Joonas Tuomikoski

Long short-term memory ja pelit tekoälytutkimuksessa

Tietotekniikan kandidaatintutkielma

13. toukokuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joonas Tuomikoski

Yhteystiedot: joarsatu@student.jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: Long short-term memory ja pelit tekoälytutkimuksessa

Title in English: Long short-term memory and games in AI research

Työ: Kandidaatintutkielma

Sivumäärä: 28+0

Tiivistelmä: Tekoälytutkimus pelien avulla on viime vuosina saanut suurta huomiota monien eri toteutusten saavutettua yli-inhimillisen tai ihmisenkaltaisen taitotason. Tässä tutkielmassa tutustutaan yhteen toteutuksissa olleeseen olennaiseen komponenttiin, Long Short-Term Memoryyn. Sen toimintaperiaate käydään läpi, sekä tehdään kirjallisuuskatsaus tutkimuksiin, joissa se on peliä pelaavan toteutuksen komponenttina. Katsauksesta havaitaan, että vaikka tämä on tärkeä komponentti, tutkimus keskittyy tällä hetkellä muihin osa-alueisiin, kuten oppimismenetelmiin ja arkkitehtuuriin.

Avainsanat: long short-term memory, tekoäly, pelitutkimus, vahvistusoppiminen

Abstract: AI-research using games has garnered a lot of attention with many different implementations achieving super-human or human-like skill level. In this thesis one essential component of these implementation, Long Short-Term Memory, is explored. It's working principles are explainen, and a literature review is made to map it's use as a component of a game-playing AI-implementation. The review shows that even though it is a core component in implementations, research today focuses in other sections, like learning methods and AI-architecture.

Keywords: long short-term memory, artificial intelligence, game research, reinforcement learning

Kuviot

Sisältö

1 Johdanto

Tietokoneiden alkuajoista asti on haaveiltu koneesta, joka kykenisi ihmisenkaltaiseen ajatteluun, ja erilaisten pelien pelaamisen on ajateltu olevan mahdollinen tie kohti sitä (Shannon 1950). Yli puoli vuosisataa myöhemmin tämä ajattelutapa on vielä hyvin voimissaan ja ehkä jopa vahvempi kuin koskaan, kuten tässä tutkielmassa havaitaan.

Laskentatehon ja tallennuskapasiteetin kasvu on mahdollistanut suurten datamäärien käytön tekoälyagenttien kouluttamisessa. Erityisesti grafiikkasuorittimien kehitys on ollut tässä suhteessa suotuisaa (Cireşan ym. 2010). Niiden teho perustuu monien rinnakkaisten laskutoimitusten suorittamiseen, ja siten ne soveltuvat hyvin syvien neuroverkkojen painokerrointen laskentaan (Baji 2018). Koska yksi ajava tekijä grafiikkaprosessorien kehityksessä on halu yhä visuaalisesti näyttävämmille peleille, muodostuu toinen mielenkiintoinen linkki tekoälytutkimuksen ja pelien välille.

Kuitenkaan pelkän raa'an laskentatehon kasvattamien ei ole tae tekoälytotetusten kehittymiselle. Uusia innovaatioita tarvitaan myös algoritmeissa ja oppimismenetelmissä. Useissa peleissä on välttämätöntä ottaa huomioon aiemmat tapahtumat, sillä kaikki informaatio, kuten reaalimaailmassa, ei ole joka hetkellä saatavilla. Tämän temporaalisen ulottuvuuden haltuunotto on ollut haaste tekoälykehitykselle. Yksi tätä ongelmaa ratkaisemaan pyrkivä menetelmä on Sepp Hochreiterin ja Jürgen Schmidhuberin vuonna 1997 esittelemä *long short-term memory* (Hochreiter ja Schmidhuber 1997). Loppututkielman ajan käytetään tästä lyhennettä LSTM.

LSTM:n tarkka toimintaperiaate selitetään kolmannessa luvussa. Kuitenkin nopeana johdantona, LSTM käyttää muistisoluja, jotka oppivat tunnistamaan relevantit tapahtumat ja syöttämällä niitä takaisin soluun kunnes ne voidaan unohtaa tarpeettomina. Tämä tapahtuu nk. porttien avulla, jotka säätelevät solun tilaa, joka edelleen määrää edellisten tapahtumien vaikutuksen ulostuloon. Solu siis oppii muistamaan tärkeät tapahtumat joista voisi olla hyötyä tulevaisuudessa sen sijaan, että pyrittäisiin muistamaan jokaista yksityiskohtaa. Yhtäläisyydet ihmismielen toimintaan ovat ilmeisiä.

Noin 25 vuotta sitten TD-Gammonista (Tesauro 1995) alkanut ihmisentasoisten tekoälype-

laajien kehitys on ottanut suuria harppauksia 2010-luvulla. Yksi merkkipaalu saavutettiin vuonna 2018, kun täysin itseoppinut ohjelma AlphaGo (Silver ym. 2017) voitti hallitsevan maailmanmestarin lautapelissä Go. Tekoälypelaaja oli kehittynyt ilman ihmisen ohjausta parhaaksi pelissä, jota pidettiin tietokoneelle paljon shakkia haastavampana.

Tekoälyn käyttö pelaajana ei kuitenkaan rajoitu lautapeleihin. Videopelien käyttö tutkimuksessa alkoi 2000-luvun alussa, ja on vain kasvattanut suosiotaan (Yannakakis ja Togelius 2018). 2010-luku olikin akateemiselle tutkimukselle mielenkiintoista aikaa: Tutkittavien peligenrejen kirjo kasvoi kattamaan lähes koko kentän, ja tutkimukset suosituissa verkkomoninpeleissä, kuten Dota 2 (OpenAI ym. 2019) ja Starcraft II (Vinyals ym. 2019), nostivat alan suuren yleisön tietoisuuteen.

Tämän tutkielman tarkoituksena on kartoittaa kirjallisuutta siitä, millaista tutkimusta LSTM:stä ja peleistä on tehty, sekä millaisia ongelmia se ratkaisee. Luvussa 2 käsitellään tekoälyn ja pelitutkimuksen taustaa, jonka jälkeen käydään läpi LSTM:n toimintaperiaate ja sen yleisimmät variaatiot. Luvussa 4 tarkastellaan kirjallisuutta pelejä pelaavista, LSTM:ä hyödyntävistä tekoälytoteutuksista. Lopuksi tehdään yhteenveto tutkielmassa esiin tulleista havainnoista.

2 Teoria

Tässä luvussa käydään läpi tarvittava teoria, jotta LSTM:n ja sitä käyttävien pelien tarkastelu olisi mielekästä. Tavoitteena on luoda lukijalle intuitiivinen käsitys käsiteltävistä tekoälymenetelmistä, joten kovin syvälle matematiikkaan ei paneuduta.

2.1 Pelit ja tekoäly

Jo 70 vuotta sitten Claude Shannon esitti ajatuksen shakkia pelaavasta tietokoneesta (Shannon 1950). Vaikkei hän ajatellutkaan shakkikoneesta olevan kovin paljon käytännön hyötyä, Shannon uskoi sen olevan askel kohti merkityksellisempiä haasteita - kuten loogiseen päättelyyn ja melodioiden säveltämiseen. Yhdeksän vuoden kuluttua vuonna 1959 tarkastellessaan kahta koneoppimismenetelmää tammen palaamisessa Arthur Samuel päätteli että tietokone voidaan ohjelmoida niin, että se pystyy oppimaan pelaamaan tammea paremmin kuin sen ohjelmoija, sekä ennusti, että tulevaisuudessa tätä voisi yleistää reaalimaailman tilanteisiin (Samuel 1959).

Pelit ovat siis olleet tekoälytutkimuksessa vahvassa asemassa koko sen historian ajan. Yannakakis ja Togelius 2018 listaavat viisi pääsyytä tietokonepelien kiinnostavuudelle tutkimuksessa:

1. Pelit ovat vaikeita ja mielenkiintoisia ongelmia
2. Rikas vuorovaikutus tietokoneen ja ihmisen välillä
3. Pelit ovat suosittuja
4. Tarjoavat haasteita jokaiselle tekoälyn osa-alueelle
5. Toteuttavat tekoälytutkimuksen pitkäaikaistavoitteita

Nämä syyt lukuun ottamatta toista ja kolmatta ovat hyvin oleellisia tämän tutkielman käsittelemissä artikkeleissa. Vuorovaikutus ja pelien suosio eivät kirjallisuutta käsitellessä nousseet esille, ja vaikuttavat lähinnä epäsuorasti. Pelitutkimuksesta voidaan oppia paljon tekoälyn ja ihmisen eroista vuorovaikutuksessa (esim. havainnointi), suosion tuodessa monimuotoisempia pelejä ja lisää dataa (Yannakakis ja Togelius 2018).

Ensimmäisellä kohdalla Yannakakis ja Togelius viittaavat pelien laskennalliseen vaikeuteen: mahdollisuudet eri pelistrategioihin ovat valtavia, ja hyvän tilan evaluointi voi olla vaikeaa. Kuitenkin lopputilanteen (voitto/häviö, pisteet) arviointi on usein helppoa. Monet pelit ovat NP-kovia ongelmia (Yannakakis ja Togelius 2018).

Kuten myöhemmin luvussa 4, harva peliä pelaava tekoälytoteutus käyttää vain yhtä tekoälymenetelmää. Pelin käyttö antaakin mahdollisuuden tutkia monia eri tekoälyn osa-alueita, kuten konenäköä ja tekstintunnistusta, sekä kuinka nämä menetelmät toimivat yhdessä.

Viimeinen kohta on monimutkaisin, sillä yksimielisyyden saavuttaminen tässä kysymyksessä on haastavaa.Yannakakis ja Togelius laskevat päätavoitteiksi sosiaalisen ja tunneälyn, laskennallinen luovuuden, sekä vahvan tekoälyn tavoittelun. Nämä saattavat kuulostaa melko suureellisilta ja kauaskatseisilta, joten voidaan löytää yksinkertaisempi motivaatio tekoälytutkimukselle: ratkaista reaalimaailman ongelmia.

2.2 Vahvistusoppiminen

Vahvistusoppiminen (reinforcement learning) oli löytyneissä artikkeleissa pääasiallinen oppimismenetelmä. Sen tomintaperiaate on hyvin samankaltainen ihmisten oppimisen kanssa: onnistuneesta suorituksesta annetaan positiivinen palkkio ja huonosta negatiivinen. Tietokoneiden ollessa kyseessä palkkiona on jokin numeroarvo, ja tavoitteena on valita kussakin tilanteessa toiminto, joka maksimoi palkkion (Sutton ja Barto 2018). Palkkiota vastaa esimerkiksi peleissä jonkin pelitilanteen "hyvyys", tai yksinkertaisesti pelin voittaminen.

Vahvistusoppimisen etu muihin koneoppimisen pääparadigmoihin, valvottuun oppimiseen (*supervised learning*) ja valvomattomaan oppimiseen (*unsupervised learning*), on ettei se tarvitse luokiteltua dataa, tai dataa ollenkaan toimintamallin saavuttamiseen. Valvotussa oppimisessa tekoälyagentti oppii yleistämään luokitellusta datasta toimintamalleja, ja valvomattomassa etsimään datassa esiintyviä rakenteita (Sutton ja Barto 2018).

Yleisin vahvistusoppimismenetelmä tutkimuksissa oli Q-oppiminen (Q-learning, Watkins 1989). Koska pelin tilojen s ja toimintojen a joukot S ja A (hetkellä t) ovat äärellisiä, voidaan pelien formaalisti ajatella olevan äärellisiä Markovin päätösprosesseja. Tällöin on olemassa

optimaalinen toimintamalli $q_*(s,a)$ joka maksimoi odotetun palkkion Bellmanin yhtälön

$$q_*(s,a) = E[R_{t+1} + \gamma \max q_*(s',a')] \tag{2.1}$$

mukaisesti, jossa $s \in S$, $a \in A$, R_{t+1} on seuraavan askeleen palkkio, γ on vähennyskerroin ja $\max q_*(s',a')$ on maksimipalatutuksen antava toimintapa. Tavoitteena on siis maksimoida myös tulevien askeleiden palkkio kuitenkin painottamalla niitä sitä vähemmän mitä kauempana ne olevat.

Jokaisen toiminto-tila -parin läpikäynti ei ole järkevää, koska tällöin konvergointi optimaaliseen toimintamalliin olisi hidasta. Sen sijaan voidaan tasapainotella tutkimisen ja hyödyntämisen välillä. Tutkiessa agentti tekee satunnaisia valintoja tavoitteenaan saada jokin hyvä palkkio. Kun palkkion määrä kasvaa ja siten lähestytään hyvää toimintamallia, voidaan tutkimisen määrää vähentää ja alkaa hyödyntämään löydettyä mallia.

Toteutuksia voidaan myös tehostaa toimija-kriitikko -mallilla (*actor-critic -model*). Lyhyesti tämä tarkoittaa kriitikon lisäämistä toteutukseen. Tämä kriitikko oppii arvioimaan tilojen hyvyyttä ja siten auttaa toimijaa valitsemaan parhaan toiminnon a kussakin tilassa s.

2.3 Neuroverkot

Neuroverkkomallien tutkimusta tietotekniikassa on tehty jo lähes 80 vuotta (McCulloch ja Pitts 1943). Neuroverkot pyrkivät matkimaan eliöiden hermostoa keinotekoisilla neuroneilla, jotka kuten normaalit neuronit säätelevät tiedonkulkua aktivoitumalla ja linkittymällä muihin neuroneihin. Neuroneita kutsutaan keinotekoisten neuroverkkojen kontekstissa solmuiksi (*node*), ja monen toisiinsa linkittyneiden solmun verkostoa neuroverkoksi (*neural network*). Jokaisella solmuun vievällä linkillä on oma painokertoimensa w, joka määrää solmun aktivoitumisen a. Keinotekoisissa neuroverkkoarkkitehtuureissa solmut on usein järjestetty kerroksiin L. Tekoälytoteutuksista jotka käyttävät hyväkseen monikerroksista neuroverkkoa käytetään nimitystä syväoppiminen (*deep learning*) (Alpaydin 2016).

Keinotekoisissa neuroverkoissa on monenlaisia arkkitehtuureja, mutta tämän tutkielman kannalta mielenkiintoisimmat ovat yksinkertainen myötäkytketty neuroverkko (*feedforward neural network*), sekä takaisinkytketty neuroverkko (*recurrent neural network*), jonka erikoista-

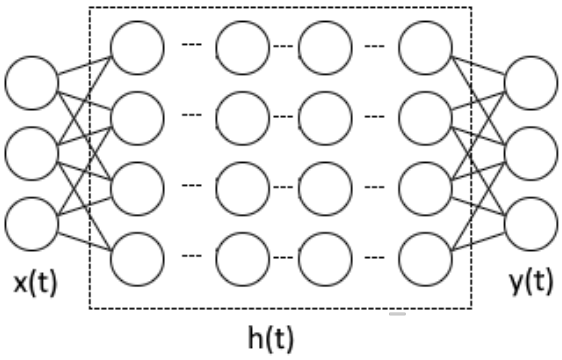
paukseksi LSTM luokitellaan.

2.3.1 Myötäkytketyt neuroverkot

Myötäkytketty neuroverkko on yksi yksinkertaisimmista arkkitehtuureista. Sisääntuleva informaatio $x(t)$ kulkee kuvan 1 mukaisesti piilokerrosten (n kappaletta), joiden viimeistä kerrosta voidaan merkitä $h(t)$, läpi. Lopuksi nämä vielä viedään yhden funktiokerroksen läpi, joka tuottaa ulostulon $y(t)$. Funktiona viimeisellä kerroksella on normalisoiva aktivaatiofunktio, kuten hyperbolinen tangentti-, logistinen sigmoidi- tai softmax-funktio.

Myötäkytketyt neuroverkot ovat yksinkertaisuudestaan huolimatta edelleen käytössä monimutkaisempien arkkitehtuurien osana (Vinyals ym. 2019). Yksi yleinen variaatio tällöin on monikerroksinen perseptroni (*multi-layered perceptron*), jossa viimeisenä kerroksena on yksi solmu. Tällä tavalla saadaan syöttämällä vektori ulostuloksi jokin lukuarvo, esimerkiksi pelin tilasta sumean logiikan totuusarvo.

Ajanhetkillä t ei ole myötäkytketyissä neuroverkoissa merkitystä, sillä sisääntulo tai piilokerrosten painot eivät riipu edellisistä tai tulevista ajanhetkistä. Ne ovat tässä merkitty yhdenmukaisuuden vuoksi, sillä niiden tärkeys tulee esiin vertailtaessa myötäkytkettyjä takaisinkytkettyihin neuroverkkoihin.

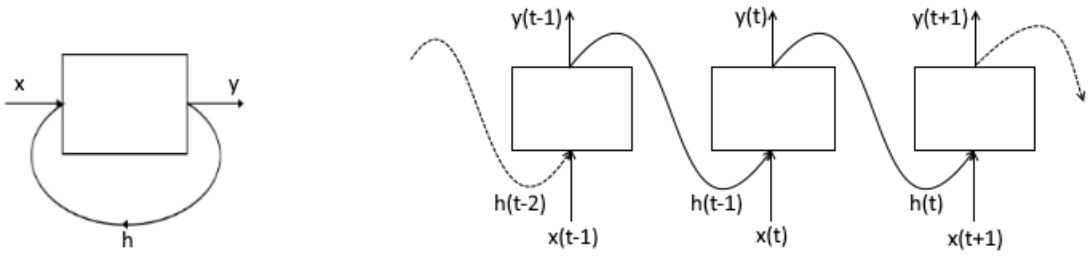


Kuvio 1. Myötäkytketty neuroverkko

2.3.2 Takaisinkytketyt neuroverkot

Tämän tutkimuksen kannalta mielenkiintoisin keinotekoinen neuroverkko on kuitenkin takaisinkytketty neuroverkko. Jos sisääntuleva informaatio on jollain tavalla kytkeytynyt edellisiin tai tuleviin informaatioihin, täytyy tämä jollain tavalla ottaa huomioon. Esimerkiksi lauseessa sanan merkitys saattaa riippua edellisistä sanoista tai jopa lauseista, tai tulevista sanoista. Samalla tavalla peleissä optimaalinen siirto voi riippua menneistä tai tulevista tapahtumista. Kaksisuuntaiset eli bidirektionaaliset neuroverkot, jotka ottavat molemmat suunnat huomioon, jätetään tässä tutkielmassa kuitenkin vain maininnan tasolle.

Takaisinkytketyt neuroverkot eroavat siis yksinkertaisemmista neuroverkoista, joissa informaatio kulkee vain yhteen suuntaan, lisäämällä vähintään yhden syklisen polun neuroverkkoon. Tavoitteena on luoda pidemmän aikavälin riippuvuuksia. Yksinkertaisin arkkitehtuuri tällaiselle neuroverkolle saadaan kytkemällä verkon ulostulo takaisin sisääntuloon uuden informaation kanssa. Kuviossa 2 vasemmalla on takaisinkytketty neuroverkko, josta nähdään helposti syklinen polku. Oikealla on sama neuroverkko, mutta avattuna ajanhetkittäin. Näin ollen edelliset aika-askeleet vaikuttavat, ainakin teoriassa, kaikkiin seuraaviin askeleihin. Käytännössä tämä kuitenkin esittelee uuden ongelman: pidemmällä aikavälillä minimoitava virhe pienenee käyttökelvottomaksi (Hochreiter 1998). Tämä tunnetaan *katoavan gradientin ongelmana*. Yksi mahdollinen ratkaisu on LSTM.



Kuvio 2. Yksinkertainen takaisinkytketty neuroverkko

3 Long short-term memory

Ihmisen muisti voidaan jakaa säilyvyyden mukaan kolmeen osaan: Aistimuistiin, työmuistiin sekä säiliömuistiin. Tekoälyn tapauksessa aistimuistina toimivat neuroverkot; esimerkiksi kuvantunnistuksessa sisääntulevan havainnon, eli kuvan, voidaan katsoa olevan aistimuistissa sen kulkiessa konvoluutiokerrosten läpi. Myös säiliömuistille on helppo löytää vastine tiedontallennusvälineistä, kuten kovalevyistä, joten datan tallentaminen ja sen palauttaminen muistista ei ole tietokoneille ongelma.

Toimivan työmuistin luominen tekoälylle ei kuitenkaan ole yksinkertaista: takaisinkytketyt neuroverkot pyrkivät ylläpitämään lyhytaikaisia aikariippuvuuksia ja käsitekokonaisuuksia, mutta kärsivät jo aiemmin mainitusta katoavan gradientin ongelmasta (Goodfellow, Bengio ja Courville 2016).

3.1 Katoavan gradientin ongelma

Käytettiin sitten vahvistusoppimista, valvottua oppimista tai valvomatonta oppimista, täytyy neuroverkon painot jollain tavalla muuttaa. Algoritmeja muuttamiseen on monia (Goodfellow, Bengio ja Courville 2016), mutta suurin osa niistä perustuu gradienttien laskentaan. Tämä voidaan tehdä *vastavirta-algoritmin* (backpropagation) avulla.

Neuroverkon ulostulon eroa haluttuun ulostuloon kutsutaan tappioksi (*loss*) ja sen laskevaa funktiota tappiofunktioksi (*loss function*). Intuitiivisesti Q-oppimiseen viitaten tämän voi ajatella olevan ero nykyisen toimintatavan ja optimaalisen toimintatavan välillä. Tällöin tappion minimointi toisi neuroverkon lähemmäksi parasta mahdollista toimintamallia. Laskemalla tappion derivaatta suhteessa painoihin saadaan siis selville, kuinka painoja tulisi muuttaa, jotta optimaaliseen toimintamalliin päästäisiin.

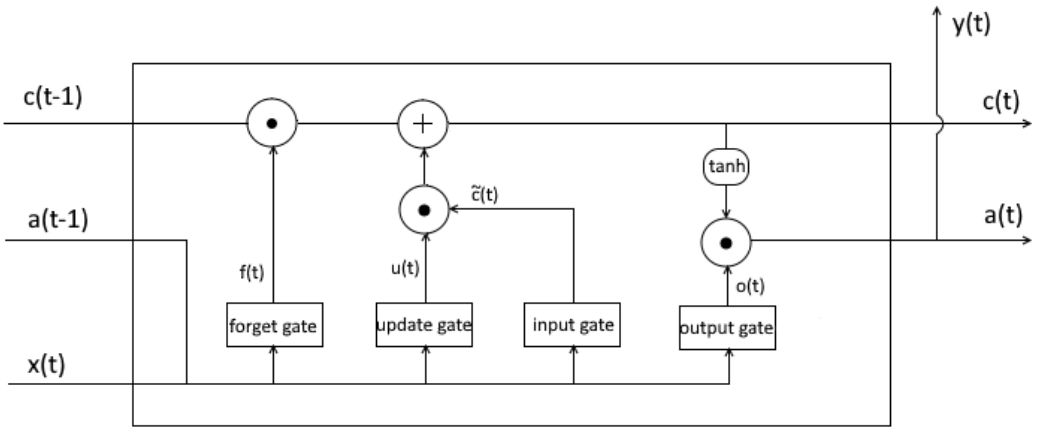
Vastavirta-algoritmi nimensä mukaisesti laskee gradientteja neuroverkon kerroksissa alkaen viimeisestä kerroksesta. Tällöin edellisen kerroksen painojen muutokset tulevat riippumaan seuraavan kerroksen painoista. Painojen ollessa usein rajoitettu välille $[0, 1]$, lähestyvät ensimmäisten kerrosten gradientit nollaa. Ensimmäisillä kerroksilla ei siis tapahdu painojen

muokkaamista varsinkin käytettäessä monikerroksisia, syviä neuroverkkoja. Tämä tunnetaan *katoavan gradientin ongelmana*.

Kuinka tämä liittyy takaisinkytkettyihin neuroverkkoihin? Yleistettyä vastavirta-algoritmia voidaan yksinkertaisesti soveltaa takaisinkytkettyihin neuroverkkoihin eikä erillistä algoritmia tarvita (Goodfellow, Bengio ja Courville 2016). Nyt jonkin sekvenssin lopussa lasketaan tappion gradientti suhteessa aiempien ajanhetkien tappioihin. Mutta taas kerroksia, tai tässä tapauksessa ajanhetkiä, ollessa monia, ei ensimmäisillä kerroksilla tapahdu muutosta. Takaisinkytketty neuroverkko ei siis opi pitkiä aikariippuvuuksia.

3.2 LSTM

Hochreiter ja Schmidhuber (1997) pyrkivät ratkaisemaan katoavan gradientin ongelman esittämällä idean muistisolusta, jossa pelkän takaisinkytkennän sijaan informaation etenemistä säädeltäisiin porttien (*gate*) ja solun tilan (*cell state*) avulla. Kuviossa 3 on yksi mahdollinen konfiguraatio muistisolulle. Käydään seuraavaksi yksityiskohtaisesti läpi muistisolun osat ja LSTM:n toimintaperiaate.



Kuvio 3. Muistisolu

Ensimmäiseksi on syytä huomauttaa, että nimeämiskäytänteissä on huomattavia eroja. Osa kirjallisuudesta käyttää ylläolevan mukaisia nimiä (Goodfellow, Bengio ja Courville 2016), osa taas nimittää muistisolua blokiksi (*block*) ja tilaa soluksi (Greff ym. 2016). Hochreiter

ja Schmidhuber soluista kirjoittaessaan tarkoittivat yksittäisiä arvoja tilassa, jota he taas nimittivät muistisolublokiksi (*memory cell block*). Tässä tutkielmassa käytetään nimiä, jotka parhaiten kuvaavat osia suomeksi käännettyinä.

3.2.1 Yleiskuvaus

LSTM sisältää normaalin takaisinkytkennän liittämällä edellisen aika-askeleen aktivaation a^{t-1} nykyisen askeleen sisääntuloon x^t. Samoin kuin muissakin neuroverkoissa voidaan aktivaatiosta a^t ottaa ulostulo y^t. Tämä voi olla esimerkiksi peleissä päätös suorittaa jokin toiminto.

Ero tavallisiin takaisinkytkettyihin neuroverkkoihin on solun tilassa c, joka kiertää silmukassa ajanhetkestä toiseen. Edellisen ajanhetken tila c^{t-1} voidaan unohtaa ja/tai päivittää, ja lopulta c^t, x^t ja a^{t-1} määräävät ulostulon. Koska solun tila on käytännössä vektori muistettavista asioista, voidaan osa alkioista unohtaa tai päivittää, tai jopa jättää koko tila täysin muuttumattomaksi. Tämä toimii ratkaisuna katoavan gradientin ongelmaan vastavirta-algoritmissa ajan suhteen (Hochreiter ja Schmidhuber 1997).

Solun tilaa ylläpidetään porttien avulla. Portit ovat neuroverkkokerroksia painoilla W ja biaseilla b, ja ovat kuviossa 3 kuvattuna suorakaiteilla. Porttien aktivaatiofunktiona käytetään usein logistista sigmoidia, paitsi sisääntuloportissa (*input gate*) jossa käytetään hyperbolista tangenttia (Greff ym. 2016).

Alkaen vasemmalta ensimmäinen portti on unohtamisportti (*forget gate*). Unohtamisportin tehtävänä on oppia nollaamaan solun tilaa, jotta turha informaatio ei enää vaikuttaisi ulostuloon. Tätä ei ollut alkuperäisessä LSTM:ssä ja se lisättiin myöhemmin rajoittamaan tilan lineaarista kasvua aika-askelittain (Gers, Schmidhuber ja Cummins 1999).

Päivittämisportti (*update gate*) ja sisääntuloportti (*input gate*) säätelevät yhdessä uuden informaation lisäämistä solun tilaan c. Päivittämisportti päättää mikä informaatio on relevanttia, ja sisääntuloportti luo uuden tilan ehdokkaan $\widetilde{c}$. Näiden alkioittainen tulo voidaan lisätä solun tilaan.

Viimeisenä on ulostuloportti (*output gate*), joka nimensä mukaisesti tuottaa nykyisen ajan-

hetken ulostulon o^t. Tämä olisi yksinkertaisen takaisinkytketyn neuroverkon ulostulo, mutta LSTM:ssä lopullinen aktivaatio saadaan kertomalla alkioittain o^t solun tilasta otetulla hyperbolisella tangentilla $\tanh c^t$.

3.2.2 Matemaattinen kuvaus

Alaindeksit painoissa W ja biaseissa b kuvaavat mille porteille ne kuuluvat. Yläindeksit kertovat aika-askeleen. Alkioittaista tuloa (Hadamardin tulo) merkitään operaattorilla $\odot$.

Unohtamisportin ulostulo on

$$f^t = \sigma(W_f[a^{t-1}, x^t] + b_f) \tag{3.1}$$

Päivittämisportin ulostulo on

$$u^t = \sigma(W_u[a^{t-1}, x^t] + b_u) \tag{3.2}$$

Sisääntuloportti taas luo uuden tilan ehdokkaan

$$\widetilde{c} = \tanh(W_i[a^{t-1}, x^t] + b_i) \tag{3.3}$$

Ehdokkaasta $\widetilde{c}$ saadaan poimittua relevantit asiat kertomalla se u^t:lla.

Viimeisenä on ulostuloportti, joka tuottaa nykyisen ajanhetken ulostulon

$$o^t = \sigma(W_o[a^{t-1}, x^t] + b_o) \tag{3.4}$$

Nyt voidaan siis kirjoittaa yhtälö solun tilalle

$$c^t = c^{t-1} \odot f^t + u^t \odot \widetilde{c} \tag{3.5}$$

Tästä nähdään helposti porttien vaikutukset solun tilaan. Esimerkiksi jos unohtamisportin ulostulon f^t alkiot ovat nollia, kaikki edelliset aika-askeleet unohdetaan. Toisaalta jos f^t:n alkiot ovat ykkösiä ja päivittämisportin ulostulon u^t alkiot ovat nollia, jää solun tila kiertämään muuttumattomana. Huomattavaa toki on, että arvot f^t, u^t ja $\widetilde{c}$ muuttuvat jokaisella aika-askeleella.

Lopullinen aktivaatio saadaan solun tilasta ja ulostuloportista

$$a^t = \tanh c^t \odot o^t \tag{3.6}$$

3.2.3 Variaatiot ja käyttötavat

Kuten aiemmin viitattiin, LSTM ei ole pysynyt muuttumattomana vuoden 1997 ensiesittelyn jälkeen. Unohtamisportin (Gers, Schmidhuber ja Cummins 1999) lisäämisen jälkeen Gers ja Schmidhuber (2000) pitivät LSTM:ä edelleen puutteellisena; portit eivät saaneet informaatiota solun tilasta kuin vasta seuraavalla aika-askeleella, ja tällöinkin muutettuna. Ratkaisuna olivat kurkistusaukot (*peephole*).

Tämä on nykyisin yleisimmin käytetty variaatio (Greff ym. 2016). Tämän tutkielman kuviosta 3 ja luvuista 3.2.1 ja 3.2.2 kurkistusaukot jätettiin pois yksinkertaistamisen vuoksi. Mistään kovin monimutkaisesta ei kuitenkaan ole kyse: solun tila syötetään portille kullakin aika-askeleella ennen tilan muokkaamista.

Kurkistusaukkojen tärkeyttä on kuitenkin kyseenalaistettu kirjallisuudessa. (Greff ym. 2016) ja Breuel (2015) tulivat suorituskykytesteissään johtopäätökseen, että kurkistusaukot eivät juurikaan lisänneet tarkkuutta, ja niiden poistaminen vähentää laskennallista raskautta. Tästä syystä on jatkotutkimuksen kannalta huonoa, ettei useimmissa tässä tutkielmassa käsiteltävissä artikkeleissa mainittu millaista LSTM:ää käytettiin. Näiden selvittäminen voisi jatkossa tuoda hyvää lisätietoa tekoälytoteutuksien rakentamiseen.

Cho ym. (2014) esittelemää GRU:a (*gated recurrent unit*) voidaan pitää yksinkertaistettuna LSTM:n varianttina. Siinä vain kaksi porttia, päivitysportti ja resetointiportti, ja solun tilan paikalla käytetään suoraan edellisen ajanhetken aktivaatiota. Suorituskyky on joissain tehtävissä verrattavissa LSTM:n suorituskykyyn (Chung ym. 2014).

Usein akateemisessa kirjallisuudessa LSTM:ä ja sen variaatioita käytetään puheen- ja tekstintunnistuksessa (Greff ym. 2016). Muitakin alueita löytyy, kuten ruuhkanhallintaa (Fu, Zhang ja Li 2016) ja huippuhetkien tunnistamista e-urheilussa (Fu ym. 2017). Kuitenkin mielenkiintoisin alue, jolla LSTM on ollut menestynyt, on pelit.

4 LSTM peleissä

Kirjallisuuskatsaus rajattiin toteutuksiin, joissa tekoäly kykeni pelaamaan peliä. Tutkimusta tehdään myös pelien saralla ennustamalla aiemmin tallennetuista pelitilanteista seuraavaa siirtoa tai ruudunpäivitystä (Tsunoda ym. 2017). Vaikka tuloksissa esiintyikin lähinnä videopelejä, ei rajausta niihin tarkoituksella tehty. Kyseessä on lopulta vain rajapintakysymys: shakkia pelaavalle tekoälylle ei ole tutkimuksen kannalta merkitystä liikkuvatko nappulat ruudulla vai liikuttaako ihminen niitä pelilaudalla käskyjen mukaan.

Tutkimuksia huomattiin olevan laajasti monesta pelityypistä. Monessa tutkimuksessa pelien tekoälylle luomat haasteet ovat kuitenkin samoja. Tämän vuoksi peleittäin tai edes genreittäin tutkimusten läpikäyminen ei ole mielekästä. Järkevintä on siis tarkastella muutamaa tutkimusta tarkemmin, ja verrata näitä lopuksi muihin tutkimuksiin. Luonnolliset valinnat tarkemmin läpikäytäviksi tutkimuksiksi ovat state-of-the-art -tutkimukset *Grandmaster level in StarCraft II using multi-agent reinforcement learning* (Vinyals ym. 2019) ja *Dota 2 with Large Scale Deep Reinforcement Learning* (OpenAI ym. 2019).

4.1 StarCraft II ja Dota 2

AlphaGo:n (Silver ym. 2017) voitettua 18-kertaisen maailmanmestarin Lee Sedolin gossa vuonna 2016, on mietitty mikä olisi seuraava peli, jossa tekoälyllä olisi mahdollisuus kehittyä ihmispelaajaa taitavammaksi. Paljon kiinnostusta on osoitettu peleihin StarCraft II (Blizzard Entertainment, 2010) ja Dota 2 (Valve Corporation, 2013) (Justesen, Debus ja Risi 2019).

4.1.1 Yleiskuvaukset

StarCraft II on reaaliaikainen strategiapeli, jossa pelaaja kasvattaa armeijaansa taistellen samaan aikaan yhtä tai useampaa vastustajaa vastaan. Pelaajan ohjattavissa voi parhaimmillaan olla useita kymmeniä yksittäisiä joukkoja. Pelin voittaa tuhoamalla kaikki vastustajan rakennukset tai jos vastustaja luovuttaa. Ennen pelin alkua pelaaja valitsee yhden kolmesta rodusta, jolla aikoo pelata. Kullakin rodulla on eri joukot ja rakennukset, joten valinnalla on suuri vaikutus käytettävään pelistrategiaan. Kaikkien rotujen sisällä on kuitenkin monipuolisesti

erilaisia joukkoja, joten pelitaktiikan muutos ottelun aikana on mahdollista.

StarCraft:lla pelisarjana on historia tietokonevastustajien kehityksessä, sillä pelisarjan aiempi osa StarCraft: Brood War on toiminut alustana tekoälyn kehityskilpailuille (Ontanon ym. 2013). Nämä tietokonevastustajat eivät kuitenkaan rajoitu käyttämään neuroverkkoja tai syväoppimista (Ontanon ym. 2013). Kuitenkin myös LSTM:ää on käytetty eräissä toteutuksissa (Gehring ym. 2018).

MOBA -genreen (*Multiplayer Online Battle Arena*) kuuluvassa Dota 2:ssa pelaaja ohjaa yhtä viiden hahmon joukkueeseen kuuluvaa sankaria toista viiden hahmon joukkuetta vastaan, tavoitteenaan tuhota vastustajan tukikohdassa sijaitseva päärakennus. Jokaisella hahmolla on hahmosta riippuen noin neljä erikoistaitoa, joten tässäkin pelissä on ennen varsinaisen ottelun alkua tapahtuvalla sankarin valinnalla suuri merkitys pelin kulkuun. Pelin aikana palaaja kerää itselleen rahaa ja kokemuspisteitä tuhoamalla vastustajan puolustusrakennuksia, tappamalla vastustajan joukkoja ja pitämällä hallussaan tiettyjä osia pelikentästä. Rahalla sankarille voi ostaa yli sata tavaraa sisältävästä kaupasta maksimissaan kuusi käytettävää tavaraa, jotka lisäävät sankarin voimaa ja/tai antavat lisää erityistaitoja. Kokemuspisteet taas kasvattavat hahmospesifejä taitoja ja attribuutteja, kuten elämäpisteet. Sankarin kuollessa pelaaja joutuu odottamaan kuluneen pelin kestosta riippuvan ajan, jonka jälkeen hahmo herää eloon päärakennuksensa luota.

4.1.2 Haasteet tekoälylle

Vaikka nämä kaksi ovat pelillisesti melko erilaisia, luovat ne osittain samoja haasteita tekoälylle. Molemmissa peleissä on staattinen kartta, josta pelaaja voi nähdä kerrallaan vain osan päänäkymässään. Pelaajalla on kutenkin jatkuvasti näkyvillä pienoisrepresentaatio (nk. minimap) koko kartasta, mutta pelaaja ei voi nähdä kuin mitä oman puolen joukot näkevät. Dota 2 ja StarCraft 2 ovat siis *epätäydellisen informaation pelejä*.

Tämä jo itsessään on suuri ero shakkiin ja gohon luoden uuden haasteen: temporaalisuuden. Shakissa ja gossa koko pelilauta on jatkuvasti näkyvillä, ja kumpikin pelaaja tietää mitkä ovat vastustajan mahdolliset siirrot. AlphaGo Zero ja sen seuraaja AlphaZero toimivat Monte-Carlo-puuhakualgoritmilla ja ottivat syötteenä pelilaudan tilanteen, simuloivat mah-

dollisia skenaarioita, ja lopuksi valitsivat siirron, joka johtaisi suotuisimpaan tilanteeseen. Näin ollen nämä toteutukset eivät välittäneet edellisistä siirroista (Silver ym. 2018). Niiden huomioimatta jättäminen ei kuitenkaan toimi Dota 2:n ja StarCraft 2:n kaltaisissa peleissä: näkökentän ulkopuolelle katoava vastustajahahmo ei poistu pelilaudalta. Peleissä ei myöskään ole aikarajaa, joten pelit saattavat kestää jopa yli tunnin. Pitkät aikariippuvuudet ja puutteellinen informaatio on siis ratkaistava jollain tavalla.

Toinen haaste liittyy pelien suureen tila-avaruuteen, niin pelikentän laajuudessa ja mahdollisten siirtojen määrässä. Kyseisissä peleissä pelikenttä on suuri, sitä ole jaettu ruutuihin, siitä ei voi nähdä kerralla kuin osan, ja tunnistettavia hahmoja ja efektejä on satoja. Shakissa pelilauta voidaan esittää 64 bitillä, ja pelitilanteet luomalla tällaisia bittikarttoja eri nappuloille. Samankaltainen (OpenAI ym. 2019) tai hieman ihmisenpelaajan näkymää vastaava representaatio (Vinyals ym. 2019) oli käytössä näissä kahdessa tutkimuksessa, jotta esim. kuvantunnistuksen virheet eivät tuottaisi ongelmia muulle toteutukselle. Mahdollisten "siirtojen"määrästä johtuen huomiokyvyn jakaminen ja siitä seuraava päätöksenteko ovat merkittäviä haasteita peleissä (Vinyals ym. 2019).

Myös pelien reaaliaikaisuus aiheuttaa omat haasteensa tekoälylle. Koska peleissä ei ole vuoroja ja jotkin toiminnot vievät enemmän aikaa kuin toiset, ei optimaalisen siirron etsimiseen ole kovin paljoa aikaa. Yllättäen myös toinen ääripää on mahdollinen: ihmisen reaktiokyvyn ja käskyjen antamisen hiiren ja näppäimistön avulla ollessa rajattu, saa tekoäly yliinhimillisen edun. Jotta tutkimus keskittyisi enemmän "älykkyyteen"ja päätöksentekoon, voi tiettyjen rajoitusten asettaminen olla perusteltua (Vinyals ym. 2019).

4.1.3 AlphaStar ja OpenAI Five

StarCraft II:een luotu toteutus nimettiin AlphaStar:ksi ja Dota 2:een OpenAI Five:ksi, ja molemmissa tutkimuksissa käytettiin tekoälytoteutuksen ytimessä LSTM:ä. Tarkasteltaessa toteutuksia korkealla tasolla ovat molempien arkkitehtuurit hyvin samanlaiset: Observaatiot pelin tilasta prosessoitiin yhdeksi vektoriksi, joka syötettiin LSTM:lle, jonka tekemä päätös toteutettiin erillisellä toteutuskerroksella. Molemmissa LSTM:n ulostulo kopioitiin lisäksi erilliselle arvoivalle neuroverkolle, joka ennustaa tulevia palkkioita ja siten ohjaa oppimista.

Observaatiokerros koostui AlphaStar:ssa myötäkytketyistä neuroverkoista, joiden tehtävänä oli muokata saatu havainto (pelin tila) LSTM:lle sopivaksi päätöksentekoa varten. Myös toteutuskerros koostui myötäkytkeyistä neuroverkoista, jotka ottivat LSTM:n ulostulon ja valitsivat sitä vastaavan toiminnon, sen suorittavat joukot, kohteen ym. riippuen toiminnosta.

Toteutuskerrokset saivat havaintoja myös suoraan observaatiokerroksilta. Jokaiseen erilliseen toimintoon ei siis tarvittu päätöstä. Tämä liittyi pelien reaaliaikaisuuteen: Päätöksen tultua ei LSTM:n tarvitse laskea uutta päätöstä pelikentän muututtua suoritettavan toiminnon aikana, ja AlphaStar:ssa LSTM:n ohitetut havainnot menivätkin suoraan valituista yksiköistä tai kohteista vastaaville neuroverkoille.

Yksi merkittävä ero toteutuksissa oli, joka johtui pelien pelaajamäärien erosta. AlphaStar luotiin pelaamaan suosituinta pelimuotoa 1vs1, joten sen arkkitehtuurissa oli yksi jokaista ylläkuvattua komponenttia. Dota 2:n ollessa 5vs5 -peli, koostui OpenAI Five nimensä mukaisesti viidestä toimijasta. Nämä eivät kuitenkaan olleet täysin itsenäisiä: Kaikki viisi saivat samat havainnot observaatiokerrokselta (hieman muokattuna riippuen mitä sankarihahmoa kukin toimija ohjasi), mutta LSTM sekä toteutuskerrokset olivat jokaiselle toimijalle omia. Näiden kerrosten neuroverkkojen painot olivat samat, joten toimijat olivat kopioita toisistaan. Kuitenkin koska jokaisella LSTM:llä oli oma piilotettu tilansa, jokainen toimija teki oman päätöksensä. Tämä on mielenkiintoinen demonstraatio millä keinoin tekoäly kykenee toimimaan yhteistyössä itsensä kanssa.

4.2 Muut artikkelit

Monet pelien luomat haasteet toistuvat pelistä toiseen, joten yhtä yksityiskohtainen tarkastelu ei ole mielekästä. Seuraavaksi siis nostetaan esiin haussa vastaan tulleita haasteita ja ratkaisuja.

4.2.1 Ensimmäisen persoonan pelit ja ajopelit

Dota 2:n ja StarCraft II:n ollessa peliympäristöltään kaksiulotteisia, ovat ensimmäisen persoonan pelit vähintään ulkoasultaan kolmiulotteisia, ja pelin tilan litistäminen vektoriksi ei mahdollisesti tuota toivottua tulosta. Tämä näkyy löytyneiden artikkelien toteutuksista siten,

että AlphaStar:n ja OpenAI Five:n käyttämän observaatiokerroksen tilalla on kuvantunnistuskerros (Lample ja Chaplot 2017) (Ratcliffe ym. 2017) (Jaderberg, Mnih ym. 2019) (Jaderberg, Czarnecki ym. 2019).

Kuvantunnistuksen käyttäminen tuo uuden haasteen LSTM:lle: Jos kuvantunnistuksen tarkkuus ei ole riittävä, LSTM ei saa riittävästi informaatiota, ja oikean toimintamallin oppiminen vaikeutuu (Lample ja Chaplot 2017). Tämä vahvistaa datan laadun merkitystä syväoppimiselle. Konenäön käyttäminen peleissä toimii hyvänä esimerkkinä ajatukselle peleistä monimuotoisena alustana tekoälyn eri osa-alueille, kuten Yannakakis ja Togelius esittivät.

Kuvantunnistus on haasteena myös ajopeleissä (Perot ym. 2017). Ne kuitenkin tarjoavat alustan tutkia ja kehittää menetelmiä, joita myöhemmin voitaisiin käyttää reaalimaailmassa (Ganesh ym. 2016).

4.2.2 Tekstipohjaiset pelit

Tekstin ollessa luonteeltaan sekventiaalista ja takaisinkytkettyjen neuroverkkojen käsitellessä tällaista dataa, ei ole yllättävää, että LSTM esiintyy tekstipohjaisten pelien tutkimuksessa. Tekstipohjaisissa peleissä pelin tila annetaan pelkästään tekstimuodossa, jonka jälkeen pelaaja vuorovaikuttaa pelin kanssa antamalla käskyjä vastaavasti kokonaisilla lauseilla.

Luonnollisten kielten prosessointi (*natural language processing*) on ongelma, johon tekoälyratkaisuja on useita (Goodfellow, Bengio ja Courville 2016). Tekstipohjaiset pelit tarjoavat alustan eri ratkaisujen testaamiseen, joten tutkimukset sisälsivät vertailutuloksia ratkaisujen välillä (Ammanabrolu ja Riedl 2019) (Yin ja May 2019) (Yuan ym. 2018) (Ansari ym. 2018).

Language Understanding for Text-based Games using Deep Reinforcement Learning (Narasimhan, Kulkarni ja Barzilay 2015) ollessa löytyneistä artikkeleista kronologisesti ensimmäinen pyrkivät muut tutkimukset luomaan monimutkaisempia ratkaisuja tekstipohjaisten pelien pelaamiseen. Tuloksista voidaan havaita että vaikka yksinkertaisempi LSTM-malli lopulta kykenee saavuttamaan hyvän onnistumisprosentin (Narasimhan, Kulkarni ja Barzilay 2015), voidaan nopeampi konvergenssi saavuttaa muilla arkkitehtuureilla (Yin ja May 2019).

4.2.3 Muut

Loput artikkelit tutkivat muita pelityyppejä, kuten tappelupelejä (Kanervisto ja Hautamäki 2019) (Li ym. 2018), pokeria (Li ja Miikkulainen 2018) ja Atari-pelejä (Oh ja Kaneko 2018). Nämä tutkimukset vahvistavat aiemmin havaittuja käsityksiä LSTM:n suorituskyvystä monella alueella, mutta myös sen haasteista aiempien kerrosten virheiden kanssa (Li ym. 2018), ja konvergenssin hitaudesta sitä koulutettaessa (Oh ja Kaneko 2018).

5 Yhteenveto

Tässä tutkielmassa tehtiin nopea esittely *Long Short-Term Memory*:n perusideasta, sekä tutkittiin kirjallisuuskatsauksen avulla sen käyttöä peleissä tekoälytutkimuksessa. Tutkielmaa voidaan hyödyntää jatkotutkimuksessa uusien tutkimusalueiden etsimiseen ja vertailukohtien löytämiseen

Tutkielmassa havaittiin LSTM:n olevan olennainen osa nykypäivän peliä pelaavia tekoälytoteutuksia. Sitä on käytetty menestyksekkäästi monen toteutuksen ytimessä päätöksentekoelimenä. Sillä on myös heikkouksia, mutta niitä voidaan minimoida kehittämällä muita tekoälyn osa-alueita. Tämä näkyi tutkimuksissa LSTM:n käytön perustelun vähyytenä, muiden toteutuksen osa-alueiden, kuten oppimisalgoritmien ja muun arkkitehtuurin, ollessa etualalla. Vain tekstipohjaisissa peleissä oli tehty perusteellisempaa vertailua eri LSTM:n ja korvaavien menetelmien välillä. Näissä yksinkertainen LSTM jäi kuitenkin suorituskyvyltään muista jälkeen.

Pelien, tai laajennetusti virtuaaliympäristöjen, käyttö tekoälytutkimuksessa tulee todennäköisesti olemaan tulevaisuudessakin merkittävä osa tekoälytutkimusta. Mahdollisuudet eri menetelmien testaamiseen, niiden vertailuun, sekä agenttien kouluttamiseen ovat niin suuret, ettei niiden käyttämättä jättäminen ole järkevää. Nykypäivän videopelien ollessa reaalimaailman tavoin informaatioltaan epätäydellisiä sekä aikariippuvaisia, tullaan jatkossakin käyttämään LSTM:n kaltaisia ratkaisuja.

Lähteet

Alpaydin, Ethem. 2016. *Machine learning: the new AI.* MIT press.

Ammanabrolu, Prithviraj, ja Mark Riedl. 2019. "Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning": 3557–3565. doi:10.18653/v1/n19-1358.

Ansari, Ghulam Ahmed, Sagar J P, Sarath Chandar ja Balaraman Ravindran. 2018. "Language Expansion In Text-Based Games".

Baji, Toru. 2018. "Evolution of the GPU Device widely used in AI and Massive Parallel Processing". *2018 IEEE Electron Devices Technology and Manufacturing Conference, EDTM 2018 - Proceedings:* 7–9. doi:10.1109/EDTM.2018.8421507.

Breuel, Thomas M. 2015. "Benchmarking of LSTM Networks".

Cho, Kyunghyun, Bart Van Merriënboer, Dzmitry Bahdanau ja Yoshua Bengio. 2014. "On the properties of neural machine translation: Encoder-decoder approaches". *arXiv preprint arXiv:1409.1259.*

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho ja Yoshua Bengio. 2014. "Empirical evaluation of gated recurrent neural networks on sequence modeling". *arXiv preprint arXiv:1412.3555.*

Cireşan, Dan Claudiu, Ueli Meier, Luca Maria Gambardella ja Jürgen Schmidhuber. 2010. "Deep, big, simple neural nets for handwritten digit recognition". *Neural Computation* 22 (12): 3207–3220. ISSN: 08997667. doi:10.1162/NECO_a_00052.

Fu, Cheng Yang, Joon Lee, Mohit Bansal ja Alex C. Berg. 2017. "Video highlight prediction using audience chat reactions". *EMNLP 2017 - Conference on Empirical Methods in Natural Language Processing, Proceedings:* 972–978. doi:10.18653/v1/d17-1102.

Fu, Rui, Zuo Zhang ja Li Li. 2016. "Using LSTM and GRU neural network methods for traffic flow prediction". Teoksessa *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC),* 324–328. IEEE.

Ganesh, Adithya, Joe Charalel, Matthew Das Sarma ja Nancy Xu. 2016. *Deep reinforcement learning for simulated autonomous driving.*

Gehring, Jonas, Da Ju, Vegard Mella, Daniel Gant, Nicolas Usunier ja Gabriel Synnaeve. 2018. “High-Level Strategy Selection under Partial Observability in StarCraft: Brood War”: 1–6.

Gers, Felix A, ja Jürgen Schmidhuber. 2000. “Recurrent nets that time and count”. Teoksessa *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium,* 3:189–194. IEEE.

Gers, Felix A, Jürgen Schmidhuber ja Fred Cummins. 1999. “Learning to forget: Continual prediction with LSTM”.

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

Greff, Klaus, Rupesh K Srivastava, Jan Koutnik, Bas R Steunebrink ja Jürgen Schmidhuber. 2016. “LSTM: A search space odyssey”. *IEEE transactions on neural networks and learning systems* 28 (10): 2222–2232.

Hochreiter, Sepp. 1998. “The vanishing gradient problem during learning recurrent neural nets and problem solutions”. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems* 6 (2): 107–116. ISSN: 02184885. doi:10.1142/S0218488598000094.

Hochreiter, Sepp, ja Jürgen Schmidhuber. 1997. “LSTM can solve hard long time lag problems”. *Advances in Neural Information Processing Systems:* 473–479. ISSN: 10495258.

Jaderberg, Max, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie ym. 2019. “Human-level performance in 3D multiplayer games with population-based reinforcement learning”. *Science* 364 (6443): 859–865. ISSN: 10959203. doi:10.1126/science.aau6249.

Jaderberg, Max, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver ja Koray Kavukcuoglu. 2019. “Reinforcement learning with unsupervised auxiliary tasks”. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings:* 1–14.

Justesen, Niels, Michael S. Debus ja Sebastian Risi. 2019. “When are we done with games?” *IEEE Conference on Computatonal Intelligence and Games, CIG* 2019-Augus. ISSN: 23254289. doi:10.1109/CIG.2019.8847963.

Kanervisto, Anssi, ja Ville Hautamäki. 2019. “ToriLLE: Learning Environment for Hand-to-Hand Combat”. Teoksessa *2019 IEEE Conference on Games (CoG),* 1–8. IEEE.

Lample, Guillaume, ja Devendra Singh Chaplot. 2017. “Playing FPS games with deep reinforcement learning”. *31st AAAI Conference on Artificial Intelligence, AAAI 2017:* 2140–2146.

Li, Xun, ja Risto Miikkulainen. 2018. “Opponent modeling and exploitation in poker using evolved recurrent neural networks”. *GECCO 2018 - Proceedings of the 2018 Genetic and Evolutionary Computation Conference:* 189–196. doi:10.1145/3205455.3205589.

Li, Yu Jhe, Hsin Yu Chang, Yu Jing Lin, Po Wei Wu ja Yu Chiang Frankwang. 2018. “Deep Reinforcement Learning for Playing 2.5D Fighting Games”. *Proceedings - International Conference on Image Processing, ICIP:* 3778–3782. ISSN: 15224880. doi:10.1109/ICIP.2018.8451491.

McCulloch, Warren S, ja Walter Pitts. 1943. “A logical calculus of the ideas immanent in nervous activity”. *The bulletin of mathematical biophysics* 5 (4): 115–133.

Narasimhan, Karthik, Tejas D. Kulkarni ja Regina Barzilay. 2015. “Language understanding for text-based games using deep reinforcement learning”. *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing:* 1–11. doi:10.18653/v1/d15-1001.

Oh, Hyunwoo, ja Tomoyuki Kaneko. 2018. “Deep Recurrent Q-Network with Truncated History”. *Proceedings - 2018 Conference on Technologies and Applications of Artificial Intelligence, TAAI 2018* 34 (1): 34–39. doi:10.1109/TAAI.2018.00017.

Ontanon, Santiago, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill ja Mike Preuss. 2013. “A survey of real-time strategy game AI research and competition in starcraft”. *IEEE Transactions on Computational Intelligence and AI in Games* 5 (4): 293–311. ISSN: 1943068X. doi:10.1109/TCIAIG.2013.2286295.

OpenAI, : Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak ym. 2019. “Dota 2 with Large Scale Deep Reinforcement Learning”.

Perot, Etienne, Maximilian Jaritz, Marin Toromanoff ja Raoul De Charette. 2017. “End-to-end driving in a realistic racing game with deep reinforcement learning”. Teoksessa *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops,* 3–4.

Ratcliffe, Dino S., Sam Devlin, Udo Kruschwit ja ZLuca Citi. 2017. “Clyde: A deep reinforcement learning DOOM playing agent”. *AAAI Workshop - Technical Report* WS-17-01 - WS-17-15:983–990.

Samuel, Arthur L. 1959. “Some Studies in Machine Learning”. *IBM Journal of Research and Development* 3 (3): 210–229.

Shannon, Claude E. 1950. “XXII. Programming a computer for playing chess”. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41 (314): 256–275. ISSN: 1941-5982. doi:10.1080/14786445008521796.

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 (6419): 1140–1144. ISSN: 10959203. doi: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 (7676). ISSN: 14764687. doi:10.1038/nature24270.

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

Tesauro, Gerald. 1995. “Td-gammon: A self-teaching backgammon program”. Teoksessa *Applications of neural networks,* 267–285. Springer.

Tsunoda, Takamasa, Yasuhiro Komori, Masakazu Matsugu ja Tatsuya Harada. 2017. “Football Action Recognition Using Hierarchical LSTM”. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* 2017-July:155–163. ISSN: 21607516. doi:10.1109/CVPRW.2017.25.

Watkins, Christopher John Cornish Hellaby. 1989. “Learning from delayed rewards”.

Vinyals, Oriol, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi ym. 2019. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. *Nature* 575 (7782): 350–354. ISSN: 14764687. doi:10.1038/s41586-019-1724-z.

Yannakakis, Georgios N., ja Julian Togelius. 2018. *Artificial intelligence and games,* 1–337. ISBN: 9783319635194. doi:10.1007/978-3-319-63519-4.

Yin, Xusen, ja Jonathan May. 2019. “Comprehensible context-driven text game playing”. *IEEE Conference on Computatonal Intelligence and Games, CIG* 2019-Augus. ISSN: 23254289. doi:10.1109/CIG.2019.8847954.

Yuan, Xingdi, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew Hausknecht ja Adam Trischler. 2018. “Counting to Explore and Generalize in Text-based Games”.