521160P Johdatus Tekoälyyn

Harjoitus #5

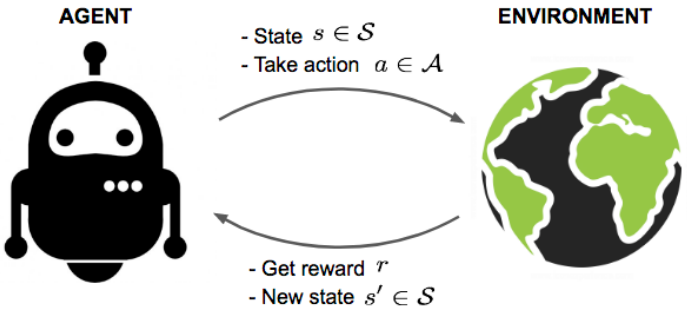
Vahvistusoppiminen ja syväoppiminen

Kevät 2019

Vahvistusoppiminen ja syväoppiminen ovat tekoälyn osa-alueita, joilla ratkaistaan usein varsin erityyppisiä ongelmia. Ne voidaan kuitenkin myös yhdistää, kuten eräässä edistyksellisessä ratkaisussa on tehty, jossa kehitettiin tekoäly Atarin videopeleihin syvä-Q-oppiminen algoritmilla [1]. Yleensä niitä kuitenkin käsitellään kahtena omana osa-alueena, kuten tehdään myös tässä harjoituksessa.

Vahvistusoppiminen

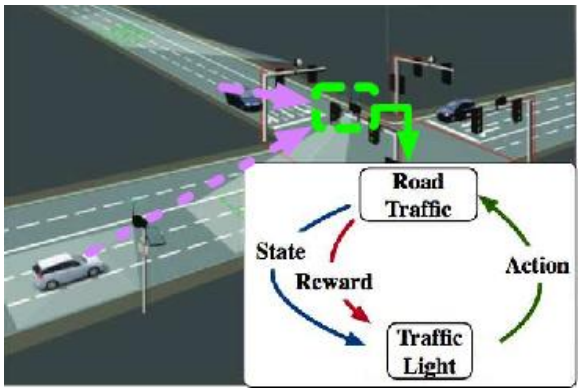
Koneoppiminen voidaan jakaa ohjattuun oppimiseen, ohjaamattomaan oppimiseen ja vahvistusoppimiseen. Vahvistusoppimisen lähestymistapa ongelmiin on hyvin erilainen kuin dataan ja luokkatietoon perustuvassa ohjatussa oppimisessa tai pelkästään datan rakenteeseen perustuvassa ohjaamattomassa oppimisessa. Vahvistusoppimisessa oppiminen tapahtuu reaaliaikaisesti tehtyjen yritysten ja erehdysten avulla [2]. Siinä toimija eli **agentti** yrittää löytää parhaan mahdollisen **liikkeen tiloista**, joihin sillä on mahdollista siirtyä sen toimiessa määrätyssä **toimintaympäristössä**. Agentin valitsemia liikkeitä arvioidaan positiivisten ja negatiivisten **palautteiden** avulla sen mukaan, johtiko liike haluttuun lopputulokseen. Agentin toimiessa toimintaympäristössä palautteiden arvot voidaan tallentaa tila-liike-taulukkoon päivittämällä aina tutkittavan tila-liike parin arvoa käytetyn vahvistusoppimismenetelmän avulla. Pitkän ajan kuluessa agentti oppii **strategian**, jota noudattamalla se kykenee pääsemään päämääräänsä mahdollisimman optimaalisin liikkein. Kuvassa 1 on mallinnettu agentin toimintaa määrätyssä toimintaympäristössä.



Kuva 1. Agentti yrittää oppia tiloille parhaat mahdolliset liikkeet toimiessaan määrätyssä toimintaympäristössä.

Otetaan yksinkertainen esimerkki vahvistusoppimisesta. Erään risteyksen liikennevalojen ohjaamiseen halutaan löytää ratkaisu vahvistusoppimista käyttämällä. Liikennevalojen vaihtamisesta päätöksen tekevä tietokone toimii tässä esimerkissä agenttina. Agentti saa tiedon autojen määrästä risteyksen eri kaistoilla, josta sen on kyettävä sulavasti päästämään autot risteyksen ohi jatkamaan matkaansa. Autojen määrät eri kaistoilla ovat tässä esimerkissä tiloja ja tietokoneen tekemät päätökset, kuten mille kaistoille näytetään vihreää valoa ja kuinka kauan, ovat liikkeitä. Agentin toiminnasta eri tiloissa voidaan antaa palautetta esimerkiksi perustuen autojen odotusaikaan risteyksessä eri ajanhetkillä. Esimerkin toimintaympäristön muodostaa kaikki mahdolliset tilanteet, miten paljon autoja voi kertyä risteyksen eri kaistoille sekä kaikki mahdolliset liikkeet, jotka agentti voi suorittaa kussakin tilanteessa. Alussa agentti toimii lähes satunnaisesti

ja liikenne voi ruuhkautua useita kertoja risteyksen kohdalla. Ajan kuluessa agentti kuitenkin oppii purkamaan ruuhkat palautteiden avulla, jonka jälkeen se pystyy muodostamaan erilaisia strategioita eri tilanteita varten. Kuvassa 3 on esitetty tekstin esimerkkiä havainnollistava kuva, jossa liikennevalot saavat tiedon autoista eri kaistoilla sekä autojen odotusajat, jonka jälkeen tietokoneen tekemä liike vaikuttaa seuraavan tilan ja seuraavan palautteen arvoon [3].



Kuva 2. Liikennevaloja ohjaava tietokone toimii tekstin esimerkissä agenttina, jonka liike vaikuttaa seuraavan tilan ja seuraavan palautteen arvoon.

Agentin oppiessa yritysten ja erehdysten avulla strategian, se hyödyntää aiemmin opittua tietoa (engl. exploitation) hyviin palautteisiin johtavien liikkeiden valitsemiseksi tutkittavasta tilasta. Tämän lisäksi agentilla on mahdollisuus valita myös huonomman palautteen saaneita liikkeitä tai aivan uusia liikkeitä (engl. exploration) ja näin saada kattavampi kuva koko toimintaympäristöstä. Usein opetettaessa strategiaa agentille tasapainoillaan näiden kahden menetelmän välillä esimerkiksi valitsemalla tutkittavalle tilalle yhdeksän kymmenestä kerrasta parhaaksi havaittu liike ja yhden kymmenestä kerrasta sattumanvarainen liike (engl. epsilon-greedy policy).

Eri vahvistusoppimismenetelmät hyödyntävät eri ominaisuuksia. Menetelmä voi olla mallivapaa (engl. model-free), jolloin oletetaan, että tilojen ja liikkeiden välillä ei ole riippuvuuksia, eikä menetelmä mallinna tällöin ympäristöään oppimisen aikana. Menetelmä voi olla myös mallipohjainen (engl. model-based), jolloin menetelmä oppii mallin ympäristöstään. Tämä tarkoittaa sitä, että esimerkiksi kun agentti tekee tilassa S_1 liikkeen A_1 ja saa uuden tilan S_2 sekä palautteen R_1, menetelmä luo mallin ympäristön ilmenemiselle.

Yksi yksinkertaisemmista vahvistusoppimismenetelmistä on **Monte Carlo-menetelmä** [4]. Se on mallivapaa menetelmä, jonka uuden tilan arvon laskemiseksi on odotettava palautetta tapahtuman päätetilaan asti. Päätetila voi olla esimerkiksi yhden Blackjack-kierroksen päättävä siirto tai ajanhetki, kun risteyksen ruuhkatiedosta voidaan antaa uusi palaute autojen ollessa pysähtyneinä. Monte Carlo-menetelmälle käytetään päivityssääntöä, joka on esitetty kaavassa 1.

$$V(S_k) \leftarrow V(S_k) + \alpha \left(R_t - V(S_k)\right) \quad , \tag{1}$$

missä $V(S_k)$ on tutkittavan tilan arvo, R_t on tilan saama palautteen arvo ajanhetkellä t ja α on oppimisnopeus (engl. learning rate), joka kertoo, missä määrin uusi tieto korvaa vanhan tiedon (saa arvoja väliltä [0,1]).

Mikäli tutkittava ympäristö on stationäärinen, oppimisnopeutena voidaan käyttää arvoa $\alpha = \frac{1}{N(S_k)}$, missä $N(S_k)$ laskee, montako kertaa kyseisessä tilassa on vierailtu. Tällöin tilan arvo on sama kuin tilan kaikkien saamien palautteiden keskiarvo.

Otetaan esimerkki Monte Carlo-menetelmän toiminnasta. Eräälle tilalle S_1 saadaan ajan kuluessa palautteiden arvot R_t= [4,3,2,1] ja oppimisnopeus α on 0,5. Kaavan 1 avulla saadaan laskettua tilalle S_1 arvot eri ajan hetkillä:

t=0, $V_0(S_1) = 0$

t=1, R_1=4, $V_1(S_1) = 0 + \frac{1}{2}(4-0) = 2$

t=2, R_2=3, $V_2(S_1) = 2 + \frac{1}{2}(3-2) = \frac{5}{2}$

t=3, R_3=2, $V_3(S_1) = \frac{5}{2} + \frac{1}{2}(2-\frac{5}{2}) = \frac{9}{4}$

t=4, R_4=1, $V_4(S_1) = \frac{9}{4} + \frac{1}{2}(1-\frac{9}{4}) = \frac{13}{8}$

Toinen erittäin tunnettu vahvistusoppimismenetelmä on **Q-oppiminen**. Se on Monte Carlo-menetelmän tapaan mallivapaa menetelmä, jossa ns. Q-arvo päivitetään heti tilasta tehdyn liikkeen jälkeen tila-liike-taulukkoon. Palautetta ei siis tarvitse odottaa koko tapahtuman päättymiseen asti, kuten Monte Carlo-menetelmän kohdalla. Q-oppimiselle käytetään päivityssääntöä, joka on esitetty kaavassa 2.

$$Q(S_k, A_k) \leftarrow Q(S_k, A_k) + \alpha\left(R_t + \gamma\left(\max\left(Q(S_{k+1}, A_k)\right)\right) - Q(S_k, A_k)\right), \quad (2)$$

missä $Q(S_k, A_k)$ on tila-liike parin Q-arvo, $max(Q(S_{k+1}, A_t))$ on seuraavien tila-liike parien suurin Q-arvo ja γ on ns. vaimennuskerroin (engl. discount factor), joka määrittää, miten paljon tulevien palautteiden arvoja arvotetaan nykyisiin palautteiden arvoihin verrattuna (saa arvoja väliltä [0,1]).

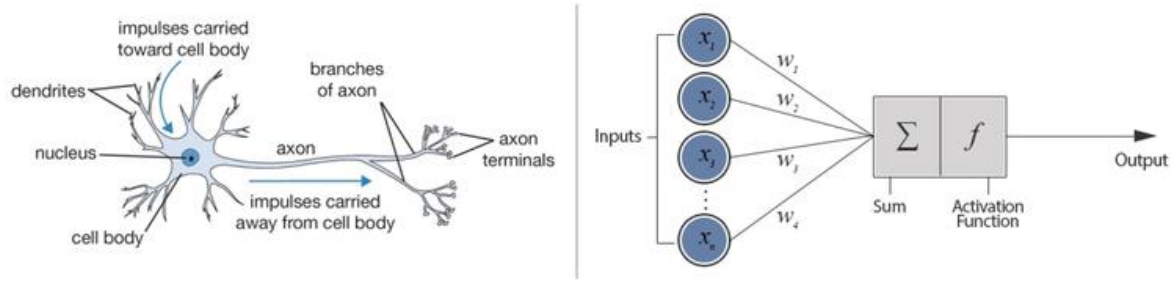
Huomataan kaavasta 2, että mikäli vain nykyisillä tila-liike parien palautteiden arvoilla on merkitystä eli vaimennuskerroin γ on 0, muistuttaa supistunut Q-oppimisen päivityssääntö hyvin paljon kaavaa 1. Tämä on esitetty kaavassa 3.

$$Q(S_k, A_k) \leftarrow Q(S_k, A_k) + \alpha\left(R_t - Q(S_k, A_k)\right) \quad (3)$$

Syväoppiminen

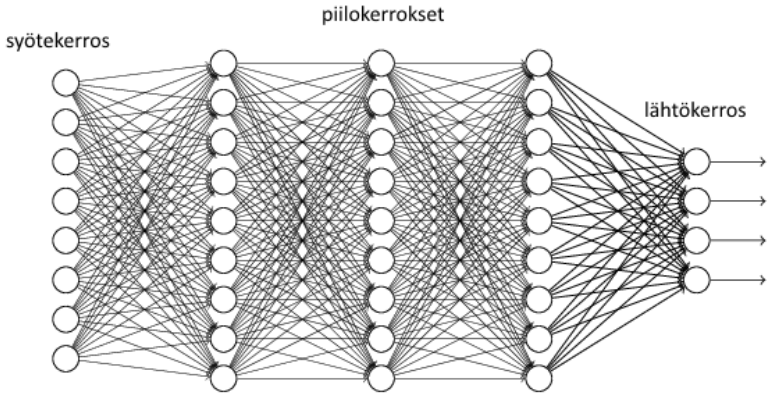
Syväoppiminen on tällä hetkellä yksi tekoälyn tutkituimmista osa-alueista, jonka laskennassa käytettyjen monikerrosneuroverkkojen kehitystä ajan saatossa ovat innoittaneet ihmisaivojen rakenne ja toiminta. Syväoppimista on tutkittu jo 1960-luvulta lähtien mutta täydellinen läpimurto saavutettiin vasta 2010-luvulle tultaessa, kun massamuistien kapasiteetti ja näytönohjaimien laskentateho kasvoivat merkittävästi. Syväoppimista voidaan soveltaa kaikkiin tekoälyn osa-alueisiin, vaikkakin sitä on yleisimmin käytetty luokittelussa.

Keinotekoiset neuroverkot koostuvat ihmisaivojen tapaan neuroneista, jotka ovat yksinkertaisia toisiinsa kytkettyjä tiedonkäsittely-yksiköitä. Biologisessa neuronissa sen tulohaarakkeet (engl. dendrities) vievät tiedon sähköisenä impulssina kohti solukeskusta (engl. nucleus), josta viejähaarake (engl. axon) kuljettaa käsitellyn tiedon eteenpäin. Vastaavalla tavalla keinotekoisessa neuronissa eri painokertoimilla kerrotut syötteet summataan yhteen summaajassa, jonka jälkeen summalle lasketaan aktivointifunktiolla, kuten sigmoid-funktiolla tai ReLU-funktiolla (engl. rectified linear unit), epälineaarinen kuvaus [5]. Jos aktivointifunktiona käytetään kynnystystä (engl. threshold), käytetään neuronista nimitystä perceptroni. Kuvassa 3 esitetyt biologisen ja keinotekoisen neuronin rakenteet muistuttavat toisiaan.



Kuva 3. Biologisen ja keinotekoisen neuronin rakenteet muistuttavat toisiaan.

Keinotekoisista neuroneista rakennettu kokonainen neuroverkko koostuu syötekerroksesta, lähtökerroksesta sekä niiden välissä olevista piilokerroksista kuvan 4 mukaisesti. Neuroverkkojen opettaminen tarkoittaa käytännössä verkossa olevien neuronien painokertoimien säätämistä. Yksittäinen datanäyte tulee sisään syötekerrokselle, josta informaatio etenee piilokerrosten kautta lähtökerrokselle. Kun näytteen oikea luokka on tiedossa, voidaan painokertoimia säätämällä saada se vastaamaan verkon avulla ennustettua luokkaa. Tämä tapahtuu vastavirta-algoritmilla (engl. backpropagation) kulkemalla verkossa taaksepäin lähtökerroksesta syötekerrokselle minimoiden kustannusfunktion gradientin, jolloin saadaan säädettyä neuroverkon painokertoimia optimaalisemmiksi.



Kuva 4. Neuroverkko koostuu syötekerroksesta, lähtökerroksesta sekä piilokerroksista.

Syöttämällä datajoukon näytteet monia kertoja neuroverkon läpi, oppii se lopulta datan perusteella optimaalisimmat painokertoimet. Mikäli dataa on liian vähän ja käytetyssä neuroverkossa on paljon neuroneja, on vaarana ylioppiminen. Vastaavasti jos erittäin kompleksiseen dataan käytetään liian yksinkertaista neuroverkkoa, ei datasta pystytä oppimaan tarpeeksi hyvin sen eri ominaisuuksia ja tapahtuu alioppiminen.

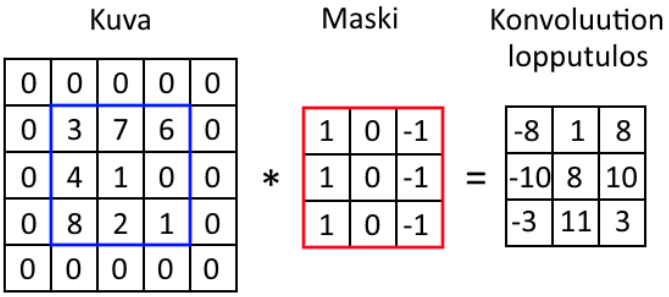
Tarkastellaan seuraavaksi tarkemmin erästä erittäin tunnettua syväoppimisen arkkitehtuuria nimeltä konvoluutioneuroverkot (engl. convolutional neural networks, CNN), joita käytetään pääasiassa kuvien luokitteluun ja tunnistamiseen [6]. Konvoluutioneuroverkossa kuvien pikseleiden arvoja käsitellään aluksi konvoluutiokerroksissa, josta tuotetut piirteet kulkevat koko neuroverkon läpi aina lähtökerrokselle, jossa kuvalle ennustetaan luokka. Konvoluutioneuroverkkojen kerrokset sisältävät normaalisti konvoluutiokerroksia (engl. convolutional layers), alinäytteistyskerroksia (engl. pooling layers), täysin-kytkettyjä kerroksia (engl. fully-connected layers) sekä mahdollisesti ns. neuronien-pudotuskerroksia (engl. drop-out layers).

Yksittäisessä konvoluutiokerroksessa lasketaan koko kuvalle konvoluutio liikuttamalla maskia jokaisessa kohtaa kuvaa samalla laskien konvoluutio-operaatio maskin ja kuvan kohdan välillä. Maskien tehtävä on

löytää kuvasta piirteitä ja usein konvoluutiokerroksissa käytetäänkin monia erilaisia maskeja monipuolisten piirteiden löytämiseksi. Kuvassa 5 on havainnollistettu 3x3 pikselin kokoisen maskin ja 3x3 pikselin kokoisen alkuperäisen kuvan konvoluution lopputulos, missä maskin siirtymä x-akselin ja y-akselin suunnassa on yksi pikseliä. Alkuperäisen kuvan ympärille on lisätty nollarivi (engl. zero-padding), sillä konvoluutio suuremmilla kuin 1x1 kokoisen pikselin maskeilla pienentää lopputuloksen korkeutta ja leveyttä. Näin kuvan alkuperäinen koko säilyy operaation jälkeen ennallaan. Konvoluution lopputuloksen koko saadaan laskettua kaavalla 4.

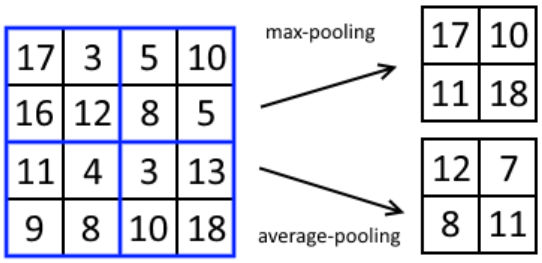
$$O = \frac{W - F + 2P}{S} + 1 \quad , \tag{4}$$

missä W on maskin leveys, F on alkuperäisen kuvan leveys, P on lisätyn nolla rivin leveys, S on siirtymä ja O on konvoluution lopputuloksen leveys.



Kuva 5. Esimerkki konvoluutiokerroksen toiminnasta. Alkuperäisen kuvan koko on 3x3 pikseliä (sininen reunus), jonka ympärille on lisätty yhden pikselin levyinen nollarivi. Maskin koko on 3x3 pikseliä (punainen reunus). Kun siirtymä x-akselin ja y-akselin suunnassa on 1 pikseliä, niin lopputuloksen kooksi saadaan laskettua kaavan 4 perusteella 3x3 pikseliä.

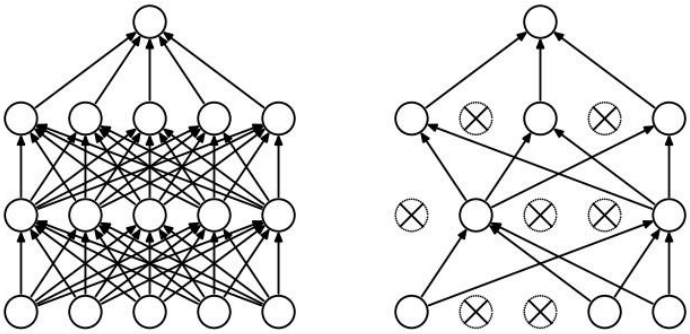
Alinäytteistyskerros sijoitetaan neuroverkossa heti konvoluutiokerroksen jälkeen. Sen tehtävä on löytää konvoluution tuottamasta lopputuloksesta sen tärkeimmät ominaisuudet tiivistäen informaatiota samalla pienentäen ylioppimisen riskiä. Yksi alinäytteistystekniikka on alinäytteistys maksimiarvoilla (engl. max-pooling), jossa ruudukolla jaettujen alueiden sisältä valitaan jokaisen alueen suurin arvo. Toinen vaihtoehto on käyttää alinäytteistystä keskiarvoilla (engl. average-pooling), jolloin ruudukolla jaettujen alueiden sisältämistä arvoista lasketaan keskiarvot. Kuvassa 6 on esitetty molemmat tapaukset 4x4 pikselin kokoiselle kuvalle, joka on jaettu 2x2 pikselin kokoisiin alueisiin.



Kuva 6. Alinäytteistys kuvalle max-pooling ja average-pooling tekniikoilla.

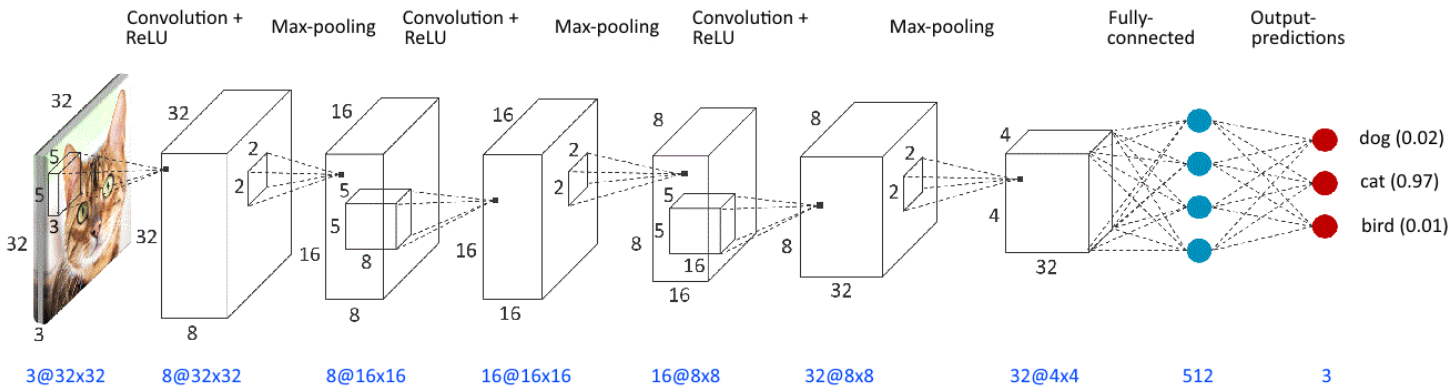
Konvoluutioneuroverkkojen luokittelu tapahtuu lopuksi täysin-kytketyissä kerroksissa vastaavalla tavalla kuin kuvassa 4. Konvoluutiokerrosten ja alinäytteistyskerrosten läpi kulkenut taulukkomuotoinen data muunnetaan aluksi yksiulotteiseksi listaksi. Seuraavaksi listassa olevat numeroarvot syötetään suoraan syötekerrokselle, josta ne muunnetaan mahdollisten piilokerrosten kautta ennustetuksi luokkatiedoksi lähtökerroksella.

Ylioppiminen on erittäin yleinen ongelma myös konvoluutioneuroverkkojen kohdalla. Ongelman välttämiseksi voidaan käyttää erilaisia neuronien-pudotuskerroksia tai normalisointimenetelmiä. Neuronien-pudotuskerroksissa tietty määrä sattumanvaraisesti valittuja neuroneja pudotetaan pois neuroverkosta kuvan 7 tyyliin. Näin sekä verkon kompleksisuus että neuroverkon riski ylioppia pienenevät. Neuronien-pudotuskerroksia käytetään usein heti alinäytteistyskerrosten jälkeen.



Kuva 7. Vasemmalla puolella on alkuperäinen neuroverkko ja oikealla puolella neuronien pudotuksen jälkeinen neuroverkko.

Lopuksi esitetyistä kerrostyypeistä voidaan muodostaa kokonaisia konvoluutioneuroverkkoja. Kuvassa 8 on esitetty erään konvoluutioneuroverkon rakenne 32x32 pikselin kokoisille RGB-kuville, jota käyttämällä halutaan tunnistaa, onko kuvassa koira, kissa vai lintu. Neuroverkko sisältää kolme peräkkäistä konvoluutiokerroksen, aktivointifunktion ja alinäytteistyskerroksen muodostamaa lohkoa, jonka jälkeen kuvasta muunnetut piirteet menevät täysin-kytkettyjen kerrosten kautta lähtökerrokselle luokiteltavaksi. Kuvan 8 konvoluutioneuroverkon opettamisen jälkeen syötekerrokselle tuleva testikuva kissasta tunnistetaan 97 prosentin varmuudella luokkaan kissa, kahden prosentin varmuudella luokkaan koira ja yhden prosentin varmuudella luokkaan lintu.



Kuva 8. Erään 32x32 pikselin kokoisille RGB-kuville muodostetun konvoluutioneuroverkon rakenne.

Tehdessäsi harjoitusta omalla tietokoneella, asenna tensorflow ja keras python-kirjastot tehtävää 2 varten tietokoneellesi seuraavasti:

Mene komentoriville ja aja seuraava komento:

pip install tensorflow keras

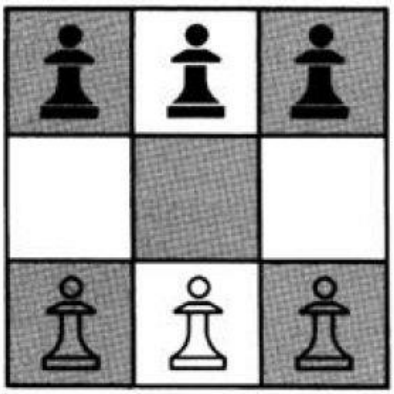
Toinen vaihtoehto on asentaa tensorflow ja keras anacondan avulla seuraavasti (install tensorflow):

https://medium.com/@potassium_hydride/install-tensorflow-and-keras-with-anaconda-for-pycharm-on-windows-step-5fa33b38fd9

Tehtävä 1 (2.5p)

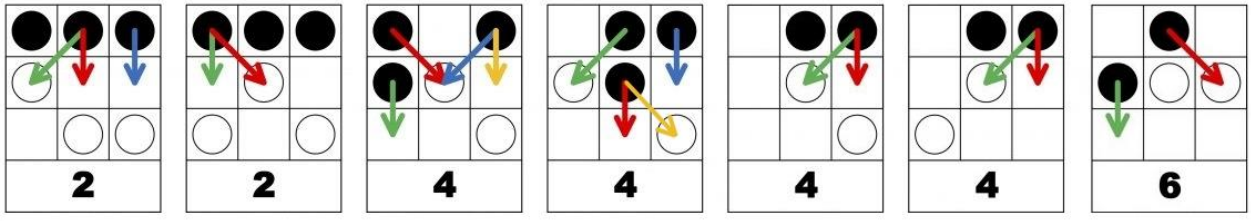
Tässä tehtävässä sinun tulee luoda Q-arvoille päivityssääntö vahvistusoppimista käyttävään peliin nimeltä Hexapawn sekä testata luomaasi päivityssääntöä eri oppimisnopeuksien α arvoilla.

Hexapawn on kahden pelaajan pelattava shakin johdannaispeli, jota pelataan 3x3 kokoisella pelilaudalla. Pelin alkutilanteessa molemmilla pelaajilla on kolme sotilasta omilla puolillaan kuvan 9 osoittamalla tavalla ja valkoinen pelaaja aloittaa aina pelin. Vain kahden tyyppiset siirrot pelissä ovat sallittuja: (1) Sotilas voi liikkua suoraan eteenpäin vain yhden ruudun kerrallaan, mikäli sen edessä oleva ruutu on tyhjä. (2) Sotilas voi syödä vastustajan sotilaan liikkumalla vinottaisesti joko vasemmalle tai oikealle, mikäli vastustajan sotilas on kyseisessä ruudussa. Shakin tapaan syödyt sotilaat poistetaan pelilaudalta. Pelin voi voittaa kolmella eri tavalla: (1) Siirtämällä oma sotilas kolmannelle riville. (2) Syömällä kaikki vastustajan sotilaat. (3) Saavuttamalla tilanne, jossa vastustaja ei voi liikuttaa mitään sotilaistaan. Lisäksi molempien pelaajien liikkuessa vuorotellen peli ei voi päättyä tasan. Mikäli hexapawn-pelissä molemmat pelaajat pelaavat täydellisesti, aloittava pelaaja eli valkoinen häviää aina pelin.



Kuva 9. Hexapawn-pelin alkutilanne.

Vahvistusoppimismenetelmänä käytetään toteutuksessa kaavan 3 mukaista supistettua Q-oppimismenetelmää (vaimennuskerroin γ = 0), joka oppii pelejä pelatessaan optimaalisimmat siirrot tekoälylle eli mustalle pelaajalle. Mitä paremmin pelin aloittava pelaaja eli valkoinen pelaa, sitä nopeammin luonnollisesti tekoäly saavuttaa voittamattoman pelistrategian. Täydellisen strategian saavuttamiseksi tekoälyn on pelattava siirtonsa oikein 7 kriittisessä tilassa, jotka on esitetty kuvassa 10. Kuvan kriittisten tilojen alapuolella on myös esitetty numeroin, monesko liike alkutilasta lähtien on kyseessä sekä nuolin tekoälyn mahdolliset liikkeet.



Kuva 10. Kriittiset tilat mustalle eli tekoälylle täydellisen strategian saavuttamiseksi Hexapawn-pelissä.

Sinulle on jo annettu valmiiksi tiedostossa **hexapawn.py** graafisesti toteutettu peli, tiedosto **hexapawnstates.py**, jossa selvitetään tila-liike-taulukkoa varten sallitut tilat ja liikkeet sekä tiedosto **hexapawnsimulation.py**, jota käytetään lopuksi arvioimaan oppimisnopeuden arvon vaikutusta oppimiseen.

Sinun tulee toteuttaa tiedostossa hexapawn.py päivityssääntö kaavan 3 mukaisesti funktiossa *QLearning(self, Q, stateaction, alpha, reward)*. Funktio saa argumentteinaan Q-arvot kaikille tila-liike pareille taulukossa *Q*, päivitettävän tila-liike parin *stateaction*, oppimisnopeuden arvon *alpha* sekä palautteen arvon *reward*. Voit viitata tietyn tila-liike parin Q-arvoon python komennolla `Q[stateaction]`. Toteuttamasi funktion tulee palauttaa päivitetty tila-liike parin Q-arvo.

Toteutettuasi päivityssäännön, pelaa tekoälyä vastaan ajamalla komentoriviltä tiedosto hexapawn.py. Ajaessasi tiedoston, sinulle avautuu kolme ikkunaa, joista ensimmäisessä on visualisoitu taulukossa tekoälyn päivittämät Q-arvot kaikille tila-liike pareille. Toisessa ikkunassa on arvioitu, kuinka hyvin tekoäly suoriutuu oppimiensa Q-arvojen perusteella kuvassa 10 esitetyistä 7 kriittisestä tilasta. Toisen ikkunan kuvaajien otsikkoihin tallentuu tieto, valitseeko tekoäly aina kriittisessä tilassa oikean liikkeen ja kuvaajien alapuolille tallentuu laillisten liikkeiden sen hetkiset Q-arvot. Kolmas ikkuna on itse pelaamista varten, jonka otsikkoon tulostuu pelitilanne ja alapuolelle tieto siitä, pelaako tekoäly täydellisesti. Pelaat valkoisilla, joten valitse siirrettävä sotilas klikkaamalla sitä kerran sekä kohta, johon haluat siirtää sen klikkaamalla siihen.

```
python hexapawn.py
```

Oppimisnopeutena käytetään graafisessa toteutuksessa vakiona arvoa 1,0. Valkoisen pelaajan voittaessa pelin, annetaan palautteen arvoksi -5. Tekoälyn eli mustan pelaajan voittaessa pelin annetaan palautteen arvoksi +5. Pelatessasi tarpeeksi monta kertaa tekoälyä vastaan, oppii se saamiensa palautteiden perusteella liikkeet, joita sen tulee välttää sekä liikkeet, joita sen tulee suosia. Tässä toteutuksessa päivittäessä Q-arvoja päivitetään samalla myös symmetriset pelitilanteet tekoälyn saavuttaakseen täydellisen pelistrategian mahdollisimman nopeasti.

Tekoälyn opittua täydellisen pelistrategian, tallentuu pelin ikkunat kuviin **criticalstates.png**, **stateactionmap.png** ja **perfectgame.png**. Huomaa, että kuvan 10 vasemmalta katsottuna kolmannen kriittisen tilan kannalta selvästi ainut oikea vaihtoehto on voittaa peli, minkä oppimiseen tekoälyn suorittaessa aluksi sattumanvaraisia siirtoja voi mennä aikaa tovi. Tämän jälkeen testaa oppimisnopeuden arvon vaikutusta oppimisprosessiin ajamalla python-tiedosto hexapawnsimulation.py seuraavasti:

`python hexapawnsimulation.py -a=1.0` ,

missä -a viittaa oppimisnopeuden arvoon

Tiedostossa hexapawnsimulation.py vahvistusoppimista käyttävä tekoäly pelaa sattumanvaraisia siirtoja tekevää vastustajaa vastaan niin kauan, kunnes tekoäly pelaa täydellisesti. Tämä prosessi toistetaan iteratiivisesti 100 kertaa, lasketaan keskiarvo ja tulostetaan komentoriville, kuinka monta peliä oli pelattava keskimäärin täydellisen pelin saavuttamiseksi. Nyt voidaan testata tälle toteutukselle eri oppimisnopeuksien arvojen vaikutusta pelattujen pelien määrän.

KYSYMYS1: Aja tiedosto hexapawnsimulation.py oppimisnopeuksien arvoilla 0, 0.5 ja 1.0 ja raportoi, miten ne vaikuttivat pelattujen pelien määrään ennen tekoälyn oppimista täydellisen pelistrategian. Oliko nollaa suuremmilla oppimisnopeuksien arvoilla merkittävästi vaikutusta oppimisprosessiin?

KYSYMYS2: Kerro yleisesti ottaen mitä tarkoittaa, kun oppimisnopeuden arvo on 0 ja kun oppimisnopeuden arvo on 1? Voit sijoittaa tutkittavan oppimisnopeuden arvon kaavaan 3 ja supistaa lauseketta selvittäessäsi vastausta kysymykseen.

Palauta muokkaamasi python-tiedosto hexapawn.py sekä kirjoita tehtävästä 1 lyhyt raportti, jossa vastaat kysymyksiin KYSYMYS1 ja KYSYMYS2. Sisällytä raporttiin myös kuvat criticalstates.png, stateactionmap.png ja perfectgame.png.

Tehtävä 2 (2.5p)

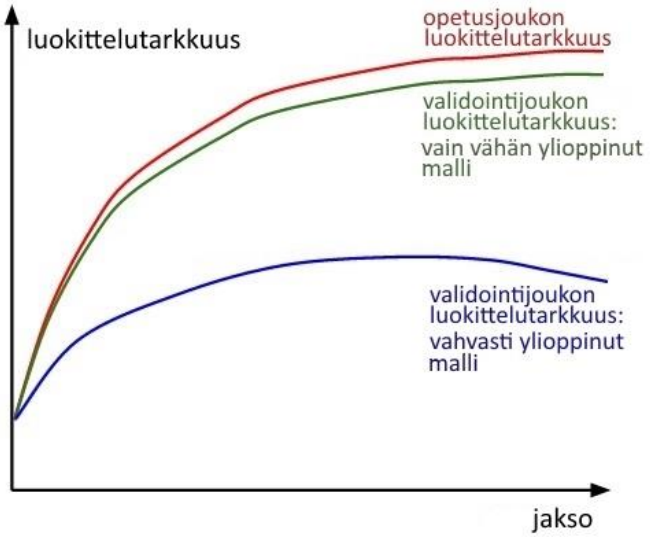
Tässä harjoituksessa sinun tulee luoda ja opettaa kaksi konvoluutioneuroverkkoa, joiden tehtävä on luokitella CIFAR10-datajoukon kuvat oikeisiin luokkiin [7].

CIFAR10-datajoukko sisältää 60 000 32x32 pikselin kokoista RGB-kuvaa objekteista, kuten autoista, linnuista, kissoista ja niin edelleen. Datajoukossa on yhteensä 10 luokkaa, jolloin yhdestä luokasta on yhteensä 6000 kuvaa. Kuvassa 11 on esitetty esimerkkikuvia datajoukon CIFAR10 sisällöstä.

Kuva 11. Esimerkkikuvia CIFAR10-datajoukon sisällöstä.

Syväoppimisessa opettaminen tapahtuu ajamalla opetusjoukon kuvat monia kertoja opetettavan neuroverkon läpi. Kun jokainen opetusjoukon kuva on ajettu yhden kerran neuroverkon läpi, on kulunut yksi jakso (engl. epoch). Konvoluutioneuroverkon tapauksessa kuvat ajetaan neuroverkon läpi erissä (engl. batch), jonka koko kertoo, montaako kuvaa käytetään yhden iteraation aikana. Jos esimerkiksi opetusjoukossa on 1024 kuvaa ja erän koko on 8 kuvaa, ajetaan yhden jakson suorittamiseksi yhteensä 128 erää.

Syväoppimisessa voidaan arvioida opetettavan mallin oppimista seuraamalla, miten validointijoukon ja opetusjoukon luokittelutarkkuudet kehittyvät opetuksen aikana. Yleisesti ottaen opetusjoukon luokittelutarkkuuden ja validointijoukon luokittelutarkkuuden välinen ero on suoraan verrannollinen ylioppimisen määrään. Kuvassa 12 tätä ilmiötä on havainnollistettu kuvaajan avulla. Sininen käyrä kuvaa validointijoukon luokittelutarkkuutta ja sen arvo jää opetuksen edetessä paljon pienemmäksi kuin punaisen käyrän osoittama opetusjoukon luokittelutarkkuus. Tällöin opetettu malli on aivan liian kompleksinen, jolloin ylioppiminen on erityisen voimakasta. Yksinkertaistamalla opetettua konvoluutioneuroverkkoa validointijoukon luokittelutarkkuus on vihreän käyrän mukaisesti koko opetusprosessin ajan vain vähän pienempi kuin opetusjoukon luokittelutarkkuus, jolloin ylioppiminen on vain vähäistä. Vastaavasti jos validointijoukon luokittelutarkkuus on yhtä suuri tai suurempi kuin opetusjoukon luokittelutarkkuus, on opetettavan konvoluutioneuroverkon rakenne liian yksinkertainen ja tapahtuu alioppiminen, jolloin on monimutkaistettava konvoluutioneuroverkon rakennetta.



Kuva 12. Opetusjoukon ja validointijoukon luokittelutarkkuuksien välinen ero on suoraan verrannollinen ylioppimisen määrään.

Taulukossa 1 on esitetty tehtävässä käytettävien konvoluutioneuroverkkojen rakenteet. Sinulle on jo annettu tiedostossa **convolutionalneuralnetwork.py** valmiiksi mallin 1 toteutus esimerkkinä funktiossa *model1()* ja tehtäväksesi jää toteuttaa taulukossa 1 esitetty mallin 2 konvoluutioneuroverkon rakenne.

Toteuta mallin 2 konvoluutioneuroverkon rakenne funktiossa *model2()* ja aja tiedosto convolutionalneuralnetwork.py sekä mallille 1 ja mallille 2. Aivan aluksi datajoukko jaetaan 50 000 opetusjoukon näytteeseen ja 10 000 validointijoukon näytteeseen. Tämän jälkeen valittua mallia opetetaan 10 jakson verran. Opetettavan mallin validointijoukon ja opetusjoukon luokittelutarkkuudet yhden jakson välein tallennetaan käyrinä kuvaajaan ja kuvaan nimeltä **<**modelname**>_accuracies.png**. Kuvaajan alapuolelle tallennetaan sekä komentoriville printataan lopullisen mallin validointijoukon ja opetusjoukon luokittelutarkkuudet. Lopuksi vielä ennustetaan validointijoukon 15 ensimmäiselle näytteelle luokat pylväsdiagrammeilla havainnollistaen, millä varmuudella konvoluutioneuroverkko luokittelee näytteet eri luokkiin. Tämä tallennetaan kuvaan nimeltä **<**modelname**>_predictions.png.** Huomaa, että yhden mallin opettamiseen voi kulua aikaa tietokoneesta riippuen 30-60 minuuttia.

```
python convolutionalneuralnetwork.py -m=model1
```
,

missä -m viittaa opetettavaan malliin. Vaihtoehdot ovat joko model1 tai model2.

KYSYMYS1: Vertaamalla opetettujen mallien model1 ja model2 lopullisia validointijoukon luokittelutarkkuuksia, kumpi malleista on parempi?

KYSYMYS2: Kerro tallennettujen kuvaajien perusteella 10 jakson opettamisen jälkeen molemmille malleille, onko malli mahdollisesti ylioppinut vai alioppinut?

Palauta muokkaamasi python-tiedosto convolutionalneuralnetwork.py sekä kirjoita tehtävästä 2 lyhyt raportti, jossa vastaat kysymyksiin KYSYMYS1 ja KYSYMYS2. Sisällytä raporttiin kuvat <modelname>_accuracies.png ja <modelname>_predictions.png molemmille malleille.

Taulukko 1. Tehtävässä käytettävien konvoluutioneuroverkkojen rakennekaaviot.

Malli 1	Malli 2
RGB-kuva (koko: 32x32x3)	RGB-kuva (koko: 32x32x3)
3x3 konvoluutio, 32 kerrosta, nollarivien lisäys (padding)	3x3 konvoluutio, 32 kerrosta, nollarivien lisäys (padding)
ReLU-aktivointifunktio	ReLU-aktivointifunktio
3x3 konvoluutio, 32 kerrosta	3x3 konvoluutio, 32 kerrosta
ReLU-aktivointifunktio	ReLU-aktivointifunktio
2x2 max-pooling alinäytteistys	3x3 konvoluutio, 32 kerrosta
25% neuronien-pudotuskerros (dropout)	ReLU-aktivointifunktio
	2x2 max-pooling alinäytteistys
3x3 konvoluutio, 64 kerrosta, nollarivien lisäys (padding)	
ReLU-aktivointifunktio	3x3 konvoluutio, 64 kerrosta, nollarivien lisäys (padding)
3x3 konvoluutio, 64 kerrosta	ReLU-aktivointifunktio
ReLU-aktivointifunktio	3x3 konvoluutio, 64 kerrosta
2x2 max-pooling alinäytteistys	ReLU-aktivointifunktio
25% neuronien-pudotuskerros (dropout)	3x3 konvoluutio, 64 kerrosta
	ReLU-aktivointifunktio
Syötekerros (flatten-layer)	2x2 max-pooling alinäytteistys
512 neuronin piilokerros (dense)	
ReLU-aktivointifunktio	Syötekerros (flatten-layer)
25% neuronien-pudotuskerros (dropout)	1024 neuronin piilokerros (dense)
	ReLU-aktivointifunktio
10 neuronin lähtökerros (dense)	100 neuronin piilokerros (dense)
Softmax-aktivointifunktio	ReLU-aktivointifunktio
	10 neuronin lähtökerros (dense)
	Softmax-aktivointifunktio

Lähteet

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

[2] Sutton R. S. & Barto A. G. (1998). Reinforcement learning: An introduction. MIT press.

[3] Liang X., Du X., Wang G. & Han Z. (2018) Deep reinforcement learning for traffic light control in vehicular networks. arXiv preprint arXiv:1803.11115.

[4] Silver D. Reinforcement learning lectures. URL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

[5] Nasrabadi N. M. (2007) Pattern recognition and machine learning. Journal of electronic imaging, 16(4), 049901.

[6] LeCun Y., Bengio Y. & Hinton G. (2015) Deep learning. Nature, 521(7553), 436.

[7] Krizhevsky, A., Nair, V., & Hinton, G. (2014). The CIFAR-10 dataset. URL: http://www.cs.toronto.edu/kriz/cifar.

Harjoituksen 5 oppimistavoitteet

- ymmärtää vahvistusoppimiseen liittyviä peruskäsitteitä kuten Q-arvo ja oppimisnopeus sekä pääpiirteittäin sen toimintatavan
- osaa muodostaa käytettävälle vahvistusoppimismenetelmälle päivityssäännön
- osaa luoda ja opettaa yksinkertaisia neuroverkkoja sekä tulkita opetusprosessin etenemistä

Palauta

Palauta muokkaamasi python-tiedostot ja raportit (.zip tai .rar tiedostoon pakattuna) Optiman palautuslaatikkoon Harjoitus 5 **30.4.2019 klo 23:59** mennessä. Tästä harjoituksesta on mahdollisuus tienata maksimissaan 5 pistettä (2.5p + 2.5p). Voit antaa palautetta tästä harjoituksesta liittämällä raporttiin erillisen osion palautetta varten. Harjoituksia tullaan kehittämään palautteen pohjalta tuleville vuosille vastaamaan paremmin oppimistavoitteita.

Huomaa, että kurssin läpäisemisen kannalta neljä harjoitusta viidestä tulee suorittaa hyväksytysti. Lisäksi, koska kaikista harjoituksista on mahdollista tienata yhteensä 20 pistettä, pyöristetään tämän pistemäärän ylitykset 20 pisteeseen.