), mutta toimivat puolestaan heikommin peleissä, jotka vaativat hienostunutta arviointia, kuten shakki ja go.

Suunnittelualgoritmien rakentaminen onkin jo pitkään ollut yksi suurimmista haasteista tekoälyjen kehityksessä. MuZeron algoritmi selviytyy sekä klassisista lautapeleistä - shakista, gosta ja shogista - että visuaalisesti monimutkaisista Atari 2600 -peleistä yhdistelemällä menetelmiä uudella tavalla. Sen sijaan, että ohjelma pyrkisi koko ympäristön tarkkaan mallintamiseen, MuZero keskittyy vain kaikkein merkityksellisimpiin ominaisuuksiin. Algoritmi opettelee niiden pohjalta mallin pelin dynamiikoista, ja oppii ymmärtämään siirtojensa syy-seuraussuhteet.

Tässä tutkielmassa tullaan selvittämään, mitä tarkoittaa mallipohjainen vahvistusoppiminen sekä syväoppiminen, ja perehdytään siihen, kuinka mallipohjaista vahvistusoppimista hyödynnetään MuZeron algoritmissa. Lopuksi käydään vielä yksityiskohtaisesti läpi MuZeron algoritmin toimintamekanismi.

2 MuZeron toiminnan perusteet

Ennen 2010-lukua suurin osa peliympäristössä toteutettavasta tekoälytutkimuksesta tapahtui klassisten lautapelien, etenkin shakin parissa. Shakki on sopinut hyvin tekoälyn testialustaksi, sillä peli on helppokäyttöinen, tuttu ja suhteellisen yksinkertainen (Ensmenger 2011). Tutkimuksien tavoitteena saattoi esimerkiksi olla tehokkaimman mahdollisen järjestelmän luominen, jolla peliteoriaa ymmärrettäisiin paremmin (Risi ja Preuss 2020). Nykyään videopelejä hallitsevista tekoälyistä ollaan kiinnostuneita paljon monipuolisemmin. Muutos on ollut nopeaa, ja sitä ovat vauhdittaneet DeepMindin kaltaiset suuret toimijat. Kone- ja vahvistusoppimista sovelletaan yhä enemmän, esimerkiksi vuonna 2019 julkaistu OpenAI Five -niminen vahvistusoppimiseen perustuva tekoäly oppi pelaamaan Dota 2 -strategiapeliä jopa ammattipelaajia paremmin (Berner ym. 2019).

Pelimaailma tarjoaa tekoälyn potentiaalin testaamiseen monipuolisen alustan, jossa ohjelman valmiuksia suoriutua ympäristön tehtävistä voidaan kehittää. Peliympäristöjä on mahdollista rakentaa hyvin vaihtelevilla ominaisuuksilla; deterministisillä ja diskreeteillä, kuten klassisissa lauta- ja korttipeleissä, tai epädeterministisillä ja jatkuvilla, kuten monissa moderneissa videopeleissä (Bowling ym. 2006). Yleispelaaminen (eng. general game playing, GGP) on eräs tekoälysuunnittelun alalaji. Yleispelaamisessa tavoitteena on luoda useaa erilaista peliä tehokkaasti pelaava tekoäly, jonka oppiminen tapahtuu ilman ihmisen väliintuloa. Toisin kuin erikoistuneet tekoälyt, MuZeron kaltaiset yleiskäyttöiset tekoälyt eivät luota tiettyä peliä varten etukäteen luotuihin algoritmeihin (Genesereth, Love ja Pell 2005). Tällainen itseään korjaava ja itsenäisesti oppiva algoritmi saadaan tavallisesti aikaan koneoppimista hyödyntäen.

2.1 Koneoppiminen

Koneoppiminen on tekoälyn alalaji, jossa algoritmi parantaa suoritustaan kokemuksen sekä tiedon avulla, ja tekoälyn oppiminen tapahtuu jotakin esimerkkidataa käyttäen. Koneoppimisen yleisimmät tekniikat ovat ohjattu-, ohjaamaton-, sekä vahvistusoppiminen. Ohjatussa oppimisessa (eng. supervised learning) opetusaineiston tarjoaa ulkoinen, asiantunteva ohjaa-

ja. Opetusaineiston avulla koneoppimismalli pyrkii muodostamaan funktion, jolla sekalainen aineisto voidaan luokitella. Ohjaamattomassa oppimisessa (eng. unsupervised learning) mallin tavoitteena on löytää piilotettuja rakenteita luokittelemattomasta datasta (Sutton ja Barto 2018). Ohjaamattoman oppimisen algoritmeja voidaan käyttää esimerkiksi samankaltaisten tietojen klusterointiin, tietojen tiivistämiseen, tai uuden, alkuperäistä dataa muistuttavan synteettisen datan luomiseen (Justesen ym. 2020).

2.1.1 Vahvistusoppiminen

Vahvistusoppiminen (eng. reinforcement learning, RL) on koneoppimisen ongelmanratkaisutekniikka, josta puhutaan toisinaan myös trial and error -oppimisena. Tekoälyagentti saa ympäristössä suorittamiensa toimintojen perusteella joko negatiivista tai positiivista palautetta. Vahvistusoppimisessa pyritään löytämään sellainen toimintatapa, jolla maksimoidaan numeerinen palkintosignaali. Agentille ei kerrota, mitkä toiminnot sen tulee tehdä, vaan sen tulee itse kokeilun ja erehdyksen kautta löytää sopiva toimintamalli positiivisen palautteen maksimoimiseksi (Sutton ja Barto 2018).

Videopeliympäristössä vahvistusoppimiseen perustuva agentti oppii oikeanlaisen toimintamallin vuorovaikuttaen peliympäristön kanssa. Videopeli voidaan helposti mallintaa vahvistusoppimiseen soveltuvaksi ympäristöksi. Pelaajista mallinnetaan agentteja, joilla on rajallinen määrä erilaisia toimintoja, ja palkintosignaali määritetään pelin pisteillä esimerkiksi niin, että pisteiden saaminen edustaa positiivista palautetta, ja niiden menettäminen negatiivista palautetta (Justesen ym. 2020).

2.1.2 Mallipohjainen vahvistusoppiminen

Mallipohjainen vahvistusoppiminen (eng. model-based RL) on vahvistusoppimisen yksi variaatio. Lähestymistapa keskittyy sellaisiin tilanteisiin, joissa on ensin hahmotettava uusi ympäristö, ennen kuin pystytään hallitsemaan siellä olevia tehtäviä; menetelmässä yhdistetään suunnittelu ja vahvistusoppiminen (Moerland, Broekens ja Jonker 2021). Mallipohjainen vahvistusoppiminen tarkoittaa periaatteessa optimaalisen toimintamallin etsimistä, joka onnistuu opettelemalla ensin ympäristömalli. Klassisesti tätä mallia edustaa Markovin pää-

töksentekoprosessi, joka koostuu kahdesta komponentista: tilasiirtymämallista, joka ennustaa seuraavan tilan, ja palkintomallista, joka ennustaa odotetun palkinnon siirtymän aikana (Puterman 1994). Kun malli on rakennettu, algoritmi soveltaa optimaalisen arvofunktion tai toimintamallin laskemiseksi Markovin päätöksentekoprosessin suunnittelualgoritmeja, kuten arvoiteraatiota tai Monte Carlo -puuhakua (Schrittwieser ym. 2020). Mallipohjaisessa vahvistusoppimisessa pyritään siis käsittelemään ongelmia opettelemalla ensin malli ympäristön dynamiikasta, ja suunnittelemalla sitten olemassa olevaan malliin perustuen.

2.1.3 Mallivapaa vahvistusoppiminen

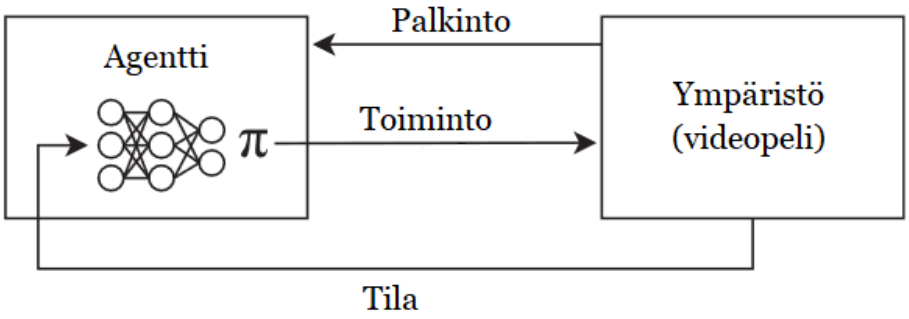
Mallipohjaisen vahvistusoppimisen lisäksi on olemassa mallivapaa vahvistusoppiminen (eng. model-free RL). Mallivapaissa menetelmissä agentit arvioivat optimaalisen toimintamallin ja/tai arvofunktion suoraan vuorovaikutuksesta ympäristön kanssa, käyttämättä lainkaan ympäristömallia (Schrittwieser ym. 2020). Suosittu mallivapaa vahvistusoppimisen menetelmä on Q-oppiminen. Kirjain ”Q” menetelmän nimessä viittaa laatuun (eng. quality), joka kuvaa, kuinka hyödyllinen tietty toiminto olisi tulevan palkinnon saamiseksi. Q-oppimisessa tuleva palkinto lasketaan valitsemalla parhaiten tunnettu tulevaisuuden tila-toiminto-pari (Shixiang ym. 2016). Tällaiset puhtaaseen kokemukseen perustuvat algoritmit luottavat täysin näytteisiin ympäristöstä.

2.2 Syväoppiminen

Syväoppiminen (eng. deep learning) on koneoppimisessa käytettävä menetelmä, jossa hyödynnetään useita käsittelykerroksia, eli neuroverkkoja (LeCun, Bengio ja Hinton 2015). Inspiraatio neuroverkkoihin on alun perin tullut biologisesta tutkimuksesta, erityisesti hermostoista, eli ihmisaivojen tietojenkäsittelymekanismeista (Bishop 1994). Keinotekoinen neuroverkko koostuu myös neuroneista, eli yksittäisistä tiedonkäsittely-yksiköistä, jotka on linkitetty toisiinsa synapseilla, eli painotetuilla kytkennöillä. Neuronit ovat kerroksittain; ensimmäistä kerrosta kutsutaan syötekerrokseksi, ja viimeistä ulostulokerrokseksi, ja näiden kerrosten välissä voi verkosta riippuen olla myös useita piilokerroksia. Neuroverkko vastaanottaa syötteenä sille annettavaa dataa ja käsittelee sen sisäisen algoritminsa mukaisesti (Bishop 1994).

Keinotekoiset neuroverkot (eng. artificial neural networks) ovat siis oppimiseen kykeneviä yleiskäyttöisiä funktioita. Neuroverkkojen yleiskäyttöisen luonteen vuoksi niitä on sovellettu useisiin erilaisiin ongelmiin, mukaan lukien erilaisten pelien pelaamiseen (Justesen ym. 2020). Keinotekoisten neuroverkkojen arkkitehtuurit voidaan jakaa karkeasti kahteen pääryhmään: eteenpäinsyöttäviin sekä takaisinkytkettyihin neuroverkkoihin. Eteenpäinsyöttävät verkot vastaanottavat yhden syötteen, esimerkiksi pelitilan esityksen, ja antavat todennäköisyyksiä kullekin mahdolliselle toiminnolle (Justesen ym. 2020). Takaisinkytkettyjä neuroverkkoja taas käytetään tyypillisesti aikasarjatietojen käsittelyyn, joissa verkon ulostulo riippuu edellisissä askeleissa tapahtuneesta verkon aktivoitumisesta. Koulutusprosessi takaisinkytketyssä verkossa on lähes samanlainen kuin eteenpäinsyöttävässä verkossa, mutta verkon edellinen piilotettu tila syötetäänkin takaisin verkkoon seuraavan syötteen kanssa. Tämä mahdollistaa neuroverkon kontekstitietoisuuden, eli se muistaa myös edelliset aktivointitiedot, mikä on hyödyllistä esimerkiksi silloin, kun yksittäinen havainto pelistä ei edusta koko pelitilaa (Justesen ym. 2020).

Videopelien pelaamisessa tekoälyllä on tavallista käyttää useaa päällekkäistä konvoluutiokerrosta, joiden jälkeen on lisäksi takaisinkytketyt kerrokset, ja viimeisenä eteenpäinsyöttävät kerrokset (Justesen ym. 2020). Konvoluutioneuroverkot (eng. convolutional neural networks) koostuvat koulutettavista suodattimista, ja soveltuvat hyvin kuvadatan, kuten näytön pikselien, käsittelyyn (O'Shea ja Nash 2015).



Kuvio 1. Syvää neuroverkkoa käytetään toimintamallin etsimiseen (mukailtu: Justesen ym. 2020)

Kuviossa 1 on hahmotelma syvään vahvistusoppimiseen pohjautuvasta ohjelmasta, joka laskee optimaalisia toimintoja ympäristöstä vastaanotettujen havaintojen, sekä saatujen palkintojen avulla. Kuvion ohjelmassa agentin noudattama toimintamalli määritetään käyttäen syvää neuroverkkoa. Ympäristön tila, tai vaihtoehtoisesti agentin vastaanottama havainto (esim. näytön pikselit), annetaan syötteenä agentin toimintamalliverkolle. Ympäristössä suoritettava toiminto satunnaistetaan toimintamalliverkon avulla saadusta tuotteesta, jota merkitään kuviossa π:llä. Vahvistusoppimisalgoritmi päivittää toimintamallinsa ympäristöstä saatavan palkinnon perusteella, ja tavoitteena on maksimoida kumulatiiviset palkinnot (Justesen ym. 2020). Tässä kuviossa esitetty toiminnallisuus on pitkälti myös MuZeron toimintamekanismi, johon syvennytään tarkemmin luvussa 4.

3 Mallipohjainen vahvistusoppiminen MuZerossa

Mallipohjainen vahvistusoppiminen, jossa palkinnot ajavat tekoälyagenttia kohti tavoitteita, on MuZeron ytimessä oleva tekniikka. Tämän luvun tarkoituksena on tarkemmin keskittyä siihen, kuinka MuZero hyödyntää mallipohjaista vahvistusoppimista, ja onnistuu sen avulla löytämään ratkaisut myös sellaisiin ongelmiin, joita tavallisesti lähdettäisiin ratkaisemaan mallivapaasti. MuZero oppii pelaamaan ja suunnittelemaan siirtoja mallinsa avulla sekä Atarin pikselisyötteen, että lautapelien tilasyötteen perusteella.

MuZeron voisi sanoa olevan jatkoa DeepMindin aikaisemmalle tekoälylle, vuonna 2017 esitellylle AlphaZerolle. AlphaZero on tietokoneohjelma, joka oppi pelaamaan shakkia, gota sekä shogia yksinkertaisesti pelaamalla itseään vastaan (Silver ym. 2018). MuZero käyttää AlphaZeron kaltaista puuhakua sekä toimintamallin iterointialgoritmia, mutta sisällyttää harjoitteluprosessiinsa opitun mallin. Vaikka MuZero pitkälti rakentuukin AlphaZeron päälle, on niiden algoritmeissa ja toiminnoissa muutamia merkittäviä eroja, jotka mahdollistavat MuZeron yleiskäyttöisemmän luonteen.

AlphaZeron suunnitteluprosessissa käytetään simulaattoria, joka tuntee pelien (shakki, go ja shogi) säännöt, ja neuroverkko ennustaa niiden perusteella sijaintien toimintamallin ja arvon. Pelin sääntöjen täydellistä tuntemusta käytetään myös mm. mallinnettaessa hakupuun tilasiirtymiä (Silver ym. 2018). MuZerolla ei ole pääsyä pelien sääntöihin, jotka sen täytyy oppia itsenäisesti harjoittelemalla, aloittaen tyhjästä. AlphaZerolla on lisäksi koko ajan vain yksi malli pelille (pelilaudan tilasta ennusteisiin), kun taas MuZero käyttää useampaa mallia yhtä aikaa: esitys pelin senhetkisestä tilanteesta, esitys tilojen dynamiikasta (eli kuinka jokin toiminto muuttaa laudan tilojen esityksiä), sekä ennustus tulevaisuuden sijainnin toimintamalleista ja arvosta. AlphaZeron käyttämä algoritmi on täydellinen shakin, shogin ja gon tapauksessa, mutta MuZero pystyy oppimaan näiden lisäksi myös Atari-pelien ympäristön vaikeasti ennustettavan dynamiikan. Yksi suurimmista eroista AlphaZeroon nähden onkin opittu ympäristömalli.

3.1 Opittu ympäristömalli

Datan avulla opittava ympäristömalli on olennainen osa vahvistusoppimiseen perustuvien tekoälyagenttien toteutusta (Grimm ym. 2020). Käsitellään aivan ensimmäiseksi dynamiikkamallin avulla, mitä mallipohjaisessa vahvistusoppimisessa varsinaisesti tarkoitetaan mallilla. Agentti vastaanottaa suoritetun siirron seurauksena saadun datan, jolloin mallipohjaisessa vahvistusoppimisessa ollaan kiinnostuneita varsinkin seuraavanlaisesta dynamiikkafunktiosta:

$$(s_t, a_t) \rightarrow s_{t+1}.$$

Tämä on eteenpäin välittävä malli (eng. forward model), joka ennakoi aina ympäristön seuraavan tilan s_{t+1}, joka saadaan nykyisen tilan s_t, sekä valitun toiminnon a_t perusteella. Eteenpäin välittävä malli on vahvistusoppimisessa ylivoimaisesti yleisin mallityyppi, jota käytetään, kun suunnitellaan tulevaisuuteen (Moerland, Broekens ja Jonker 2021). Muihin tehtäviin tarkoitettuja, samaisesta datasta mahdollisesti saatavia malleja ovat taaksepäin välittävä malli (eng. backward model) sekä käänteinen malli (eng. inverse model); (Moerland, Broekens ja Jonker 2021). MuZerossa käytettävä malli eroaa perinteisestä eteenpäin välittävästä mallista jonkin verran. Joissain tilanteissa dynamiikan oppiminen voi olla monimutkaisempaa, kuin sijainnin arvon ennustamisen kannalta merkityksellisten dynamiikkojen oppiminen. Tällaisissa tilanteissa ns. arvoa vastaavien mallien (eng. value equivalent models) käyttäminen saattaa olla tehokkaampaa, kuin eteenpäin välittävän mallin käyttäminen (Grimm ym. 2020). Arvoa vastaavissa malleissa ennustetaan siis tulevaisuuden sijainnin arvo tulevaisuuden tilan sijaan.

3.1.1 Mallin soveltamisen hyödyt

Varsinkin visuaalisesti monimutkaisissa ympäristöissä tehokkaan ja yleiskäyttöisen mallipohjaiseen oppimiseen perustuvan tekoälyn tulee kyetä sekä mallintamaan ennalta tuntematon toimintaympäristö, että soveltamaan tätä mallia, kun suunnitellaan optimaalisia toimintoja. Mallipohjaisessa vahvistusoppimisessa on yleensä keskitytty havainnointivirran suoraan mallintamiseen pikselitasolla. Näin tarkka mallinnus on kuitenkin osoittautunut laskennallisesti raskaaksi varsinkin suurissa ympäristöissä (Hafner ym. 2019). Hienostuneeseen suunnitteluun soveltuvien mallien rakentaminen on ollut pitkäaikainen haaste, jossa malliva-

paat menetelmät ovat menestyneet paremmin. Yleisimpiä vaikeuksia ovat olleet mm. mallien epätarkkuudet, sekä monivaiheisissa ennusteissa tapahtuva virheiden kasaantuminen (Hafner ym. 2019).

Mallipohjaisessa suunnittelussa on kuitenkin paljon etuja verrattuna mallivapaisiin lähestymistapoihin. Opittu malli on potentiaalisesti tehtävästä riippumaton, ja sopeutuu siksi sujuvammin myös muihin ympäristön tehtäviin (Hafner ym. 2019). Mallin tuoma kyky ennustaa tulevaisuuteen kannustaa agenttia myös ympäristön tutkimiseen, tärkeiden yksityiskohtien huomioimiseen sekä sisäisen motivaation syntymiseen (Kaiser ym. 2020). Lisäksi mallipohjaisen algoritmin suorituskykyä voidaan kasvattaa yksinkertaisesti lisäämällä laskentatehoa, kun etsitään optimaalista toimintoa (Silver ym. 2017), eikä vuorovaikutusta ympäristön kanssa tarvita niin valtavia määriä. Mallivapaissa algoritmeissa vaadittujen askelten määrä voi kohota kymmeniin tai satoihin miljooniin, kun taas mallipohjaisessa oppimisessa huomattavasti vähempi riittää ratkaisun löytymiseen (Kaiser ym. 2020). Esimerkiksi MuZero saavutti AlphaZeron tason shakissa vain miljoonan askeleen jälkeen.

3.1.2 Ympäristön mallintaminen MuZerossa

MuZero käyttää erilaista lähestymistapaa mallipohjaiseen suunnitteluun ylittääkseen aiempien menetelmien rajoitukset ja haittapuolet. Ohjelmalla ei edes yritetä jatkuvasti mallintaa koko peliympäristöä pikseli kerrallaan, vaan algoritmi mallintaa ympäristöstä vain ne asiat, jotka ovat välttämättömiä päätöksentekoprosessille (Schrittwieser ym. 2020). Esimerkiksi sen sijaan, että keskityttäisiin vastustajan jokaisen pikselin mallintamiseen, keskitytäänkin oppimaan sen iskujen väistäminen. MuZero mallintaa ympäristöstä kolme suunnittelun kannalta kriittistä elementtiä, jotka ovat arvo, toimintamalli sekä palkinto. Arvon ennustamisella saadaan tietää, kuinka hyvä nykyinen sijainti on. Käytännössä tämä tarkoittaa ennustetta siitä, kuinka todennäköisesti agentti voittaa kyseisestä sijainnista, esimerkiksi tietystä shakkiruudusta. Toimintamalli (eng. policy) edustaa arviota siitä, mikä toiminto olisi paras kyseisessä sijainnissa. Palkinto kertoo, kuinka hyvä viimeisin suoritettu toiminto oli.

MuZero oppii kaiken tämän tiedon peliympäristöstä syvällä neuroverkolla, ja se on samalla kaikki tieto, jonka tekoäly tarvitsee ymmärtääkseen pelin säännöt ja dynamiikan tehokasta

suunnittelua varten (Schrittwieser ym. 2020).

3.2 MuZeron tehokkuus

MuZeron käyttämä algoritmi saavutti huipputason suorituskyvyn, kun sitä testattiin 57:ssä eri Atari-pelissä. Avoimen lähdekoodin Arcade -oppimisympäristö (ALE) on yleinen alusta monimutkaisemmissa videopeliympäristöissä toteutettavalle tekoälytekniikoiden testaamiselle. Oppimisympäristö rakentuu Atari 2600-emulaattorin päälle, ja agentille voidaan antaa siellä syötteenä pelin pisteet, ruudun pikselit sekä RAM-muisti. Atari-pelit tarjoavat sopivasti erilaisia haasteita, jotka pakottavat tekoälyn keksimään jatkuvasti uusia strategioita niistä selviytyäkseen. Suurin osa peleistä vaatii nopeaa reaktiokykyä sekä tarkkaa ajoitusta. Osa peleistä, kuten Montezuma's Revenge, edellyttävät puolestaan pitkäaikaista suunnittelua ja toistaiseksi näkymättömissä olevien pelitilojen muistamista. Joissakin peleissä on siis epätäydellistä tietoa ja stokastisuutta, kun taas toiset ovat hyvinkin deterministisiä (Justesen ym. 2020). Peleissä on myös selkeä menestysmittari, eli pisteet, joita vastaan harjoitella. MuZeron algoritmi ylitti ihmispelaajien taidot lähes kaikissa peleissä moninkertaisesti, ja osassa peleistä monin kymmen-, tai jopa monin satakertaisesti (Schrittwieser ym. 2020). Lisäksi MuZero osoitti olevansa olemassa olevista Atari-algoritmeista tehokkain suurimmassa osassa testatuista peleistä, kuten Space invaders- sekä Ms. Pacman -peleissä.

MuZeroa arvioitiin myös klassisissa lautapeleissä: shakissa, gossa ja shogissa. MuZeron algoritmi saavutti kaikissa kolmessa vähintään yhtä hyvän tason, kuin DeepMindin vuonna 2017 julkaisema AlphaZero, joka tunsi pelien säännöt, ja käytti apunaan suunnittelussa simulaattoria. Gossa MuZero ylitti hienoisesti AlphaZeron suorituskyvyn, vaikka hakupuussa käytettiin vähemmän laskelmia solmua kohden. DeepMindin mukaan tämä viittaa siihen, että MuZero saattaa välimuistittaa laskentaansa hakupuuhun, ja käyttää dynamiikkamallin jokaista lisäsovellusta saadakseen syvemmän käsityksen asemastaan (Schrittwieser ym. 2020).

3.3 Haasteita

MuZeron algoritmi kykeni oppimaan ja päihittämään suurimman osan testatuista Atari -peleistä (Schrittwieser ym. 2020). Muutamassa pelissä MuZeron suoritus jäi kuitenkin pait-

si parhaita Atari-algoritmeja, kuten mallivapaata R2D2:a, myös keskivertoista ihmispelaajaa huonommaksi. Näin tapahtui varsinkin Pitfall- ja Montezuma's Revenge -peleissä, joista molemmista MuZero sai tulokseksi pyöreän nollan.

Erityisesti Montezuma's Revengeä pidetään vaikeana ongelmana vahvistusoppimiseen pohjautuville agenteille, sillä se vaatii useiden pelinsisäisten taitojen yhdistämistä palkintojen löytämiseksi. Positiivista palautetta tuottavat palkinnot ovat satojen askelien päässä toisistaan jopa optimaalisen pelin aikana, ja tämän takia peli vaatii agentilta erityisen pitkäjänteistä suunnittelua (Burda ym. 2018). Vahvistusoppimisessa tekoälyagentti luottaa ympäristöltä saatuihin palkintosignaaleihin. Tavallisesti pelit antavat palkintoja onnistuneista strategioista, ja juuri sen vuoksi vahvistusoppiminen ja pelit sopivat hyvin yhteen. Siksi sellaiset pelit, joissa ei ole selkeää palkintomallia, ovat vaikeita niihin luottaville algoritmeille. Kun palkinnot ovat harvassa, haasteena on määrittää, kuinka palkinnon saamiseen lopulta johtaneet lukuisat toimet pystytään hyvittämään (Justesen ym. 2020).

Montezuma's Revengen kaltaisissa peleissä tulosten syntyminen on tällaisten tekijöiden vuoksi ollut hidasta, ja ne ovat edelleen vaikeita tai mahdottomia jopa parhaimmille vahvistusoppimiseen perustuville tekoälyille, sillä uteliaisuutta on vaikea opettaa. Merkittävää edistystä on saavutettu tosin menetelmillä, joilla on pääsy esimerkiksi pelin taustalla olevaan emulaattoritilaan (Tang ym. 2017). Montezuma's Revengen perustavanlaatuinen idea on hyvin erilainen verrattuna esimerkiksi shakkiin tai gohon, joissa pelaaja saa toimintojensa seurauksena lähes välittömästi joko negatiivista tai positiivista palautetta, ja joissa pelaaja käytännössä pakotetaan toimimaan vastustajan liikkeiden motivoimana.

4 MuZeron algoritmin toiminta

MuZeron algoritmin perusideana on pelata itseään vastaan, tallentaa pelatut pelit, ja harjoitella tallennettujen pelien dataa apuna käyttäen. Algoritmissa yhdistetään opittu malli sekä puuhaku. MuZero käyttää optimaalisen siirron valitsemiseen Monte Carlo -puuhakuna tunnettua menetelmää. Menetelmässä seuraavan siirron valitsemiseksi simuloidaan todennäköisiä tulevaisuuden skenaarioita senhetkisestä sijainnista, arvioidaan niiden arvo neuroverkon avulla, ja lopuksi valitaan toiminto, joka maksimoi tulevaisuuden odotetun arvon. Opittua mallia sovelletaan suunnittelussa sekä ympäristössä etenemisessä, ja MuZeron malli on optimoitu niin, että simuloidussa ympäristössä hyvin toimivat strategiat toimivat hyvin myös todellisessa ympäristössä.

Siinä, missä tekoälyagentti normaalisti tuntisi pelin säännöt, MuZeron täytyy ensin luoda itselleen oma kuviteltu peli, jota se voi käyttää apuna suunnitellessaan seuraavia siirtoja. MuZero ei siis ymmärrä pelin sääntöjen lisäksi myöskään sitä, kuinka jokin toiminto vaikuttaa pelitilaan. Tästä syystä MuZero ei voi suoraan simuloida tulevaisuuden skenaarioita Monte Carlo -puuhaussa; lisäksi tarvitaan yhden sijaan kolme neuroverkkoa ennustamista, dynamiikkaa sekä esitystä varten.

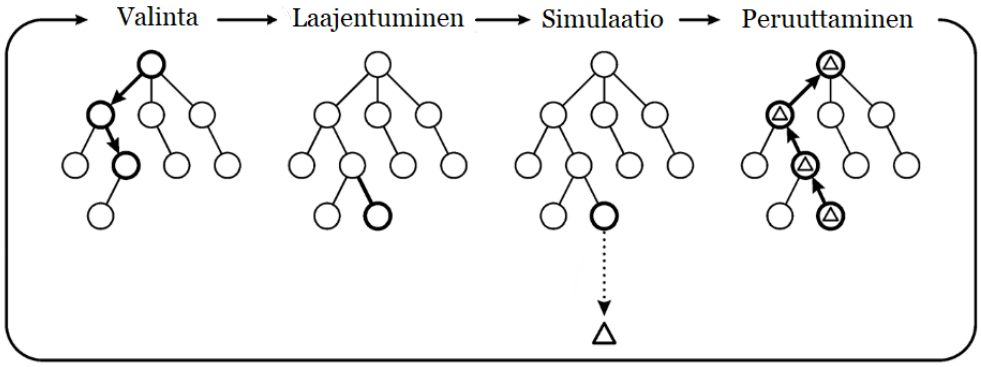
Tässä luvussa tullaan tarkemmin avaamaan Monte Carlo -puuhaun perusprosessia ja toimintaa MuZerossa. Lopuksi käsitellään MuZeron algoritmin toimintaperiaate pääpirteittäin kolmen kuvion avulla; mallin avulla suunnittelu, agentin toiminta ympäristössä, sekä mallin kouluttaminen.

4.1 Monte Carlo -puuhaku

Monte Carlo -puuhaku on menetelmä optimaalisen siirron löytämiseksi. Menetelmää on aikaisemminkin käytetty mm. tietokoneshakissa ja -gossa ennustamaan polkuja, joita agentin tulisi noudattaa voittoon johtavan ratkaisun saavuttamiseksi. Polkujen ennustaminen tapahtuu ottamalla satunnaisnäytteitä, ja rakentamalla hakupuu saatujen tulosten mukaisesti (Browne ym. 2012). MuZero käyttää parhaan siirron löytämiseksi Monte Carlo -puuhakua. MuZeron algoritmi pysähtyy pelitilan muuttuessa aina uuden havainnon saatuaan laskemaan,

mikä toiminto tulisi valita seuraavaksi. Monte Carlo -puuhakua hyödynnetään lisäksi pelattujen siirtojen tallentamiseen, sillä aikaisemmat pelit ovat tärkeä osa MuZeron oppimisprosessissa.

Monte Carlo -puuhaun keskeinen ajatus on simuloida tuhansia satunnaisia pelejä. Uudet sijainnit lisätään hakupuuhun, jossa jokainen solmu sisältää ennusteen siitä, kuka voittaa kyseisestä sijainnista (Gelly ja Silver 2011). Nämä ennusteet päivitetään Monte Carlo -simulaatiolla; solmun arvo on kaikkien kyseisessä sijainnissa vierailevien simuloitujen pelien keskimääräinen tulos. Hakupuuta käytetään ohjaamaan simulaatioita pitkin lupaavia polkuja, valitsemalla sellaisia lapsisolmuja, joilla on suurin potentiaalinen arvo. Tämä johtaa todella valikoivaan hakuun, joka tunnistaa nopeasti hyvät siirtosekvenssit (Gelly ja Silver 2011). Monte Carlo -puuhaun perusprosessi on käsitteellisesti melko yksinkertainen. Prosessin neljään päävaiheeseen kuuluvat valinta, laajentuminen, simulaatio sekä peruuttaminen. Päävaiheet toistetaan järjestyksessä niin monta kertaa, että optimaalinen siirto on löytynyt.



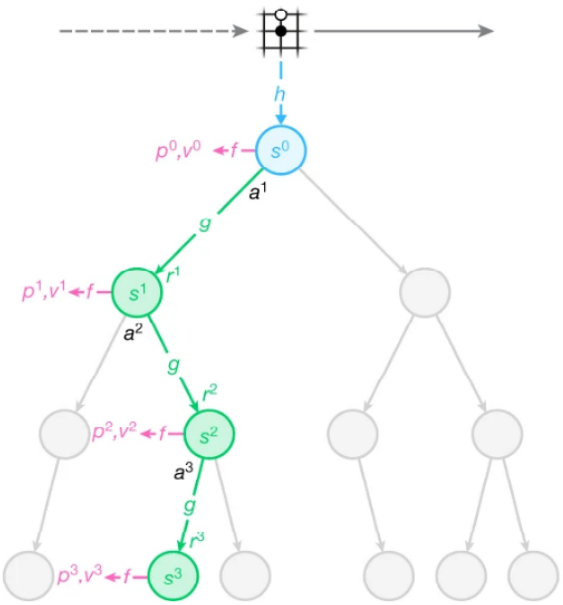
Kuvio 2. Monte Carlo -puuahun perusprosessi (mukailtu Browne ym. 2012)

Puuhaun valintavaiheessa edetään juurisolmusta alkaen, valiten aina toimintoja, joiden arvioitu arvo on korkein. MuZero laskeutuu rekursiivisesti puun läpi, kunnes saavutetaan lehtisolmu. Lehtisolmun vanhempaan kutsutaan algoritmin tässä vaiheessa funktiota, jolla saadaan ennustettua palkinto, uusi piilotettu tila, sekä uuden piilotetun tilan toimintamalli ja arvo. Laajennusvaiheessa lehtisolmu laajennetaan luomalla uusia lapsisolmuja, yksi jokaiselle pelin mahdolliselle toiminnolle, ja osoitetaan jokaiselle uudelle solmulle vastaava toimintamalli. Simulaatio suoritetaan syntyneestä solmusta tuloksen saamiseksi, toisin sanoen peli

simuloidaan loppuun. Peruutusvaiheessa simulaation tulos ns. varmuuskopioidaan; ennustettu arvo kuljetetaan puussa samaa reittiä pitkin takaisin ylös, jolloin reitti jää myös talteen (Browne ym. 2012). Kaikki pelin tiedot tallennetaan toistopuskuriin (eng. re-play buffer), ja agentti voi aloittaa uuden pelin.

4.2 Mallin avulla suunnittelu

Tarkastellaan, kuinka MuZero käyttää oppimaansa mallia suunnittelussa. Ohjelma vastaanottaa ensin havainnon, joka on kuva käynnissä olevan pelin senhetkisestä tilanteesta. MuZero käsittelee sitten havaintoa löytääkseen siitä tärkeimmät elementit seuraavan siirron valitsemiseksi. Tietojen avulla opitaan ymmärtämään, mitä seuraa mistäkin toiminnosta, esimerkiksi kuinka go-pelissä kiven siirtäminen toiseen kohtaan vaikuttaa pelitilanteeseen, ja samalla voiton mahdollisuuksiin. MuZero hahmottelee mallinsa avulla lukuisia skenaarioita, joita havainnon aikaisesta sijainnista voisi eri siirtojen seurauksena syntyä. Lopulta monivaiheisen suunnitteluprosessin tuloksena löytyy optimaalinen siirto, eli toiminto. Toiminnon seurauksena puolestaan saadaan uusi havainto muuttuneesta pelitilasta, ja näin peliä pelataan eteenpäin. Käsitellään seuraavaksi yksityiskohtaisemmin algoritmin ensimmäinen vaihe, eli yhden siirron suunnitteluun vaadittava prosessi.



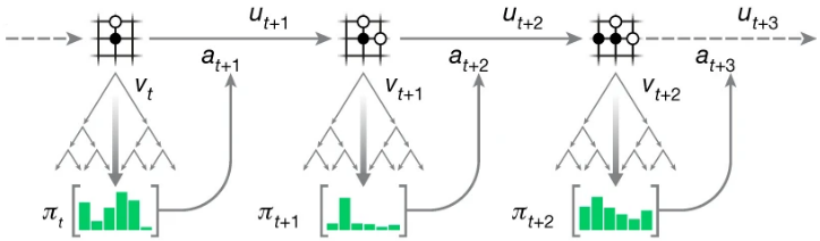
Kuvio 3. MuZero suunnittelee mallin avulla (lähde: Schrittwieser ym. 2020)

Kuviossa 3 h edustaa esitysfunktiota, joka saadaan suoraan pelilaudasta raa'an havainnon muodossa. Kuvion yläreunassa on 2x2 go-pelilauta, joka edustaa saatua havaintoa. Sen tilalla voisi yhtä hyvin olla shakkilauta tai Atarin näyttö. Havainto muunnetaan ensimmäiseksi piilotetuksi tilaksi s^0. MuZero voi piilotetun tilan pohjalta lähteä tulkitsemaan tilannetta "pelaamalla" peliä eteenpäin ilman, että todellinen tilanne pelilaudalla muuttuu vielä. Piilotetun tilan tarkoituksena on yksinkertaisesti kaapata havainnosta kaikki oleellinen tieto suunnittelua varten; ennustamisfunktion f avulla tilasta ennustetaan sekä arvofunktio v että toimintamalli p (Schrittwieser ym. 2020). Arvofunktio kertoo, kuinka suuren määrän palkintoja tilassa s saisi tulevaisuudessa. Toimintamallin tarkoituksena taas on antaa arvio siitä, kuinka agentti toimisi tilassa s. Toimintamallin tuottaman epätäydellisen, mutta riittävän hyvän tulevaisuuden arvion ansiosta MuZeron ei tarvitse tarkistaa jokaista mahdollista tulevaisuuden toimintoa puuhaulla, ja tällä vältetään päättymätön rekursio. Toimintamallia käytetään simuloidun toiminnon a generoimiseen. Tässä vaiheessa piilotetun tilan ja toiminnon avulla voidaan ennustaa seuraava piilotettu tila, sekä mahdollinen välitön palkinto r dynamiikkafunktiolla g. Välitön palkinto voi olla esimerkiksi pisteet, jotka on saatu pelaamalla siirto. Alkuperäinen piilotettu tila saadaan siirtämällä aikaisemmat havainnot esitysfunktioon h (Schrittwieser ym. 2020).

4.3 Ympäristössä toimiminen

Seuraavaksi käydään läpi, kuinka agentti toimii ympäristössä, eli suorittaa valitsemiaan siirtoja ja vastaanottaa uusia havaintoja. Suunnitteluvaiheessa muodostetun piilotetun tilan ei tarvitse kerätä kaikkia alkuperäisen havainnon rekonstruoimiseksi tarvittavia tietoja. Tämä vähentää huomattavasti sen tiedon määrää, jota mallin täytyy ylläpitää ja ennustaa. Piilotetun tilan ei edes odoteta vastaavan ympäristön todellista tilaa, eikä muitakaan tilan semantiikan rajoituksia. Sen sijaan piilotettu tila voi vapaasti edustaa mitä tahansa tilaa, joka arvioi oikein tomintamallin, arvofunktion sekä palkinnon. Agentti keksii intuitiivisesti kaiken tärkeän dynamiikan, joka johtaa tarkkaan suunnitteluun (Schrittwieser ym. 2020).

Jokaisessa kuviossa 4 esitetyssä askeleessa suoritetaan luvussa 4.1. kuvatun kaltainen Monte Carlo -puuhaku. Uusiin piilotettuihin tiloihin suoritetaan samat ennustukset, joiden kautta saadaan tietää seuraava generoitu toiminto, ja uusi piilotettu tila. Lopulta useiden piilotet-

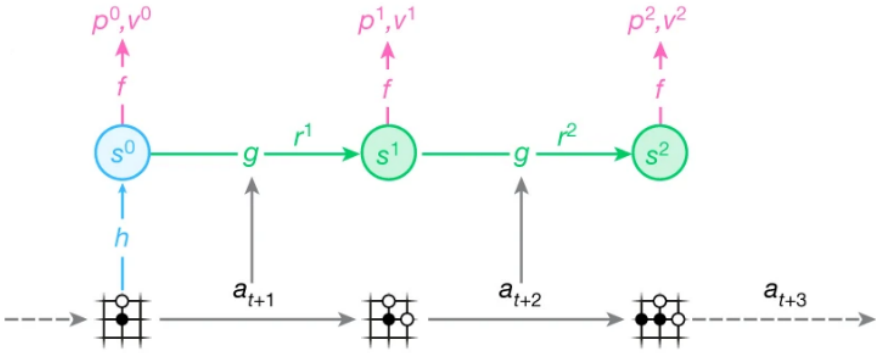


Kuvio 4. MuZeron toiminta ympäristössä (lähde: Schrittwieser ym. 2020)

tujen tilojen muodostaman ketjun tuloksena saadaan aikaan toimintajakauma, ja toimintamalliverkkoa koulutetaan jäljittelemällä jakaumaa. Seuraava pelilaudalla tapahtuva toiminto a_{t+1} satunnaistetaan tästä hakukäytännöksi kutsutusta osiosta π_t, joka on verrannollinen juurisolmun kunkin toiminnon vierailujen määrään. Ympäristö vastaanottaa tämän perusteella valitun toiminnon, ja tuottaa uuden havainnon o_{t+1} sekä palkinnon u_{t+1} (Schrittwieser ym. 2020). Jakson lopussa reittitiedot tallennetaan toistopuskuriin.

4.4 Mallin kouluttaminen

Agentti käyttää keräämäänsä kokemusta vuorovaikutuksessa ympäristön kanssa neuroverkkojensa kouluttamiseen, käyttämällä toistopuskuriin varastoitua dataa aikaisemmista peleistä. Kerätty kokemus sisältää sekä havainnot ja palkinnot ympäristöstä, että tulokset suoritetuista hauista, kun päätetään parhaasta toiminnosta.



Kuvio 5. MuZero kouluttaa malliaan (lähde: Schrittwieser ym. 2020)

Kuvio 5 on esitys siitä, kuinka MuZero kouluttaa malliaan. Reitti satunnaistetaan toistopuskurista. Ensimmäisessä askeleessa esitysfunktio h vastaanottaa syötteenä aikaisemmat havainnot valitulta reitiltä. Kussakin askeleessa dynamiikkafunktio g vastaanottaa syötteenä piilotetun tilan s edellisestä askeleesta ja suoritettavan toiminnon a_{t+1}. Esitys-, dynamiikka-, ja ennustamisfunktioiden parametrit koulutetaan yhteisesti end-to-end, jotta saadaan ennustettua kolme suuretta: toimintamalli p, arvofunktio v, sekä palkinto r (Schrittwieser ym. 2020). Piilotetusta tilasta saadun toimintamallin tulisi siis olla mahdollisimman samanlainen, kuin todellisen toiminnon, joka suoritettiin havainnossa, ja joka johti kyseiseen piilotettuun tilaan.

Tällä lähestymistavalla on DeepMindin mukaan etuna se, että MuZerolla on potentiaali toistuvasti käyttää oppimaansa mallia parantaakseen suunnitteluaan sen sijaan, että agentti keräisi aina uutta tietoa ympäristöstä. Atari-peleissä tehdyissä testeissä *MuZero Reanalyze* -nimellä tunnettu variaatio käytti opittua mallia suunnitellakseen aikaisempien jaksojen toimia uudelleen, parantaen suoritustaan (Schrittwieser ym. 2020).

5 Yhteenveto

Tietokone on onnistunut ylittämään ihmispelaajien kyvyt jo vuonna 1996, kun IBM:n Deep Blue -tietokone päihitti shakin suurmestari Gerry Kasparovin kuusipelisen ottelun ensimmäisessä pelissä. Deep Bluen tehokkuus perustui kuitenkin pitkälti sen massiiviseen laskentatehoon. Teknologiat ovat kehittyneet ja hienostuneet Deep Bluen jälkeen huomattavasti. Shakkikoneet ovat kohonneet yli-inhimilliselle tasolle mm. Stockfish ja AlphaZero -ohjelmien myötä. Nykypäivän tekoälyn kehityksessä vahvistusoppimisesta on tullut yksi suosituimmista lähestymistavoista myös muiden strategiapelien parissa (Risi ja Preuss 2020). Vuonna 2018 vahvistusoppimiseen perustuva AlphaStar -ohjelma löi ammattipelaajat reaaliaikaisessa strategiavideopeli StarCraft II:ssa, joka on shakkia monimutkasisempi peli ympäristöltään.

Nyt tietokoneohjelmalla on kyky itse oppia pelaamaan shakkia, gota, shogia, sekä kymmeniä eri Atari 2600-pelejä. Tietokoneohjelma MuZeron algoritmin taustalla on koneoppimisen ongelmanratkaisutekniikka, mallipohjainen vahvistusoppiminen. MuZero yhdistää toiminnassaan mallipohjaisen vahvistusoppimisen ja -suunnittelun, sekä Monte Carlo- puuhaun, saavuttaen kyvyn suoriutua keskenään hyvin erilaisista peleistä tuntematta niiden sääntöjä ennalta. MuZeron algoritmi opettelee mallin peliympäristöstä, ja ennustaa iteratiivisesti pelin merkityksellisimmät suureet, eli arvon, toimintamallin ja palkinnon. Pohjimmiltaan MuZero siis vastaanottaa havaintoja - kuvia pelilaudasta tai Atarin näytöstä - ja muuntaa ne piilotetuksi tilaksi. Tämä piilotettu tila päivitetään iteratiivisesti prosessilla, joka vastaanottaa edellisen tilan ja hypoteettisen seuraavan toiminnon (Schrittwieser ym. 2020). Malli ennustaa jokaisessa vaiheessa toimintamallin, arvofunktion, sekä välittömän palkinnon.

DeepMindin mukaan MuZero on merkittävä edistysaskel matkalla kohti yleiskäyttöisiä algoritmeja. MuZero onkin suuri edistysaskel verrattuna AlphaZeroon, DeepMindin vuonna 2017 julkaisemaan shakkia, gota, ja shogia pelaavaan ohjelmaan, joka puolestaan oli edistysaskel verrattuna 2015 julkaistuun AlphaGo -ohjelmaan. DeepMind suunnittelee soveltavansa MuZeron tehokkuuden taustalla olevia oppimis- ja suunnittelualgoritmeja robotiikassa, sekä muissa reaalimaailman monimutkaisissa ympäristöissä, joiden pelisääntöjä ei vielä tunneta.

Lähteet

Berner, Christopher, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi ym. 2019. *Dota 2 with Large Scale Deep Reinforcement Learning*. arXiv: 1912.06680 [cs.LG].

Bishop, C. 1994. "Neural networks and their applications". *Review of Scientific Instruments* 65:1803–1832. https://doi.org/10.1063/1.1144830.

Bowling, Michael, Johannes F urnkranz, Thore Graepel Graepel ja Ron Musick. 2006. "Machine learning and games". *Mach Learn* 63.3:211–215. https://doi.org/10.1007/s10994-006-8919-x.

Browne, Cameron B., Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis ja Simon Colton. 2012. "A Survey of Monte Carlo Tree Search Methods". *IEEE Transactions on Computational Intelligence and AI in Games* 4 (1): 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810.

Burda, Yuri, Harrison Edwards, Amos Storkey ja Oleg Klimov. 2018. "Exploration by Random Network Distillation". *CoRR* abs/1810.12894. arXiv: 1810.12894. http://arxiv.org/abs/1810.12894.

Ensmenger, Nathan. 2011. "Is chess the drosophila of artificial intelligence? A social history of an algorithm". *Social Studies of Science* 42(1):5–30. https://doi.org/10.1177/0306312711424596.

Gelly, Sylvain, ja David Silver. 2011. "Monte-Carlo tree search and rapid action value estimation in computer Go". *Artificial Intelligence* 175 (11): 1856–1875. https://doi.org/10.1016/j.artint.2011.03.007.

Genesereth, Michael, Nathaniel Love ja Barney Pell. 2005. "General Game Playing: Overview of the AAAI Competition". *AI Magazine* 26:62. https://doi.org/10.1609/aimag.v26i2.1813.

Grimm, Christopher, André Barreto, Satinder Singh ja David Silver. 2020. *The Value Equivalence Principle for Model-Based Reinforcement Learning.* arXiv: 2011.03506 [cs.LG].

Hafner, Danijar, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee ja James Davidson. 2019. "Learning Latent Dynamics for Planning from Pixels". *Proceedings of the 36th International Conference on Machine Learning* 97:2555–2565. arXiv: 1811.04551 [cs.LG].

Justesen, N., P. Bontrager, J. Togelius ja S. Risi. 2020. "Deep Learning for Video Game Playing". *IEEE Transactions on Games* 12 (1): 1–20. https://doi.org/10.1109/TG.2019.2896986.

Kaiser, Lukasz, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan ym. 2020. "Model-Based Reinforcement Learning for Atari", arXiv: 1903.00374 [cs.LG].

LeCun, Yann, Yoshua Bengio ja Geoffrey Hinton. 2015. *Deep learning.* 521:436–444. Nature. https://doi.org/10.1038/nature14539.

Moerland, Thomas M., Joost Broekens ja Catholijn M. Jonker. 2021. "Model-based Reinforcement Learning: A Survey", arXiv: 2006.16712 [cs.LG].

O'Shea, Keiron, ja Ryan Nash. 2015. "An Introduction to Convolutional Neural Networks". *CoRR* abs/1511.08458. arXiv: 1511.08458. http://arxiv.org/abs/1511.08458.

Puterman, Martin L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc.

Risi, Sebastian, ja Mike Preuss. 2020. "From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI". *Künstl Intell* 34:7–17. https://doi.org/10.1007/s13218-020-00647-w.

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez ym. 2020. "Mastering Atari, Go, chess and shogi by planning with a learned model". *Nature* 588:604–609. https://doi.org/10.1038/s41586-020-03051-4.

Shixiang, Gu, Timothy Lillicrap, Ilya Sutskever ja Sergey Levine. 2016. "Continuous Deep Q-Learning with Model-based Acceleration". *CoRR* abs/1603.00748. arXiv: 1603.00748.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot ym. 2018. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. *Science* 362:1140–1144. https://doi.org/10.1126/science.aar6404.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert ym. 2017. “Mastering the game of Go without human knowledge”. *Nature* 550:354–359. https://doi.org/10.1038/nature24270.

Sutton, Richard S., ja Andrew G. Barto. 2018. *Reinforcement Learning, second edition: An Introduction.* Bradford Books.

Tang, Haoran, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip DeTurck ja Pieter Abbeel. 2017. “#Exploration: A study of count-based exploration for deep reinforcement learning”. 30:1–18.