Konvoluutioneuroverkko - sovelluksena DNA-sekvenssien luokittelu

Tilastotieteen pro gradu -tutkielma

Syksy 2022
Joona Purkamo
Matematiikan ja tilastotieteen laitos
Jyväskylän yliopisto

JYVÄSKYLÄN YLIOPISTO
Matematiikan ja tilastotieteen laitos
Purkamo, Joona: *Konvoluutioneuroverkko - sovelluksena DNA-sekvenssien luokittelu*
Tilastotieteen pro gradu -tutkielma, 56 sivua, 2 liitettä (4 sivua)
25. marraskuuta 2022

Tiivistelmä

Luokitteluongelman ratkaisussa tavoitteena on määrätä havainto kuuluvaksi johonkin tiedossa olevaan luokkaan. Erilaisia luokittelumenetelmiä on paljon, mutta erityisesti viime vuosina syväoppimismenetelmät ovat osoittautuneet soveltuvan erinomaisesti luokitteluongelmien ratkaisuksi monissa sovelluksissa. Tässä tutkielmassa esitellään täysin kytketty neuroverkko ja konvoluutioneuroverkko (CNN) luokitteluongelman ratkaisuna sekä sovelletaan CNN:ää hyönteislajien tunnistuksessa, jossa lajit luokitellaan niiden DNA-sekvenssien perusteella.

Neuroverkkojen tarkastelu aloitetaan täysin kytketystä neuroverkosta. Aluksi esitetään neuroverkkojen perusidea, keskeistä termistöä, toiminta luokittelutilanteessa, rakenne sekä neuroverkon parametrien estimointi stokastista gradienttimenetelmää, neuroverkon virhefunktiota ja vastavirta-algoritmia käyttämällä. Vastavirta-algoritmi on laaja kokonaisuus erilaisia toimintoja, joiden tehtävä on päivittää neuroverkon parametrit siten, että neuroverkon virhe minimoituu. Vastavirta-algoritmi monesti ohitetaan sen monimutkaisuuden vuoksi, mutta tässä tutkielmassa se esitetään yksityiskohtaisesti.

CNN:n yhteydessä esitetään ensin kaksi tapaa esikäsitellä aineistoa luokittelua varten: Ensimmäinen tapa liittyy tavallisten kuvien luokitteluun, kun taas toinen tapa liittyy DNA-sekvenssiluokittelun sovellukseen. Näin saadaan kaksi erilaista CNN-luokitinta: yksi- ja kaksiulotteinen CNN. Lisäksi CNN:n yhteydessä esitetään huolellisesti CNN:n rakenne ja sen matemaattiset operaatiot väri- ja harmaasävykuvien sekä DNA-sekvenssien luokittelussa.

Käytetty DNA-sekvenssiaineisto sisältää tuhansia DNA-sekvenssejä tuhansilta eri hyönteislajeilta. Analyysivaiheessa mukaan valitaan vuorollaan sellaiset 10, 100, 500 tai 1000 lajia, joista DNA-sekvenssejä on saatavilla eniten. DNA-sekvenssit esikäsitellään kahdella esitetyllä tavalla, jolloin varsinainen luokittelu tehdään käyttäen sekä yksi- että kaksiulotteista CNN:ää. Luokittelutarkkuuksien ja neuroverkon parametrien estimointiin tarvittavan ajan perusteella yksiulotteinen versio osoittautui paremmaksi DNA-sekvenssien luokittelumenetelmäksi. Luokittelutarkkuudet testiaineistolla nousivat parhaimmillaan noin 95–100 % välille valitusta lajien lukumäärästä riippuen.

Avainsanat: DNA-sekvenssi, k-mer, konvoluutioneuroverkot, lajien tunnistus, neuroverkot, vastavirta-algoritmi

Sisällys

1 Johdanto **1**

2 Genomiikka, geenit ja DNA **4**
2.1 Genomiikka 4
2.2 DNA:n rakenne 4
2.3 PROTAX-aineisto ja DNA-viivakoodaus 5

3 Neuroverkot **7**
3.1 Täysin kytketyt eteenpäin syöttävät neuroverkot 7
3.2 Painot ja vakioneuronit 9
3.3 Aktivaatiofunktiot 11
3.3.1 Piilokerrosten aktivaatiofunktiot 11
3.3.2 Ulostulokerroksen aktivaatiofunktiot 12

4 Neuroverkon opettaminen **14**
4.1 Opetus-, testi- ja validointiaineistot 14
4.2 Ristientropia ja neuroverkon virhefunktio 16
4.3 Stokastinen gradienttimenetelmä 17
4.4 Vastavirta-algoritmi 18
4.4.1 Etenemisvaihe 19
4.4.2 Peruutusvaihe: ketjusääntö ja dynaaminen ohjelmointi 19
4.4.3 Peruutusvaihe: vektorikeskeinen neuroverkkorakenne 24
4.5 Neuroverkon opetukseen liittyviä ongelmia 27
4.5.1 Ali- ja ylioppiminen 27
4.5.2 Katoavan ja räjähtävän gradientin ongelma sekä sammuneet neuronit 29

5 Konvoluutioneuroverkot **32**
5.1 Harmaasävy- ja värikuvien tensoriesitykset 32
5.2 DNA-sekvenssien tensoriesitys 34
5.3 Konvoluutio 36
5.4 Tiivistys 42
5.5 Konvoluutioneuroverkon rakenne 43
5.6 Vastavirta-algoritmi konvoluutioneuroverkossa 46

6 Aineiston analyysi **47**

7 Yhteenveto ja pohdintaa **50**

Viitteet **57**

Liitteet **62**

1 Johdanto

Kun tarkastellaan nimeöityjen aineistojen (*labeled data*) luokitteluongelmia, aineisto vaikuttaa oleellisesti siihen, kuinka onnistuneesti havainnot pystytään luokittelemaan. Yksinkertaisessa tilanteessa aineistossa on kaksi luokkaa, joihin havainnot on luokiteltava. Tällöin eräs soveltuva luokittelumenetelmä luokitteluongelman ratkaisemiseksi on esimerkiksi lineaarinen erotteluanalyysi (*linear discriminant analysis*, LDA), joka luokittelee havainnot niiden piirteiden (*feature*) perusteella lineaarisen päätöspinnan mukaisesti (James et al. 2021, 141–144). Piirre tarkoittaa luokittelumenetelmien yhteydessä sellaista aineiston muuttujaa, jonka mukaan luokittelu tehdään. Päätöspinta on puolestaan raja, joka erottaa kaksi luokkaa toisistaan. Kahdelle piirteelle päätöspinta on suora, kolmelle taso ja useammalle muuttujalle hypertaso. Monimutkaisemmassa tilanteessa luokat erotteleva päätöspinta ei ole lineaarinen, jolloin LDA ei ole sopiva luokitin. Tällöin esimerkiksi kvadraattinen erotteluanalyysi (*quadratic discriminant analysis*, QDA) on sopiva vaihtoehto, sillä sen päätöspinta ei ole lineaarinen ja se soveltuu siten monimutkaisempien luokitteluongelmien ratkaisuksi.

Molemmat edellä esitetyt menetelmät kuuluvat ohjattujen koneoppimismenetelmien (*supervised machine learning*) joukkoon, sillä niiden käyttö vaatii menetelmän ”opettamisen ohjatusti” erillistä aineistoa käyttäen. Nimi on kuvaava, sillä ohjatussa koneoppimisessa menetelmälle annetaan esimerkkejä havainnoista ja niiden oikeista luokista, ja esimerkkien perusteella koneoppimismenetelmän on tarkoitus luokitella oikein myös ennalta tuntematon havainto. LDA ja QDA ovat lisäksi esimerkkejä niin sanotuista pinnallisista koneoppimismenetelmistä (*shallow learning*), joiden rakenne sisältää yksinkertaisia funktioita ja/tai vain vähän yhdistettyjä funktioita. Pinnallisten menetelmien vastakohtana ovat niin sanotut syväoppimismenetelmät (*deep learning*), joissa esiintyy yhdistettyjä funktioita useissa kerroksissa (Goodfellow et al. 2016, 164–165).

Luokitteluongelman ratkaisussa on oleellista tietää, mitkä havainnon piirteet ovat tärkeitä luokan määräämisessä. Esimerkiksi kuvien luokittelussa kuvattavan kohteen asennolla ei pitäisi olla merkitystä luokittelun kannalta, mutta itse kohteesta havaittujen piirteiden sen sijaan kuuluisi olla tärkeitä. Luokittelussa havainnon piirteet pitää siten ensin valita (*feature extraction*) ja tämän jälkeen päättää, mitkä piirteet ovat tärkeitä. Piirteet voivat olla myös jo valmiiksi saatavilla (esimerkiksi datamatriisin muuttujina), mutta tällöinkin piirteiden tärkeys on vielä määritettävä. Pinnallisissa luokittelumenetelmissä piirteiden erottelu ja niiden tärkeyden päättäminen perustuvat johonkin ennalta päätettyyn sääntöön. Esimerkiksi tukivektorikoneen eri ydinfunktiot ovat tällaisia ennalta päätettyjä sääntöjä. Ennalta määrätyissä luokittelusäännöissä ei ole mitään vikaa, mutta ne eivät aina kykene suorittamaan monimutkaisempien aineistojen luokittelua. Lisäksi, säännön valitseminen perustellusti vaatii ennakkotietoa aineistosta sekä ratkaistavasta luokitteluongelmasta. (LeCun et al. 2015.)

Syväoppimismenetelmissä edellä kuvattu piirteiden erottelu ja luokittelusäännön laatiminen tapahtuvat automaattisesti, eikä sääntöjä tarvitse luoda tai tietää ennalta (LeCun et al. 2015). Syväoppimismenetelmän opetusprosessissa menetelmä tekee

piirteiden erottelun ja päätöksen piirteiden tärkeydestä opetuksessa käytetyn aineiston perusteella. Näin saatu luokitin kykenee ratkaisemaan myös monimutkaisempia luokitteluongelmia verrattuna pinnallisiin menetelmiin. Tosin on muistettava, että luokittimen valmius monimutkaisempien luokitteluongelmien ratkaisemiseksi riippuu täysin siitä, millaisella aineistolla se on opetettu.

Neuroverkko on yleisnimitys sellaisille syväoppimismenetelmille, joiden rakenteessa esiintyy toisiinsa kytkettyjä elementtejä ja johon syötteenä annettu informaatio kuvataan jollain tavalla menetelmän ulostulona (Picton 1994). Neuroverkkojen rakenne tarkoituksella muistuttaa elävien organismien biologista hermojärjestelmää, jossa hermojärjestelmän biologiset neuronit ovat yhdistetty toisiinsa niin sanotuissa synapseissa. Synapsien välisten yhteyksien voimakkuus muuttuu ulkoisen tekijän vaikutuksesta, mikä saa esimerkiksi ihmisen oppimaan uusia taitoja harjoittelun kautta. (Aggarwal 2018, 1.) Myös neuroverkon ajatellaan oppivan samalla tavalla, mutta ulkoinen tekijä on tällä kertaa opetuksessa käytetty aineisto. On huomattava, että termiä neuroverkko voidaan käyttää sekä biologisesta että syväoppimisessa käytetystä neuroverkosta, eli niin sanotusta keinotekoisesta neuroverkosta (*artificial neural network*). Tässä tutkielmassa jatkossa termillä neuroverkko viitataan keinotekoiseen neuroverkkoon.

Neuroverkon keskeinen idea ja tavoite voivat kuulostaa hyvin moderneilta, mutta neuroverkkojen ja niiden tutkimuksen historia ulottuu jo 1980-luvulle. Tällöin matemaatikot, tilastotieteilijät ja koneoppimisen asiantuntijat tutkivat ja kehittivät neuroverkkoja ja niihin liittyvää laskentaa eteenpäin. Kehitystä kuitenkin hidastivat saatavilla olevan datan ja tietokoneiden laskentakapasiteetin vähyys, minkä seurauksena neuroverkkoihin perustuvien menetelmien tehokas käyttö ei ollut vielä käytännössä mahdollista. Neuroverkkojen sijaan monissa käytännön sovelluksissa yksinkertaisemmat menetelmät, kuten satunnaismetsä ja tukivektorikone, syrjäyttivätkin neuroverkkoihin perustuvat menetelmät useiden vuosien ajaksi lähes kokonaan. Vasta 2010-luvulla neuroverkoilla on saatu aikaiseksi paljon onnistumisia esimerkiksi kuvien tunnistuksessa, tekstien ja puheen kääntämisessä toiselle kielelle, strategiapelien pelaamisessa (esimerkiksi Go-peli) sekä itseajavien autojen kehityksessä. (James et al. 2021, 403; Buckner 2019.) Tämä johtuu erityisesti saatavilla olevan datan määrän räjähdysmäisestä kasvusta, ja siten neuroverkkojen opettaminen monimutkaisia sovelluksia varten on mahdollista. (Aggarwal 2018, 4.)

Kuten mainittiin, neuroverkko ei ole yksi ainoa menetelmä, vaan neuroverkko-termin alle voidaan nykyään lukea monia neuroverkkorakenteen sisältäviä syväoppimismenetelmiä. Näitä menetelmiä ovat tavallisen täysin kytketyn neuroverkon lisäksi esimerkiksi konvoluutioneuroverkko (*convolutional neural network*), takaisinkytketty neuroverkko (*recurrent neural network*), sädeperusteinen neuroverkko (*radial basis function network*), rajoitettu Boltzmann-kone (*restricted Boltzmann machine*) ja niin sanottu pitkän lähimuistin neuroverkko (*long short-term memory*). Tässä tutkielmassa esitellään tavallinen täysin kytketty neuroverkko sekä konvoluutioneuroverkko ja molempia neuroverkkoja tarkastellaan luokitteluongelman näkökulmasta. Eri neuroverkkorakenteilla on erilaisia ominaisuuksia ja toiminnallisuuksia, jotka eivät rajoitu pelkästään luokitteluongelmien ratkaisemiseen. Oikein valitulla neuroverkkomenetelmällä voidaan siis ratkaista monentyyppisiä

ongelmia.

Eräs esimerkki luokitteluongelmasta on lajien tunnistaminen niiden perinnöllisyystekijöiden, erityisesti lajin DNA-sekvenssin, perusteella. Mikäli DNA-sekvenssi on tietynlainen, niin tällöin DNA-sekvenssiä kutsutaan DNA-viivakoodiksi. Vastaavasti DNA-viivakoodeihin perustuvaa lajien tunnistusta sanotaan DNA-viivakoodaukseksi. DNA-viivakoodauksessa varsinainen lajien tunnistaminen toteutetaan käyttämällä jotain tilastollista luokittelumenetelmää. Vastakohta DNA-viivakoodaukselle on ihmisen suorittama lajin tunnistus, jossa alan asiantuntija pyrkii määrittämään lajin manuaalisesti ilman koneellista luokittelijaa. Molempiin tapoihin liittyy luonnollisesti epävarmuutta: sekä kone että ihminen voivat erehtyä tunnistuksessa.

Hebert et al. (2005) mukaan DNA-viivakoodaukseen liittyy kaksi merkittävää etua verrattuna ihmisen tekemään luokitteluun. Ensimmäinen etu on DNA-viivakoodauksen hinta, joka on huomattavasti matalampi verrattuna vastaavaan ihmisen tekemään luokitteluun. Kirjoittajien mukaan asiantuntijan tekemä lajin tunnistaminen maksaa keskimäärin 50–100 dollaria (USD) yhtä lajia kohden, kun taas yhden DNA-sekvenssin eristäminen lajista ja sekvenssin esikäsittely luokittelua varten maksaa noin 5 dollaria (USD). Toinen merkittävä etu koneellisessa luokittelussa on sen nopeus ja helppous: Tavoitteena on kehittää täysin itsenäisiä järjestelmiä, joilla lajeja voidaan tunnistaa DNA-viivakoodeihin perustuen edullisesti ja nopeasti.

DNA-viivakoodaus on saanut osakseen myös kritiikkiä. Esimerkiksi Zamani et al. (2022) esittävät, kuinka DNA-sekvenssin eristäminen lajista vaatii kalliita laitteita, joihin kaikilla tutkijoilla ei ole pääsyä. Kirjoittajien mukaan seurauksena kokonaan uusien lajien havaitseminen jää näin yhä harvemman asiantuntijan harteille, jolloin kokonaan uusien lajien löytäminen kaiken kaikkiaan hidastuisi. Kirjoittajat kuitenkin myös toteavat, että DNA-sekvensseihin perustuvat lajien tunnistusmenetelmät voivat olla hyvinkin hyödyllisiä ensimmäisenä keinona tuntematonta lajihavaintoa tutkittaessa.

Kritiikistä huolimatta DNA-viivakoodeihin ja -sekvensseihin perustuvien luokittimien tutkimuksia on julkaistu säännöllisesti. Esimerkiksi Somervuo et al. (2016) esittivät multinomiaaliseen logistiseen regressioon perustuvan DNA-sekvenssien luokittimen, jota Abarenkov et al. (2018) sovelsivat sienilajien luokittelussa. Zhou et al. (2011) puolestaan esittivät DNA-sekvenssien luokittelumenetelmän, jossa käytetään k-keskiarvon ryhmittelyä piirteiden valinnassa ja tämän jälkeen tukivektorikonetta varsinaisessa luokittelussa. Etenkin viimeisimpien vuosien aikana, monissa vastaavissa tutkimuksissa on päädytty käyttämään jotain syväoppimismenetelmää. Esimerkiksi Le et al. (2021) esittivät neuroverkkoluokittimen, jossa DNA-sekvenssejä käsitellään luonnollisen kielen käsittelyn (*natural language processing*, NLP) menetelmin. Yang et al. (2022) puolestaan käyttivät konvoluutioneuroverkkoluokitinta ja vertailevat sitä muihin menetelmiin. Osoittautuu, että tämänkin tutkielman aiheena oleva konvoluutioneuroverkko soveltuu hyvin sekvenssimuotoisten aineistojen (kuten DNA-sekvenssien) luokittimeksi.

Tämän pro gradu -tutkielman tavoitteena on esitellä täysin kytketty neuroverkko sekä konvoluutioneuroverkko, tarkastella neuroverkkojen opettamisessa käytettyjä algoritmeja ja esittää konvoluutioneuroverkon sovellus, jossa hyönteislajeja luokitellaan nii-

den DNA-viivakoodien perusteella. Tutkielman aluksi luvussa 2 esitetään lyhyesti geenit, DNA ja DNA:n rakenne sekä tutustutaan DNA-viivakoodaukseen ja käytettyyn DNA-sekvenssiaineistoon tarkemmin. Tämän jälkeen luvuissa 3 ja 4 esitetään täysin kytketyt eteenpäin syöttävät neuroverkot sekä niiden opettamisessa käytettyjä algoritmeja ja muita matemaattisia toimenpiteitä. Luvussa 5 tutustutaan konvoluutioneuroverkkoihin ja niissä tapahtuviin operaatioihin tavallisten värikuvien sekä myös DNA-sekvenssien luokittelun näkökulmasta. Luvussa 6 esitetään luokittelussa käytetyt konvoluutioneuroverkkoluokittimet ja luokittelun tulokset. Lopuksi luvussa 7 annetaan tutkielman yhteenveto ja esitetään joitain mahdollisia jatkotutkimusten aiheita ja niihin soveltuvia menetelmiä.

2 Genomiikka, geenit ja DNA

Tässä luvussa esitellään ensiksi lyhyesti tämän tutkielman kannalta oleellisin genomiikan termistö. Toiseksi kuvataan tutkielman analyyseja varten saatu DNA-sekvenssiaineisto sekä DNA-viivakoodaus.

2.1 Genomiikka

Genomiikka (*genomics*) on biologian tieteenala, joka tutkii elävien organismien geenejä. Geenit puolestaan ovat organismin ”perinnöllisyyden yksikköjä”, jotka joko yksittäin tai yhdessä tiettyjen muiden geenien kanssa muodostavat yksiselitteiset toimintaohjeet jollekin organismin ominaisuudelle tai toiminnolle. Kaikki yhden organismin geenit muodostavat organismin genomin (*genome*) eli perimän. (Amaratunga et al. 2004, 17–18.) Geeneistä ei tiedetty juurikaan mitään muuta konkreettista ennen 1900-luvun alkua (Amaratunga et al. 2004, 9; Dahm 2005). 1880-luvun puolivälissä oli kuitenkin osoitettu jo useamman tutkijan toimesta, että perinnöllisyyden yksiköt sijaitsevat organismin solun ytimessä. Termiä ”geeni” alettiin käyttämään vuonna 1909. (Dahm 2005.)

Aiemmin epäiltiin, että geenit ovat koostumukseltaan proteiineja. Lääkäri Oswald T. Avery ja geneetikot Colin MacLeod ja Maclyn McCarty (Avery et al. 1944) kuitenkin osoittivat, että geenit ovat proteiinien sijasta deoksiribonukleiinihappoa (*deoxyribonucleic acid*, DNA). Deoksiribonukleiinihapon eli DNA:n rakenne oli tuntematon vielä tässä vaiheessa. Tilanne muuttui jo alle vuosikymmenen kuluttua vuonna 1953, kun molekyylibiologit James Watson ja Francis Crick (Watson et al. 1953) esittivät DNA:n kaksoiskierrerakenteen (*double helix*). (Dahm 2005.)

2.2 DNA:n rakenne

Tässä alaluvussa esitetään tarkemmin DNA:n rakenne. Esitys perustuu pääosin teoksen Amaratunga et al. (2004) lukuun 2.

DNA-molekyyli koostuu kahdesta pitkästä nauhasta, jotka ovat kietoutuneet toistensa ympärille muodostaen spiraalimaisen kaksoiskierrerakenteen. Rakenne muistuttaa tikapuita, joita on väännetty sen molemmista päistä eri suuntiin. Kumpikin DNA:n nauha

koostuu nukleotideistä (*nucleotide*), jotka ovat sijoittuneet nauhoille nukleotidipareiksi. Yksi nukleotidipari siis koostuu kahdesta nukleotidistä, jotka sijaitsevat samoissa kohdissa DNA:n kummallakin nauhalla.

Nukleotidi puolestaan koostuu sokeriosasta, fosfaattiosasta ja typpipitoisesta emäksestä. Sokeriosa sisältää tietyn sokerimolekyylin (DNA:n tapauksessa deoksiriboosimolekyylin (*deoxyribose molecule*)), joka on kiinnittynyt fosfaattiosaan. Typpipitoinen emäs on yksi neljästä emäsvaihtoehdosta: adeniini (*adenine*, lyhenne: `A`), tymiini (*thymine*, `T`), guaniini (*guanine*, `G`) tai sytosiini (*cytosine*, `C`). (Tefferi 2006.) Nukleotidiparien sijoittuminen DNA:n nauhoilla noudattaa aina niin sanottuja emäsparisääntöjä (*base pair rule*): Adeniinin pari on aina tymiini, ja guaniinin pari on aina sytosiini. Emäsparien lukumäärä on DNA:n pituus, jonka yksikkönä käytetään emäspariin (*base pair*) viittaavaa lyhennettä *bp*. Emäsparisääntöjen vuoksi emäsparien emäkset ovat aina pääteltävissä, kun parista tiedetään vähintään toinen. Kuitenkin emäsparien keskinäisen järjestyksen pitää olla tiedossa, sillä se on sekä DNA:n geneettisten ominaisuuksien että DNA:n tilastollisen analysoinnin kannalta tärkeää. DNA-sekvenssiksi (*DNA sequence*) sanotaan DNA:n emästen järjestystä yhdellä DNA:n nauhalla. DNA-sekvenssi esitetään kirjoittamalla emästen järjestys yhteen merkkijonoon käyttäen emästen lyhenteitä (`A`, `T`, `G` ja `C`).

Emästen lyhenteet noudattavat nimeämissuositusta, jonka antoi *International Union of Pure and Applied Chemistry* (IUPAC) julkaisussa Cornish-Bowden (1985). Sama suositus määrää myös muita symboleja eri emäsvaihtoehdoille sen perusteella, kuinka paljon tietystä emäksestä tiedetään. Esimerkiksi symboli `N` voidaan antaa sellaiselle emäkselle, joka voi olla mikä tahansa neljästä emäsvaihtoehdosta. Tällaisessa tilanteessa emästä joko ei ole pystytty lainkaan määräämään tarkasti tai sitten DNA:n toiminnan kannalta ei ole tärkeää, mikä emäs on kyseessä. Suosituksen mukaan symboli `N` voidaan myös korvata viivalla (`-`) erityisesti silloin, jos tekstin joukossa halutaan korostaa tiedossa olevia emäksiä (esim. `NNNNNCNNGNTNN` ⇒ `-----C--G-T--`).

2.3 PROTAX-aineisto ja DNA-viivakoodaus

Tässä tutkielmassa käytetty DNA-sekvenssiaineisto, johon jatkossa viitataan nimellä PROTAX-aineisto, on saatu Jyväskylän yliopiston bio- ja ympäristötieteiden laitoksen akatemiaprofessori Otso Ovaskaisen kautta. Kyseessä on sama aineisto, jonka Roslin et al. (2022) keräsivät ja jota he käyttivät FinPROTAX-luokittimensa opettamisessa. FinPROTAX-luokitin puolestaan perustuu Somervuo et al. (2016) kehittämään PROTAX-luokittimeen, joka kehitettiin organismin taksonomian eri tasojen hierarkkiseen luokitteluun. Organismin taksonomian tasoja ovat esimerkiksi (hierarkkisessa järjestyksessä laajemmasta suppeampaan) kunta, pääjakso, luokka, lahko, heimo, suku ja laji. Esimerkiksi kissa (*Felis catus*) kuuluu eläinkunnan kuntaan, selkäjänteisten pääjaksoon, nisäkkäiden luokkaan, petoeläinten lahkoon, kissaeläinten heimoon, ja kissojen sukuun (Lajitietokeskus 2022).

PROTAX-aineistossa on yhteensä 38 243 DNA-sekvenssiä, jotka on eristetty niveljal-

kaisten pääjaksoon kuuluvilta hyönteislajeilta Suomessa. Sekvenssien pituuksien vaihteluväli on $[503; 1503]$ (*bp*). Lajien keruussa on hyödynnetty sekä lajien museokokoelmia että myös alan harrastajien yksityisiä kokoelmia. Hyönteisistä otettiin kudosnäyte, joka oli tavallisimmin hyönteisen jalka tai jalan osa riippuen hyönteisen koosta. Tämän jälkeen näytteet pakattiin ja lähetettiin Guelphin yliopiston biodiversiteettigenomiikan keskukseen (*Centre for Biodiversity Genomics*) Ontarioon Kanadaan, jossa lajien DNA eristettiin alan standardien mukaisesti. (Roslin et al. 2022.)

PROTAX-aineiston DNA-sekvenssit eivät sisällä tietyn organismin koko perimää tai edes tiettyä kokonaista geeniä. Sen sijaan aineiston sekvenssit on tarkasti valittu sellaisista organismin geeneistä ja sellaisista DNA:n kohdista, jotka ovat lajin tunnistamisen kannalta tärkeitä. Valinta perustuu Hebert et al. (2003) esittämään ideaan: Organismin perimässä on olemassa tietty geeni, jonka tietyn kohdan perusteella lajien luotettava tunnistaminen on mahdollista. Tämä kyseinen geeni on eläimillä sytokromi c-oxidaasigeeni. Edellä kuvatun kaltaista DNA-sekvenssiä kutsutaan DNA-viivakoodiksi. Vastaavasti DNA-viivakoodauksella tarkoitetaan menetelmää, joka tunnistaa lajeja DNA-viivakoodien perusteella.

Jotta DNA-viivakoodausta voidaan käyttää luokittelussa, tarvitaan tietoa jo tunnistettujen lajien DNA-viivakoodeista, joihin luokittelemattoman lajin DNA-viivakoodia verrataan. Roslin et al. (2022) mukaan Suomessa tavoitteena on luoda tietokanta, joka sisältää DNA-sekvenssitiedot Suomen kaikista tunnetuista monisoluisista organismeista. Näitä arvioidaan olevan Suomessa noin 48 000. Hanke on käynnissä ja se tunnetaan nimellä FinBOL (*The Finnish Barcode of Life Initiative*, `https://www.finbol.org/`). Vastaavia hankkeita on myös muissa maissa Suomen lisäksi (esim. Saksassa, Kanadassa ja Norjassa). Kansallisten hankkeiden lisäksi kansainvälinen BOLD-projekti (*The Barcode of Life Data System*, Sujeevan et al. (2007)) ylläpitää muun muassa avointa tietokantaa, joka sisältää DNA-viivakoodihavaintoja eri lajeilta ympäri maailmaa.

PROTAX-aineisto muiden DNA-sekvenssiaineistojen tapaan tallennetaan fasta-formaatissa (`.fasta-` tai `.fa`-tiedostopääte). Fasta-tiedosto on tekstitiedosto, jossa on DNA-sekvenssit ja niiden tunnisteet. Tunniste on kirjaimista, numeroista ja muista merkeistä koostuva merkkijono, joka yksilöi jokaisen DNA-sekvenssin. Tunniste alkaa yleensä erikoismerkillä (kuten '>'), jotta se voidaan helposti erottaa koneellisesti DNA-sekvenssistä. Tunnisteen alapuolella seuraavalla rivillä on tätä tunnistetta vastaava DNA-sekvenssi. Alla on esimerkki PROTAX-aineiston yhdestä DNA-sekvenssistä fasta-formaatissa (DNA-sekvenssi on lyhennetty):

```
>TRIFI916-13
AACAATTTACTTTATTTTTGGAATCTGAGCTGGTATGGTAGGAACTTCACTAAGAATGATTATTCGAAC...
```

Yllä oleva DNA-sekvenssi on peräisin rytihiidekkäältä (*Agrypnia pagetana*[1]), joka on vesiperhosten lahkoon kuuluva hyönteinen. Koko DNA-sekvenssin pituus on 658 *bp*.

Alkuperäinen, tätä tutkielmaa varten saatu PROTAX-aineisto sisälsi vain DNA-viivakoodien tunnisteet, mutta ei tunnisteita vastaavia lajeja. Lajit saadaan kuitenkin

[1] `https://laji.fi/taxon/MX.231845`

haettua vapaasti BOLD-tietokannasta[2], sillä FinBOL-projektin DNA-viivakoodit muine oheistietoineen ladataan myös BOLD-tietokantaan (Roslin et al. 2022). Lopullisessa aineistossa DNA-viivakoodeja on yhteensä 11 528 eri lajilta. Viivakoodien lukumäärät lajeittain ovat jakautuneet kuitenkin epätasaisesti. Nimittäin, sellaisia lajeja, joista on vain yksi sekvenssi, on 3265 kappaletta (28.3 %). Vastaavasti, sellaisia lajeja, joista sekvenssejä on saatavilla 10 tai enemmän on vain 416 kappaletta (3.6 %). Siispä, valtaosaa aineiston lajeista voi olla haastavaa käyttää minkäänlaisessa koneoppimismallissa johtuen näiden lajien viivakoodien pienestä lukumäärästä. Kuitenkin on muistettava, että DNA-viivakoodit on juurikin valittu lajin tunnistettavuus silmällä pitäen, joten luokittelu voi onnistua myös pienemmällä määrällä sekvenssejä.

3 Neuroverkot

Tässä luvussa esitellään täysin kytketty eteenpäin syöttävä neuroverkko ja siihen liittyviä käsitteitä ja teoriaa. Luvut 3.1 ja 3.3 pohjautuvat teoksen Goodfellow et al. (2016) lukuihin 6.0 ja 6.3 ja luku 3.2 puolestaan teoksen Aggarwal (2018) lukuun 1.

3.1 Täysin kytketyt eteenpäin syöttävät neuroverkot

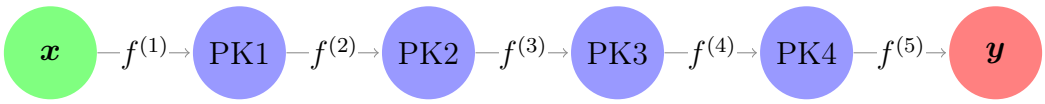
Täysin kytketty eteenpäin syöttävä neuroverkko (*fully connected feed-forward neural network*) on neuroverkkorakenteen omaava syväoppimismenetelmä, joka soveltuu muun muassa luokitteluongelman ratkaisuksi. Täysin kytkettyä eteenpäin syöttävää neuroverkkoa monesti pidetään ”tavallisena” tai ”perinteisenä” neuroverkkona, joten viitataan tähän menetelmään jatkossa yksinkertaisuuden vuoksi neuroverkko-termillä.

Luokittelutilanteessa neuroverkon tarkoituksena on luokitella yksittäinen havainto $(Y, \boldsymbol{x})$, jossa Y on havainnon luokka ja vektori $\boldsymbol{x}$ sisältää havainnon piirteet. Olkoon luokkia M kappaletta ja merkitään niitä kokonaisluvuilla $1, \ldots, M$. Luokka voidaan esittää nyt myös M-pituisena luokkavektorina $\boldsymbol{y} = (y_1, y_2, \ldots, y_M)^T$, jossa on ykkönen sen luokan kohdalla, johon havainto kuuluu. Muut luokkavektorin alkiot ovat nollia. Neuroverkon varsinainen tehtävä on approksimoida funktio f kuvauksessa $\boldsymbol{y} = f(\boldsymbol{x})$. Toisin sanoen funktio f kuvaa piirremuuttujat $\boldsymbol{x}$ jollain tavalla luokkavektoriksi $\boldsymbol{y}$.

Neuroverkossa funktio f on yhdistetty funktio, jonka seurauksena neuroverkoissa esiintyy nimenmukainen verkkorakenne. Yhdistetty funktio f puolestaan koostuu funktioista $f^{(i)}$, $i = 1, 2, \ldots, k$. Funktiota $f^{(i)}$ kutsutaan neuroverkon i:nneksi kerrokseksi (*layer*). Myös piirremuuttujia $\boldsymbol{x}$ ja luokkavektoria $\boldsymbol{y}$ ajatellaan kerroksina, joita kutsutaan neuroverkon syötekerrokseksi (*input layer*) ja ulostulokerrokseksi (*output layer*), vastaavasti. Neuroverkon käyttäjä havaitsee neuroverkosta vain syöte- ja ulostulokerroksen, joten niiden välissä olevia kerroksia sanotaan neuroverkon piilokerroksiksi (*hidden layer*). Piilokerroksia on $k - 1$ kappaletta. Funktioiden $f^{(i)}$ lukumäärää k kutsutaan neuroverkon syvyydeksi. Esimerkiksi neuroverkon, jonka syvyys on viisi, tavoitteena on approksimoida viidestä yhdistetystä funktiosta koostuva kuvaus $\boldsymbol{y} = f^{(5)}\left(f^{(4)}\left(f^{(3)}\left(f^{(2)}\left(f^{(1)}\left(\boldsymbol{x}\right)\right)\right)\right)\right)$.

[2] `https://www.boldsystems.org/index.php/Public_BINSearch?query=Finland`

Yhdistetyissä funktioissa näkyy eteenpäin syöttävien neuroverkkojen ominainen rakenne, jossa yhdistetty funktio saa syötteeksi aiemman funktion koko ulostulon. Eteenpäin syöttävissä neuroverkoissa tämä näkyy siten, että seuraavan kerroksen syöte on aina edellisen kerroksen ulostulo, eikä ulostuloa miltään kerrokselta aseteta takaisin saman neuroverkon minkään kerroksen syötteeksi. Eteenpäin syöttävä neuroverkko voidaan tällä periaatteella esittää suunnattuna asyklisenä graafina (*directed acyclic graph*, DAG), jossa eri kerrosten väliset yhteydet kuvataan nuolilla. Kuviossa 1 on esitetty aiempi viisikerroksinen neuroverkko graafina.



Kuvio 1: *Neuroverkon ja sen rakenteen havainnollistus suunnattuna asyklisenä graafina. Kuvion neuroverkossa on syötekerros $\boldsymbol{x}$, neljä piilokerrosta (PK) ja ulostulokerros $\boldsymbol{y}$. Tämän neuroverkon syvyys (k) on viisi.*

Kuvion 1 graafissa jokainen solmu kuvaa yhtä neuroverkossa laskettua reaalilukua, joita kutsutaan neuroneiksi. Toisin kuin kuvion 1 graafissa, käytännön sovelluksissa neuroverkon kaikki kerrokset koostuvat lähestulkoon aina useammasta kuin yhdestä neuronista. Neuroverkon tietyn kerroksen neuronien lukumäärää kutsutaan kyseisen kerroksen leveydeksi.

Syöte $\boldsymbol{x}$ koostuu piirteiden arvoista. Jokainen piirre voidaan antaa neuroverkolle omana syötteenään siten, että yksi neuroni varataan yhtä piirremuuttujaa kohti. Syötekerroksen leveys on tällöin vähintään sama kuin alkuperäinen muuttujien lukumäärä. Muuttujille kuitenkin pitää lähes aina tehdä esikäsittelyä ennen kuin ne voidaan syöttää neuroverkkoon. Erityisesti luokiteltujen muuttujien eri luokat pitää muuttaa numeeriseen muotoon indikaattorimuuttujien avulla, joten syötekerroksen leveys voi olla myös suurempi kuin alkuperäisten muuttujien lukumäärä.

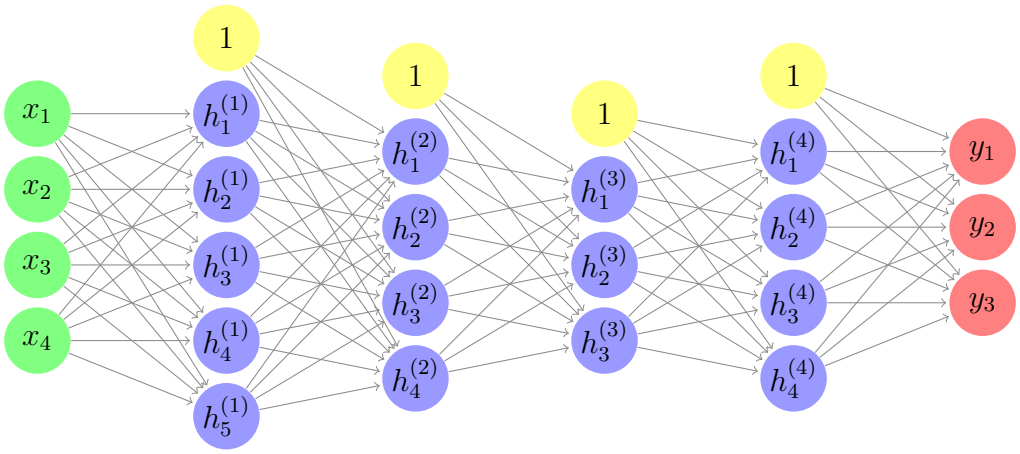
Vastaavaan tapaan piilokerrokset ja ulostulokerros koostuvat yhdestä tai useammasta neuronista. Neuronit i:nnellä kerroksella muodostavat vektorin $\boldsymbol{h}^{(i)} = (h_1^{(i)}, h_2^{(i)}, \ldots, h_{p_i}^{(i)})^T$, jossa vektorin pituus p_i on samalla kerroksen i leveys. Alaindeksi $1, 2, \ldots, p_i$ ilmaisee neuronin järjestysluvun kerroksella ja yläindeksi (i) sen, millä kerroksella tämä neuroni sijaitsee. Neuroverkon graafissa saman piilokerroksen neuronit sijoitetaan allekkain noudattamalla aiemmin kuvattua eteenpäin syöttävän neuroverkon periaatetta, jossa aiemman piilokerroksen ulostulo syötetään seuraavalle piilokerrokselle syötteenä. Toisin sanoen jokainen neuroni saa syötteenä vektorin, joka koostuu kaikista edellisen piilokerroksen neuroneista. Tällä tavalla muodostetun neuroverkon yhtä kerrosta kutsutaan täysin kytketyksi kerrokseksi (*fully connected layer*) ja koko neuroverkkoa voi siten sanoa täysin kytketyksi eteenpäin syöttäväksi neuroverkoksi. Esimerkki tällaisesta neuroverkkograafista ja neuronien indekseistä on esitetty kuviossa 2.

Tarkastellaan seuraavaksi vielä tarkemmin neuroverkon neuroneita ja sitä, miten neuronien arvot lasketaan neuroverkossa.

3.2 Painot ja vakioneuronit

Neuroverkossa termillä paino (*weight*) tarkoitetaan kahden peräkkäisillä piilokerroksilla sijaitsevien neuronien välisen ”yhteyden voimakkuutta”. Esimerkiksi kuviossa 2 jokainen nuoli kuvaa kahden neuronin välistä yhteyttä, ja näiden yhteyksien voimakkuudet ovat neuroverkon painoja. Painot ovat reaalilukuja samoin kuin neuronitkin ja niitä muuttamalla voi muuttaa koko neuroverkon toimintaa. Painoja kutsutaan tilastotieteen kirjallisuudessa myös neuroverkon parametreiksi, mutta termi paino on vakiintunut erityisesti syväoppimismenetelmien yhteydessä (LeCun et al. 2015).

Painot, jotka ovat kerrosten $(i-1)$ ja (i) välissä, kootaan $p_{i-1} \times p_i$ -painomatriisiin $\boldsymbol{W}_i$. Painomatriisin alaindeksi on siis sen kerroksen indeksi, johon painot vaikuttavat. Esimerkiksi kuviossa 2 ensimmäinen painomatriisi on $\boldsymbol{W}_1$ ja sen painot ovat syötekerroksen ja ensimmäisen piilokerroksen välissä, ja vastaavasti viimeisen painomatriisin $\boldsymbol{W}_5$ painot ovat viimeisen piilokerroksen ja ulostulokerroksen välissä. Painomatriisin koko riippuu niiden piilokerrosten leveyksistä, joiden välissä painomatriisin painot ovat. Piilokerrosten $(i-1)$ ja (i) välisten painojen painomatriisi on siten $p_{i-1} \times p_i$ -matriisi. Matriisin yksittäinen alkio on paino $w_{rivi,sarake}$, joka lähtee kerroksen $(i-1)$ *rivi*-indeksin osoittamasta neuronista ja vaikuttaa kerroksen (i) *sarake*-indeksin osoittamaan neuroniin. Painojen kokoamista matriisiin on havainnollistettu kuviossa 3.

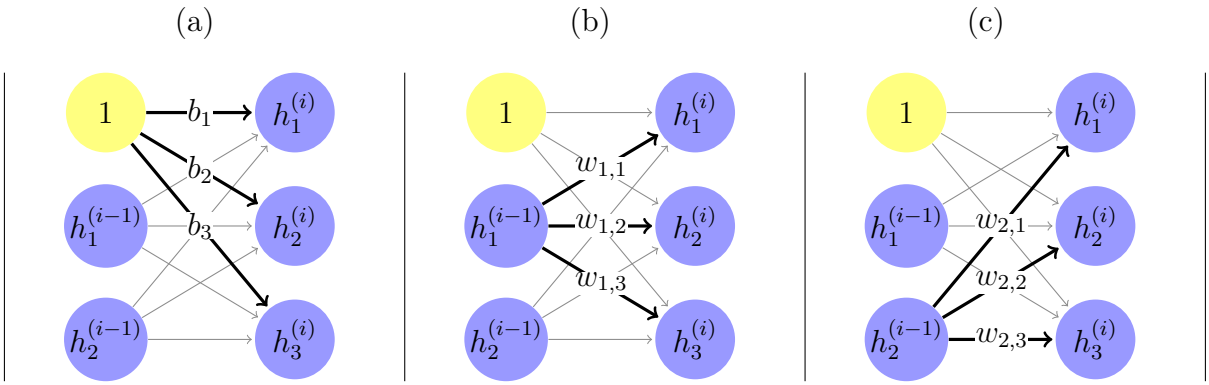


Kuvio 2: *Neljä täysin kytkettyä piilokerrosta sisältävä neuroverkko, jossa piilokerrosten leveydet ovat viisi, neljä, kolme ja neljä. Lisäksi kullakin piilokerroksella on vakioneuroni, jota merkitään luvulla yksi. Neuroverkko ottaa syötteenä neljä piirremuuttujaa sisältävän vektorin* $\boldsymbol{x} = (x_1, x_2, x_3, x_4)^T$*. Ulostulo on kolmeluokkainen luokkavektori* $\boldsymbol{y} = (y_1, y_2, y_3)^T$*.*

Neuroverkkojen piilokerroksissa voi olla neuronien lisäksi myös niin sanottuja vakioneuroneita (*bias neuron*). Vakioneuronit ovat neuroneita, jotka saavat neuroverkossa aina arvon yksi. Koska vakioneuroni pysyy muuttumattomana neuroverkossa, niin ainoastaan vakioneuronista lähteviä painoja muuttamalla voidaan vaikuttaa seuraavan kerroksen neuroneihin. Kullakin neuroverkon piilokerroksella voi olla maksimissaan yksi vakioneuroni. Vakioneuronien toiminta muistuttaa regressiomallin vakiotermiä, jota vastaavat alkiot asetelmamatriisissa ovat ykkösiä.

Vakioneuronin painot käyttäytyvät samalla tavalla kuin muidenkin neuronien painot:

vakioneuronista lähtee yksi paino jokaiseen seuraavan kerroksen neuroniin. Vakioneuronin painoja ei kuitenkaan esiinny painomatriiseissa eikä vakioneuronia lasketa mukaan kerroksen i leveyteen p_i. Vakioneuroneita sen sijaan käsitellään erillään muista neuroneista ja niistä lähtevät painot kootaan omiin vektoreihin $\boldsymbol{b}_i = (b_1, b_2, \ldots, b_{p_i})^T$. Alaindeksi i on jälleen sen kerroksen indeksi, johon vakioneuronin painot vaikuttavat. Neuroverkkojen graafeissa vakioneuronit esitetään samalla tavalla kuin muutkin neuronit. Vakioneuronit kuitenkin poikkeavat tavallisista neuroneista siten, että vakioneuronit eivät ota syötteenä yhtään painoa edelliseltä kerrokselta. Neuroverkon graafissa vakioneuroniin ei siten saavu yhtään nuolta, mutta siitä lähtee nuolet muiden neuronien tapaan. Vakioneuroni, sen painot ja niiden sijoittuminen graafissa on esitetty kuvioissa 2 ja 3.



$$\boldsymbol{W}_i = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{bmatrix}$$

Kuvio 3: *Esimerkki kahden piilokerroksen välisistä painoista ja niiden sijoittumisesta painomatriisiin. Piilokerroksessa $(i-1)$ on kaksi neuronia sekä yksi vakioneuroni. Seuraavassa piilokerroksessa (i) on kolme neuronia ja ei vakioneuronia ollenkaan. Tilanteessa (a) on korostettu vakioneuronista lähtevät painot jokaiseen seuraavan kerroksen neuroniin. Nämä painot eivät ole painomatriisissa $\boldsymbol{W}_i$. Tilanteissa (b) ja (c) on vastaavasti korostettu kummastakin kerroksen $(i-1)$ neuronista lähtevät painot jokaiseen seuraavan kerroksen (i) neuroniin. Nämä painot tulevat järjestyksessä painomatriisin ensimmäiselle ja toiselle riville.*

Nyt painojen ja neuroverkon eri kerrosten neuronien avulla voidaan laskea rekursiivisesti seuraavan kerroksen neuronit. Ensimmäisen piilokerroksen $\boldsymbol{h}^{(1)}$ neuronit saadaan syötekerroksen ja ensimmäisen painomatriisin avulla kaavoilla

$$\begin{aligned} \boldsymbol{a}^{(1)} &= \boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{b}_1 \\ \boldsymbol{h}^{(1)} &= \sigma\left(\boldsymbol{a}^{(1)}\right), \end{aligned} \tag{1}$$

jossa funktio $\sigma(\cdot)$ on ennalta määrätty aktivaatiofunktio. Aktivaatiofunktioita käsitellään tarkemmin luvussa 3.3. Vastaavasti seuraavien piilokerrosten neuronit saadaan laskettua aina edellisen piilokerroksen neuronien ja edellisten painojen avulla

$$\begin{aligned} \boldsymbol{a}^{(i)} &= \boldsymbol{W}_i^T \boldsymbol{h}^{(i-1)} + \boldsymbol{b}_i \\ \boldsymbol{h}^{(i)} &= \sigma\left(\boldsymbol{a}^{(i)}\right), \end{aligned} \tag{2}$$

kun $i = 2, 3, \ldots, k-1$. Viimeisenä, tilanteessa $i = k$, ulostulokerroksen neuronit lasketaan kaavoilla

$$\begin{aligned} \boldsymbol{a}^{(k)} &= \boldsymbol{W}_k^T \boldsymbol{h}^{(k-1)} + \boldsymbol{b}_k \\ \boldsymbol{h}^{(k)} &= \sigma\left(\boldsymbol{a}^{(k)}\right). \end{aligned} \tag{3}$$

Esimerkiksi kuvion 3 mukaisessa neuroverkkorakenteessa voidaan laskea kerroksen (i) neuronien arvot kaavojen (2) mukaisesti. Tällöin painomatriisi on sama matriisi kuin kuviossa 3, ja vakioneuronin painot sisältävä vektori on $\boldsymbol{b}_i = (b_1, b_2, b_3)^T$.

3.3 Aktivaatiofunktiot

Aktivaatiofunktion tarkoituksena on tehdä jokin ennalta päätetty muunnos neuroneille ennen kuin ne syötetään neuroverkossa eteenpäin seuraavalle kerrokselle. Aktivaatiofunktion valintaan ei ole olemassa yleispäteviä sääntöjä, vaan valinta yleensä tehdään eri aktivaatiofunktioita testaamalla ja mallin hyvyyttä tarkastelemalla.

Seuraavaksi esitellään yleisimpiä vaihtoehtoja kaavoissa (1)–(3) esiintyvälle aktivaatiofunktiolle $\sigma(\cdot)$. Aktivaatiofunktiot jaetaan tässä luvussa kahteen ryhmään niiden käyttökohteen perusteella: Ensin luvussa 3.3.1 esitellään yleisimpia vaihtoehtoja piilokerroksilla käytetyille aktivaatiofunktioille, minkä jälkeen luvussa 3.3.2 esitellään ulostulokerrokselle hyvin soveltuvia aktivaatiofunktioita.

Kaavoissa (1)–(3) aktivaatiofunktio ottaa syötteenään vektorin. Tästä huolimatta, kaikki tässä luvussa esitellyistä aktivaatiofunktioista yhtä funktiota lukuunottamatta ottavat syötteenään skalaarin. Tällöin aktivaatiofunktio operoi jokaisen vektorin alkion yksitellen alla olevan esimerkin mukaisesti:

$$\sigma\left(\boldsymbol{a}^{(i)}\right) = \sigma\left(\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{p_i} \end{bmatrix}\right) = \begin{bmatrix} \sigma(a_1) \\ \sigma(a_2) \\ \vdots \\ \sigma(a_{p_i}) \end{bmatrix}. \tag{4}$$

3.3.1 Piilokerrosten aktivaatiofunktiot

Yksinkertaisimmillaan aktivaatiofunktio on identtinen funktio $\sigma(x) = x$ tai jokin muu lineaarinen funktio. Tällöin neuroneille tehdään lineaarinen muunnos ennen seuraavalle kerrokselle siirtymistä. Yksinkertaisuudestaan huolimatta lineaariset aktivaatiofunktiot eivät kuitenkaan ole hyviä vaihtoehtoja neuroverkon suorituskyvyn kannalta. Aktivaatiofunktioiden on oltava nimittäin epälineaarisia, jotta neuroverkko oppisi myös aineiston epälineaarisia piirteitä. Epälineaarisuuksien tehokas oppiminen saavutetaan käyttämällä epälineaarisia aktivaatiofunktioita piilokerroksilla. Kun neuroverkosta tehdään lisäksi tarpeeksi syvä kerroksia lisäämällä, niin tällöin neuroverkon kyky havaita aineiston epälineaarisuuksia paranee ja neuroverkon suorituskyky siten myös kasvaa. (Gu et al. 2018.) Aineiston epälineaarisuuksien oppiminen onkin yksi syvien neuroverkkojen tärkeimmistä ja hyödyllisimmistä ominaisuuksista, mikä erottaa neuroverkot muista pinnallisista koneoppimismenetelmistä.

Neuroverkkomenetelmien yleistyessä aktivaatiofunktiona käytettiin useasti hyperbolista tangenttia

$$\sigma(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{tanh}$$

tai sigmoidifunktiota

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{sigmoidi}$$

jotka molemmat ovat epälineaarisia aktivaatiofunktioita. Näiden funktioiden käyttöä neuroverkon piilokerrosten aktivaatiofunktioina ei kuitenkaan enää suositella, sillä hyperbolinen tangentti ja sigmoidifunktio kärsivät katoavan gradientin ongelmasta, mikä voi usein hidastaa neuroverkon opettamista tai pahimmillaan estää sen kokonaan. Katoavan gradientin ongelmaan palataan tarkemmin luvussa 4.5.2.

ReLU (*Rectified Linear Unit*) on nykyajan syväoppimismalleissa yleisimmin käytetty aktivaatiofunktio. Sillä on saavutettu erinomaisia tuloksia ja nykyään ReLU:a pidetäänkin yhtenä parhaimmista syväoppimismallien aktivaatiofunktioista (Ramachandran et al. 2017). ReLU on epälineaarinen paloittain määritelty kuvaus

$$\sigma(x) = \begin{cases} x, & \text{kun } x \geq 0 \\ 0, & \text{kun } x < 0 \end{cases}, \tag{ReLU}$$

joka pitää voimakkaat (positiiviset) neuronien väliset yhteydet voimakkaina ja mitätöi muut yhteydet kokonaan. Lisäksi ReLU:n derivaatta on vakio (yksi) neuronin ollessa aktiivinen ja muulloin nolla, mikä tekee neuroverkon opettamisesta tehokasta. Aktivaatiofunktion derivaatta liittyy oleellisesti neuroverkon opetusprosessiin, mihin palataan myöhemmin luvuissa 4.4.2 ja 4.5.2.

ReLU:a on tutkittu paljon ja sen pohjalta on kehitetty myös muita aktivaatiofunktioita. Yksi esimerkki ReLU:n varianteista on Maas et al. (2013) esittelemä niin sanottu ”vuotava” ReLU (*leaky ReLU*, LReLU)

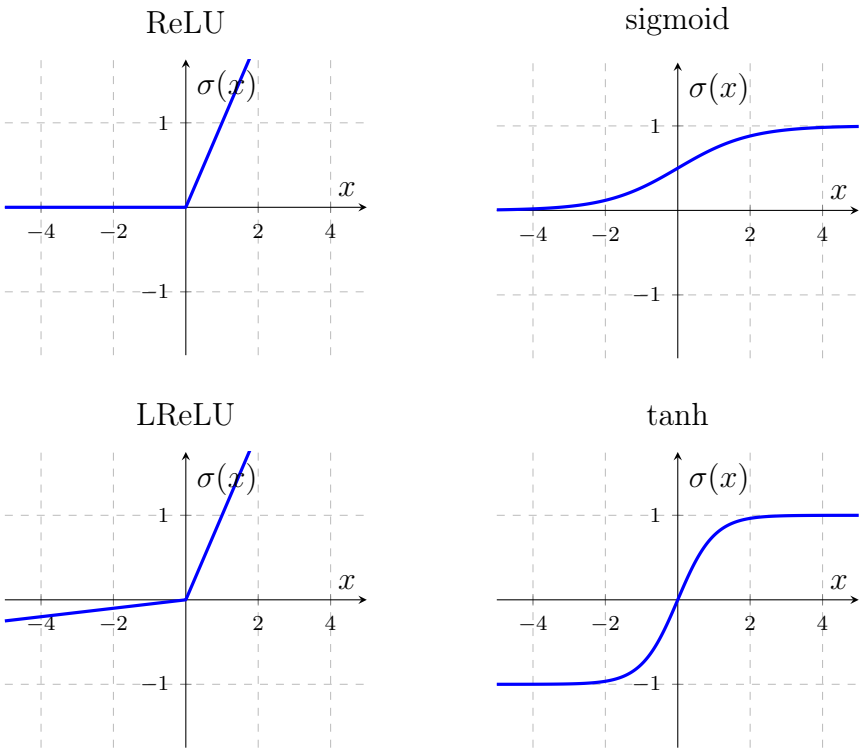
$$\sigma(x) = \begin{cases} x, & \text{kun } x \geq 0 \\ \alpha x, & \text{kun } x < 0 \end{cases}, \tag{LReLU}$$

jossa α on pieni positiivinen vakio, tyypillisesti 0.01. Tavallisesta ReLU:sta poiketen LReLU ei kokonaan sammuta neuronia, vaikka se saisi syötteenään negatiivisen arvon. Toinen tavallisen ReLU:n variantti on He et al. (2015) esittelemä parametrinen ReLU (*parametric ReLU*, PReLU), joka on muuten vastaava kuin LReLU, mutta α on vakion sijaan parametri, joka estimoidaan neuroverkon opetuksen yhteydessä.

Eri aktivaatiofunktioiden kuvaajia on esitetty kuviossa 4.

3.3.2 Ulostulokerroksen aktivaatiofunktiot

Edellä esitetyt aktivaatiofunktiot ovat toimivia vaihtoehtoja neuroverkon piilokerrosten aktivaatiofunktioiksi. Neuroverkon ulostulokerroksella aktivaatiofunktiolta vaaditaan kui-



Kuvio 4: *ReLU:n, sigmoidifunktion, LReLU:n* ($\alpha = 0.05$) *ja hyperbolisen tangentin kuvaajat.*

tenkin hieman erilaisia ominaisuuksia. On esimerkiksi toivottavaa, että neuroverkon ulostulokerroksella laskettaisiin havainnon eri luokkiin kuulumisen todennäköisyydet, sillä näitä todennäköisyyksiä voidaan hyödyntää myös luokittelun tuloksia esitettäessä.

Oletetaan seuraavaksi, että luokittelutilanteessa luokkia on M kappaletta. Neuroverkon ulostulokerroksella on nyt M neuronia, yksi kutakin luokkaa kohden. Ulostulokerroksen aktivaatiofunktioksi valitaan softmax-funktio, joka kuvaa kaikkien M:n luokan todennäköisyysjakauman. Softmax-kuvauksen arvo eli todennäköisyys, että yksittäinen havainto kuuluu luokkaan l, $l = 1, 2, \ldots, M$, on

$$\text{softmax}(\boldsymbol{a}^{(k)})_l = \frac{\exp\left(\boldsymbol{a}_l^{(k)}\right)}{\sum_{l=1}^{M} \exp\left(\boldsymbol{a}_l^{(k)}\right)}, \qquad \text{(softmax)}$$

jossa $\boldsymbol{a}^{(k)}$ sisältää neuroverkon ulostulokerroksen neuronit ja $\boldsymbol{a}_l^{(k)}$ on tämän vektorin l:nnes alkio. Todennäköisyys, että havainto kuuluu luokkaan l, on siis luokkaa vastaavan ulostulokerroksen neuronin exp-muunnos jaettuna kaikkien ulostulokerroksen neuronien exp-muunnosten summalla. Kun softmax-operaatio on tehty ulostulokerroksella, voidaan saatujen todennäköisyyksien mukaisesti muodostaa luokkavektori $\boldsymbol{y}$, jossa on tasan yksi ykkönen sen luokan kohdalla, jonka ennustettu todennäköisyys on suurin (ja muuten nollia).

Softmax-aktivaatiofunktio on hyödyllinen, sillä kyseisen aktivaatiofunktion tulos on tulkittavissa tiettyyn luokkaan kuulumisen todennäköisyytenä. Luokittelun voi myös toteuttaa yksinkertaisemminkin, jos todennäköisyystulkintaa ei haluta tai tarvitse hyödyntää. Jos vaste on binäärinen 0/1-muuttuja, ulostulokerroksella erittäin yksinkertainen aktivaatiofunktio on merkkifunktio $\text{sign}(x)$, jota käytettäessä ulostulokerroksen yhdestä

ainoasta neuronista riittää tarkastella sen etumerkkiä. Neuroverkko voidaan esimerkiksi opettaa niin, että positiivinen ulostulokerroksen neuroni johtaa havainnon luokitteluun luokkaan 0 ja negatiivinen neuroni vastaavasti luokkaan 1. Binäärisen vasteen tapauksessa ulostulokerroksen aktivaatiofunktiona voidaan käyttää myös logistisesta regressiosta tuttua logit- eli sigmoidifunktiota. Tällöin ulostulokerroksella käytetään vain yhtä neuronia, jonka arvo on todennäköisyys $\mathrm{P}(Y = 1 \mid \boldsymbol{x})$. Jos tämä todennäköisyys on suuri (> 0.5), havainto luokitellaan luokkaan 1. Muulloin havainto luokitellaan luokkaan 0.

Taulukossa 1 on koottuna kaikki ulostulokerroksella tapahtuvat operaatiot ja niiden merkinnät tilanteessa, kun aktivaatiofunktio on softmax-funktio. Ulostulokeroksella tapahtuu luokittelua tehtäessä kaikki kolme vaihetta kerralla. Tietyissä tilanteissa (esimerkiksi neuroverkon opetusvaiheessa, ks. luku 4.4.2) voidaan tarkastella kuitenkin vain kahta ensimmäistä vaihetta, jolloin neuroverkossa ulostulokerroksen arvot ovat luokkavektorin $\boldsymbol{y}$ alkioiden sijaan vektorin $\boldsymbol{h}^{(k)}$ alkioita.

Taulukko 1: *Ulostulokerroksen vaiheet. Taulukon ensimmäinen rivi esittää ulostulokerroksen eri tapahtumat: Ulostulokerros saa syötteenään vektorin $\boldsymbol{a}^{(k)}$, jonka jälkeen se operoidaan aktivaatiofunktiolla (softmax) ja lopuksi muutetaan vektoriksi $\boldsymbol{y}$. Toinen rivi esittää kunkin vaiheen matemaattiset operaatiot ja saadun vektorin alkioiden arvojoukon.*

Ulostulokerroksen syöte $\boldsymbol{a}^{(k)}$	$\rightarrow$	Aktivaatiofunktion ulostulo $\boldsymbol{h}^{(k)}$	$\rightarrow$	Neuroverkon ulostulo
$\boldsymbol{a}^{(k)} = \boldsymbol{W}_k^T \boldsymbol{h}^{(k-1)} + \boldsymbol{b}_k$ $\boldsymbol{a}_l^{(k)} \in \mathbb{R}$	$\rightarrow$	$\boldsymbol{h}^{(k)} = \sigma\left(\boldsymbol{a}^{(k)}\right)$ $\boldsymbol{h}_l^{(k)} \in (0, 1)$	$\rightarrow$	$\boldsymbol{y}$ $\boldsymbol{y}_l \in \{0, 1\}$

4 Neuroverkon opettaminen

Ulostulokerros ja sen aktivaatiofunktio ovat viimeiset komponentit neuroverkossa ennen varsinaisen luokittelun tulosta. Edellä esitettyjen tietojen pohjalta luokittelu voidaan jo suorittaa, jos neuroverkon kaikki painot ovat opetettu valmiiksi. Esitetään seuraavaksi neuroverkon opettamisen periaatteet ja opetuksessa käytetyt algoritmit, joita hyödyntämällä neuroverkon voi opettaa alusta asti luokitteluongelman ratkaisuksi.

4.1 Opetus-, testi- ja validointiaineistot

Neuroverkon opettamisprosessissa neuroverkon painoja säädetään siten, että neuroverkko lopulta toimii valitussa sovelluksessa mahdollisimman hyvin. Yksi tapa säätää painoja on valita ne täysin itse ”käsin”. Voidaan kuitenkin helposti kuvitella, että tällaisten neuroverkkojen laatiminen vie paljon aikaa ja laadittu neuroverkko toimii vain juuri siinä sovelluksessa, jota varten se on laadittu (Goodfellow et al. 2016, 166).

Neuroverkot kuuluvat kuitenkin ohjattujen oppimismenetelmien joukkoon samoin kuin monet muut koneoppimismenetelmät. Painoja ei siis tarvitse valita itse, vaan ne niin sanotusti opetetaan aineiston perusteella käyttäen erilaisia opetusalgoritmeja.

Opetusvaiheessa neuroverkolle annetaan syötteeksi osa-aineisto, jota kutsutaan opetusaineistoksi. Opetusaineisto sisältää sekä havainnon piirteet $\boldsymbol{x}$ että myös tiedon havainnon todellisesta luokasta Y. Toisin sanoen $(Y, \boldsymbol{x})$ on kokonaan tiedossa. Koska luokat ovat tiedossa, voidaan neuroverkon virhettä arvioida käyttämällä sopivaa virhefunktiota. Virhefunktion rooli ohjattujen oppimismenetelmien opetuksessa on kertoa, kuinka paljon juuri opetettu malli on pielessä. Mallin näkökulmasta toivottava tilanne olisi, että opetusvirhe olisi mahdollisimman pieni. (Hastie et al. 2009, 29.) Jos virhe on liian suuri, sitä voi pienentää säätämällä joko mallin hyperparametreja tai varsinaisia (opetettavia) parametreja. Hyperparametreja voi säätää mallin käyttäjä suoraan itse, kun taas mallin varsinaisia opetettavia parametreja voi opettaa lisää jatkamalla opetusprosessia.

Kun neuroverkko on aikanaan opetettu käyttäen opetusaineistoa, mallia voidaan testata käyttäen toista erillistä osa-aineistoa eli testiaineistoa. Testiaineisto on täysin samankaltainen kuin opetusaineistokin, eli se sisältää sekä havaintojen piirteet että todelliset luokat. Tyypillisesti testiaineiston koko on kuitenkin pienempi verrattuna opetusaineistoon. Toisin kuin opetusaineisto, testiaineistoa ei ole käytetty neuroverkon opetuksessa lainkaan, joten testiaineisto on neuroverkon näkökulmasta katsottuna täysin vierasta aineistoa. Siispä, kun neuroverkkoa testataan opetuksen jälkeen, testiaineiston havainnot luokitellaan ja ennustettujen luokkien sekä tiedossa olevien todellisten luokkien perusteella voidaan tarkastella esimerkiksi neuroverkon luokittelutarkkuutta tai -virhettä ja sitä, kuinka hyvin neuroverkon luokittelukyky yleistyy sille ennalta tuntemattoman aineiston tapauksessa.

Kolmas osa-aineisto, jota voidaan käyttää neuroverkkojen opettamisessa, on niin sanottu validointiaineisto. Siinä missä testiainestoa käytetään lopullisen mallin yleistettävyyden tarkastelussa, validointiaineiston avulla on tarkoitus tutkia mallin ja sen hyperparametrien valintaa. (Aggarwal 2020, 168.) Validointiaineiston käytön periaatetta kuvataan tarkemmin luvussa 4.5.1.

On tärkeää huomata, että opetuksessa käytettyä opetusaineistoa ei käytetä mallin testi- tai validointiaineistoina. Vastaavasti testi- tai validointiaineistoja ei pidä missään vaiheessa käyttää mallin opetusaineistoina. Mikäli näin tehtäisiin, niin väistämättä saataisiin pieni luokitteluvirhe. Pieni luokitteluvirhe opetusaineistolla ei takaa, että neuroverkko toimisi yhtä tarkasti jokaisessa tilanteessa. Siis, ideaalisessa tilanteessa mallin ja sen hyperparametrien valinnassa käytetään opetusaineistosta erillistä validointiaineistoa, ja lopullisen mallin luokitteluvirheen tarkastelussa erillistä testiaineistoa. Näin saatu pieni opetusvirhe antaa aihetta ymmärtää, että neuroverkko on kykenevä yleistämään luokittelun myös opetusaineiston ulkopuolelle (Aggarwal 2020, 29). Koko aineiston jakaminen opetus-, testi- ja validointiaineistoihin yleisesti ottaen mahdollistaa sen, että neuroverkkoluokittimesta tulee toimiva tilastollinen malli, joka yleistyy hyvin. Jaolla myös ennaltaehkäistään neuroverkkojen opetukseen liittyviä ongelmia, kuten ylioppimista, joihin palataan myöhemmin luvussa 4.5.

4.2 Ristientropia ja neuroverkon virhefunktio

Neuroverkon virhefunktion valintaan vaikuttaa sen sovelluskohde. Luokittelumenetelmille ristientropia (*cross entropy*, CE) on sopiva virhefunktio, jonka laskemiseksi tarvitaan havainnon luokkavektori $\boldsymbol{y}$. Ristientropia määritellään seuraavasti:

$$\mathrm{CE}(\boldsymbol{y}) = -\sum_{l=1}^{M} y_l \log\left[\mathrm{P}(Y = l \mid \boldsymbol{x})\right],$$

jossa y_l on l:nnes alkio todellista luokkaa vastaavassa luokkavektorissa $\boldsymbol{y}$, M on luokkien lukumäärä ja todennäköisyys $\mathrm{P}(Y = l \mid \boldsymbol{x})$ on todennäköisyys sille, että havainnon todellinen luokka Y on l. Näin ollen y_l saa arvon yksi ainoastaan kerran. Lisäksi todennäköisyys $\mathrm{P}(Y = l \mid \boldsymbol{x})$ saadaan softmax-aktivaatiofunktiolla neuroverkon ulostulokerroksella

$$\mathrm{P}(Y = l \mid \boldsymbol{x}) = \mathrm{softmax}\left(\boldsymbol{a}^{(k)}\right)_l.$$

(Aggarwal 2018, 118; James et al. 2021, 410.) Kuten tiedetään, vektori $\boldsymbol{a}^{(k)}$ saadaan aiempien neuroverkon kerrosten neuronien ja painojen mukaisesti: $\boldsymbol{a}^{(k)} = \boldsymbol{W}_k^T \boldsymbol{h}^{(k-1)} + \boldsymbol{b}_k$. Toisin sanoen, todennäköisyyteen $\mathrm{P}(Y = l \mid \boldsymbol{x})$ vaikuttavat kaikki neuroverkon aiempien kerrosten painomatriisien $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_k$ painot ja mahdollisesti myös vakioneuronivektoreiden $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k$ painot, mikäli vakioneuroneita on käytössä. Olkoot nämä kaikki painot samassa parametrivektorissa $\boldsymbol{\theta}$, jonka pituus saadaan piilokerrosten leveyksistä: Koska vakioneuroneita on mahdollisesti vain piilokerroksilla $1, \ldots, k-1$, vakioneuronien painojen lukumäärä on kerrosten $2, \ldots, k$ leveyksien summa $p_2 + p_3 + \cdots + p_k$. Edellisessä summassa summattava voi olla myös nolla, jos vakioneuroni ei ole käytössä vastaavalla kerroksella. Vastaavasti painomatriiseissa olevien painojen lukumäärä on $p_0 p_1 + p_1 p_2 + \cdots + p_{k-1} p_k$, jossa p_0 viittaa syötekerroksen leveyteen. Parametrivektorin $\boldsymbol{\theta}$ pituudeksi saadaan näin vakioneuronien painojen ja painomatriisien painojen lukumäärien summa

$$(p_2 + p_3 + \cdots + p_k) + (p_0 p_1 + p_1 p_2 + \cdots + p_{k-1} p_k).$$

Nyt virhefunktiota varten otetaan käyttöön merkintä

$$\mathrm{P}(Y = l \mid \boldsymbol{x}) = \mathrm{softmax}\left(\boldsymbol{a}^{(k)}\right)_l = p_l(\boldsymbol{\theta}).$$

Kun ristientropiaa käytetään virhefunktiona, tulee yksittäisten havaintojen virheet summata yhteen. Olkoon havaintoja n kappaletta, jolloin virhefunktio L voidaan kirjoittaa edellä kuvattujen merkintöjen mukaisesti parametrien $\boldsymbol{\theta}$ funktiona seuraavasti:

$$\begin{aligned} L(\boldsymbol{\theta}) &= -\sum_{j=1}^{n}\sum_{l=1}^{M} y_{jl} \log\left[\mathrm{softmax}\left(\boldsymbol{a}^{(k)}\right)_{jl}\right] \\ &= -\sum_{j=1}^{n}\sum_{l=1}^{M} y_{jl} \log\left[\mathrm{softmax}\left(\boldsymbol{W}_k^T \boldsymbol{h}^{(k-1)} + \boldsymbol{b}_k\right)_{jl}\right] \\ &= -\sum_{j=1}^{n}\sum_{l=1}^{M} y_{jl} \log\left[p_{jl}(\boldsymbol{\theta})\right], \end{aligned} \tag{5}$$

jossa indeksi $j = 1, \ldots, n$ viittaa yksittäiseen havaintoon.

4.3 Stokastinen gradienttimenetelmä

Opetusvaiheessa neuroverkolle syötetään opetusaineiston piirteiden havaittuja arvoja, ja tiedossa olevien todellisten luokkien sekä neuroverkon laskemien softmax-todennäköisyyksien avulla lasketaan opetusvirhe. Tätä kierrosta toistetaan, mutta aina kierrosten välissä opetusalgoritmi muuttaa neuroverkon parametrien $\boldsymbol{\theta}$ arvoja siten, että virhefunktio lähestyy sen minimiä. (Aggarwal 2018, 21.) Edellä kuvattu algoritmi on gradienttimenetelmä (*gradient descent*). Gradienttimenetelmä muokkaa neuroverkon parametrivektoria $\boldsymbol{\theta}$ virhefunktion gradienttiin perustuen: Koska parametrien suhteen lasketun virhefunktion gradientti kertoo virhefunktion suurimman kasvunopeuden ja kasvun suunnan, niin tämän gradienttivektorin vastavektorin suunta määrää vastaavasti suunnan, johon siirryttäessä virhefuntio vähenee eniten. Menetelmä voidaan kirjoittaa yleisessä tilanteessa iteratiivisena kaavana

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \, \nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}_t),$$

jossa $L(\boldsymbol{\theta}_t)$ on virhefunktio, $\nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}_t)$ on virhefunktion gradienttivektori parametrien $\boldsymbol{\theta}_t$ suhteen, vakio η on niin sanottu oppimisnopeus (*learning rate*) ja t on iteraatioita kuvaava indeksi. Sijoittamalla kaavan (5) mukainen ristientropian lauseke virhefunktion paikalle saadaan luokitteluongelmalle soveltuva gradienttimenetelmä:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \, \nabla_{\boldsymbol{\theta}_t} \left\{ -\sum_{j=1}^{n} \sum_{l=1}^{M} y_{jl} \log \left[p_{jl} \left(\boldsymbol{\theta}_t \right) \right] \right\}. \tag{6}$$

Gradienttimenetelmän oppimisnopeus η on pieni positiivinen luku, jolla voidaan kontrolloida gradienttivektorin suuruuden vaikutusta parametrien muutokseen gradienttimenetelmässä. Oppimisnopeus vaikuttaa ratkaisevasti gradienttimenetelmän konvergenssiin. Liian pieni oppimisnopeus vaatii useita iteraation askelia ennen kuin menetelmä konvergoi. Liian suuri opetusnopeus puolestaan pahimmillaan johtaa prosessin hajaantumiseen, tai tilanteeseen, että prosessi vaihtelee virhefunktion minimin ympärillä. (LeCun et al. 2012.) On lisäksi tavanomaista, että opetusnopeutta pienennetään iteraatioiden lukumäärän t kasvaessa (Goodfellow et al. 2016, 290).

Edellä kaavassa (6) gradienttivektori lasketaan koko aineiston yli, toisin sanoen summaus käy koko aineiston koon n yli. Syväoppimismenetelmissä opetusaineiston koko on tyypillisesti todella iso, joten jo pelkän gradientin laskeminen voi kestää huomattavan pitkään. Gradientin laskentaa voi nopeuttaa, jos aineiston kokoa pienennetään tarkoituksella. Tällöin gradienttimenetelmää kutsutaan stokastiseksi gradienttimenetelmäksi (*stochastic gradient descent*, SGD). Stokastisessa gradienttimenetelmässä virhefunktion gradienttia ei lasketa tarkasti koko opetusaineistoa käyttämällä, vaan gradientti arvioidaan pienemmällä osa-aineistolla eli erällä (*batch*). Stokastista gradienttimenetelmää käytettäessä gradientin laskeminen on nopeaa, ja gradientti silti kuvaa virhefunktion vähenemisen suuntaa tarpeeksi hyvin. (LeCun et al. 2015.)

Yksi erä muodostetaan valitsemalla U kappaletta opetusaineiston havaintoja satunnaisesti siten, että valittujen havaintojen indeksit kootaan joukoksi $B = \{j_1, j_2, \ldots, j_U\}$.

Joukon B kokoa U kutsutaan kyseisen erän eräkooksi (*batch size*), ja se on neuroverkon käyttäjän päätettävissä. (Aggarwal 2018, 123.) Yhden erän havaintojen valinta tehdään ilman takaisinpalautusta, jolloin yksittäinen havainto voi tulla valituksi vain yhteen erään. Stokastisen gradienttimenetelmän mukainen iteratiivinen parametrien päivitys tapahtuu muuten vastaavasti kuin tavallisessa gradienttimenetelmässä, mutta koko aineiston sijaan summataan vain joukon B yli:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \left\{ -\sum_{j \in B} \sum_{l=1}^{M} y_{jl} \log \left[p_{jl} \left(\boldsymbol{\theta}_t \right) \right] \right\}. \tag{7}$$

Stokastista gradienttimenetelmää käytetään käytännössä aina siten, että opetusaineiston jokainen erä syötetään useita kertoja SGD:n läpi. Tällöin neuroverkon parametrit päivittyvät joka kierroksella hieman enemmän kohti optimaalisinta tilannetta. Yksittäisen havainnon yhtä kierrosta SGD:n läpi sanotaan epookiksi (*epoch*) (Aggarwal 2018, 7). Epookkien määrä on neuroverkon käyttäjän päätettävissä, ja niiden sopivaan lukumäärään vaikuttavat merkittävästi eräkoko, aineiston koko, tietokoneen laskentakapasiteetti sekä se, kuinka monimutkainen neuroverkko on käytössä: Yksinkertaisessa neuroverkossa ja kohtuullisella aineiston koolla epookkeja voidaan ajaa SGD:n läpi useita satoja vain muutamissa sekunneissa. Monimutkaisemmassa neuroverkossa ja suurella aineistolla jo yhden epookin vaatimien operaatioiden laskeminen voi viedä jo useita minuutteja.

Lopuksi huomautetaan, että parametrivektoria $\boldsymbol{\theta}$ käytetään vain symbolisessa merkityksessä gradienttimenetelmien kaavoissa. Nimittäin, myöhemmin luvussa 4.4.3 näytetään, että todellisuudessa virhefunktion gradienttia ei lasketa koko parametrivektorin $\boldsymbol{\theta}$ suhteen, vaan sen sijaan lasketaan gradientit $\nabla_{\boldsymbol{W}_i}$ ja $\nabla_{\boldsymbol{b}_i}$, eli gradientti lasketaan kerralla vain tietyn painomatriisin $\boldsymbol{W}_i$ painojen tai tietyn vakioneuronivektorin $\boldsymbol{b}_i$ painojen suhteen. Siispä painoja päivitetään vain yhdellä kerroksella vuorollaan.

4.4 Vastavirta-algoritmi

Eteenpäin syöttävä neuroverkko on kaikessa yksinkertaisuudessaan monimutkainen epälineaarinen malli, jonka hyödyntämistä varten mallin parametrit pitää estimoida. Parametrien estimointi, eli neuroverkon painojen opettaminen, tapahtuu käyttäen niin sanottua vastavirta-algoritmiä (*backpropagation algorithm*, *backprop*). On syytä heti aluksi huomata, että vastavirta-algoritmi-termillä voidaan tarkoittaa useita eri asioita lähteestä riippuen: Vastavirta-algoritmilla tarkoitetaan toisinaan koko neuroverkon opettamisessa käytettyä algoritmia, ja toisinaan se voi viitata ainoastaan virhefunktion gradientin laskemiseen. Joka tapauksessa neuroverkon opettaminen kokonaisuudessaan on laaja ja monimutkainen matemaattinen operaatio, joka koostuu useista muista algoritmeista ja menetelmistä.

Vastavirta-algoritmi jaetaan kahteen vaiheeseen, jotka ovat etenemisvaihe (*forward phase*) ja peruutusvaihe (*backward phase*). Peruutusvaihe on huomattavasti monimutkaisempi vaihe matemaattisesti, sillä tässä vaiheessa tapahtuu neuroverkon painojen päivitys eli neuroverkon varsinainen oppiminen. Tämä alaluku keskittyy etenkin vastavirta-

algoritmin peruutusvaiheeseen ja siinä tapahtuvaan virhefunktion gradientin laskentaan. Tässä tutkielmassa gradientin laskennalle esitetään kaksi tapaa: Ensiksi, luvussa 4.4.2 gradientti lasketaan yksittäisen painon suhteen differentiaalilaskennan ketjusääntöä ja dynaamista ohjelmointia hyväksi käyttäen. Toiseksi, luvussa 4.4.3 gradientin laskentaa tarkastellaan laajemmin ottamalla käyttöön vektorikeskeinen neuroverkkorakenne (*vector-centric architecture*) ja laskemalla gradientti kerralla useiden painojen suhteen. Luvun lopuksi esitetään sekä etenemis- että peruutusvaihe lyhyesti algoritminotaatiolla.

4.4.1 Etenemisvaihe

Etenemisvaiheessa syöte ja laskutoimitukset siirtyvät syötekerrokselta neuroverkon kerrosten läpi eteenpäin kohti ulostulokerrosta. Piilokerroksilla lasketaan neuronien arvot niillä painoilla, jotka neuroverkko kullakin hetkellä sisältää. Etenemisvaiheessa painoja ei siis vielä päivitetä, eikä neuroverkon oppimista tapahdu. Lopuksi, ulostulokerroksella estimoidaan opetusaineiston havainnoille softmax-todennäköisyydet sekä lasketaan virhefunktion L arvo. (Aggarwal 2018, 113.) Etenemisvaiheen neuronien ja virhefunktion arvon laskennat ovat jo esitelty aiemmin tässä tutkielmassa luvuissa 3.2 ja 4.2.

4.4.2 Peruutusvaihe: ketjusääntö ja dynaaminen ohjelmointi

Peruutusvaiheessa lasketaan virhefunktion gradientti suhteessa aiempien piilokerrosten painoihin ja päivitetään neuroverkon painot aiemmin kuvatulla stokastisella gradienttimenetelmällä (SGD). Gradientin laskennassa käytetään differentiaalilaskennan ketjusääntöä, jossa operaatiot tapahtuvat nyt ulostulokerrokselta lähtien kohti syötekerrosta. Neuroverkossa edetään nyt siis peruuttaen.

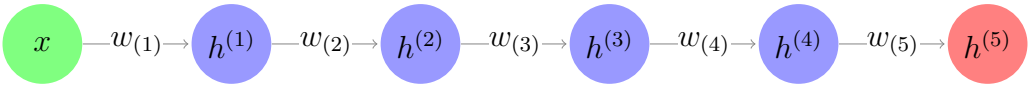
Tässä alaluvussa tarkastellaan aluksi differentiaalilaskennan ketjusääntöä neuroverkkorakenteiden tapauksessa. Lisäksi esitetään dynaaminen ohjelmointi, jonka tekniikoita käytetään ketjusäännön tehokkaassa laskennassa.

Esitellään ketjusääntö ensin reaaliluvuille. Olkoon $u \in \mathbb{R}$ ja $f, g : \mathbb{R} \to \mathbb{R}$ funktioita. Olkoon lisäksi $z = f(v)$ ja $v = g(u)$, jolloin $z = f(g(u))$. Nyt ketjusäännön mukaisesti saadaan laskettua kuvauksen z derivaatta parametrin u suhteen:

$$\frac{dz}{du} = \frac{dz}{dv}\frac{dv}{du}. \tag{8}$$

(Goodfellow et al. 2016, 201–203.)

Ketjusääntö voidaan yleistää myös tilanteeseen, jossa funktioiden ketjutus esitetään neuroverkoista tutulla graafilla: Oletetaan esimerkiksi, että neuroverkon jokaisella kerroksella on vain yksi neuroni, piilokerroksia on neljä kappaletta ja vakioneuroneita ei ole käytössä. Merkitään piilokerrosten neuroneita $h^{(1)}, \ldots, h^{(4)}$ ja olkoon ulostulokerroksen arvo aktivaatiofunktion laskemisen jälkeen $h^{(5)}$. Kerrosten (i) ja $(i+1)$ välinen paino on $w_{(i)}$, $i = 1, \ldots, 5$. Tämän neuroverkon graafi on esitetty kuviossa 5.



Kuvio 5: *Yksinkertainen neuroverkkorakenne, jossa on vain yksi polku.*

Neuroverkon yhdistettyjen funktioiden $f^{(i)}$ rakenne voidaan kirjoittaa seuraavasti:

$$\begin{aligned} h^{(1)} &= f^{(1)}\left(x; w_{(1)}\right), \\ h^{(2)} &= f^{(2)}\left(h^{(1)}; w_{(2)}\right), \\ h^{(3)} &= f^{(3)}\left(h^{(2)}; w_{(3)}\right), \\ h^{(4)} &= f^{(4)}\left(h^{(3)}; w_{(4)}\right), \\ h^{(5)} &= f^{(5)}\left(h^{(4)}; w_{(5)}\right). \end{aligned}$$

Tämän lisäksi virhefunktio on tässä esimerkissä (yhdelle havainnolle ja yhdelle luokalle) $L = -\log\left[h^{(5)}\right]$. Kiinnostuksen kohteena on laskea virhefunktion gradientti jokaisen painon suhteen. Esimerkiksi ensimmäisen ja toisen piilokerroksen välisen painon $w_{(2)}$ suhteen laskettu gradientti saadaan ketjusäännön mukaisesti

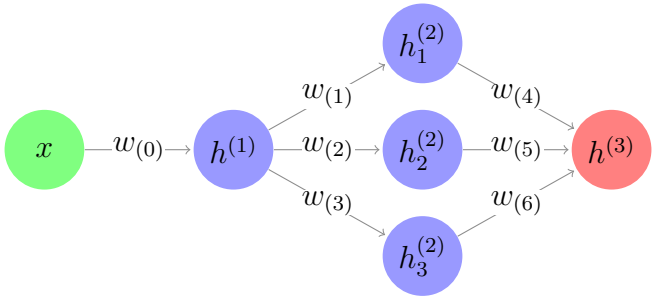
$$\frac{\partial L}{\partial w_{(2)}} = \frac{\partial L}{\partial h^{(5)}} \frac{\partial h^{(5)}}{\partial h^{(4)}} \frac{\partial h^{(4)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial w_{(2)}}.$$

Yleisesti, jos neuroverkossa on k kappaletta kerroksia, virhefunktion gradientti minkä tahansa neuroverkon painon $w_{(i)}$ suhteen on

$$\frac{\partial L}{\partial w_{(i)}} = \frac{\partial L}{\partial h^{(k)}} \left[\prod_{j=i}^{k-1} \frac{\partial h^{(i+1)}}{\partial h^{(i)}} \right] \frac{\partial h^{(i)}}{\partial w_{(i)}}. \tag{9}$$

Kaava (9) on samalla ketjusäännön (kaava (8)) yleistys useammalle kuin kahdelle funktiolle. (Aggarwal 2018, 22.)

Edellisessä esimerkissä miltä tahansa kerrokselta lähtee ainoastaan yksi yhtenäinen polku ulostulokerrokselle, minkä takia gradientti voidaan laskea suhteellisen yksinkertaisesti. Tilanne ei siten vastaa lainkaan syviä neuroverkkorakenteita, joissa polkuja on useita. Esimerkiksi kuvion 6 neuroverkkorakenteessa yhdellä piilokerroksella on kolme neuronia, jolloin polkujen lukumäärä syötekerrokselta ulostulokerrokselle nousee kolmeen.



Kuvio 6: *Esimerkki neuroverkkorakenteesta, jossa on useita polkuja.*

Kuvion 6 graafissa kaikki piilokerrosten neuronit ovat vastaavassa asemassa kuin kuvion 5 tilanteessa, sillä neuroniin tulee ja siitä lähtee vain yksi polku. Ulostulokerroksella tilanne on eri, sillä nyt ulostulokerroksen neuroniin saapuu kolme eri polkua. Yhdistettyjen funktioiden rakenne voidaan jälleen kirjoittaa auki:

$$
\begin{aligned}
h^{(1)} &= f^{(1)}\left(x; w_{(0)}\right), \\
h_1^{(2)} &= f_1^{(2)}\left(h^{(1)}; w_{(1)}\right), \\
h_2^{(2)} &= f_2^{(2)}\left(h^{(1)}; w_{(2)}\right), \\
h_3^{(2)} &= f_3^{(2)}\left(h^{(1)}; w_{(3)}\right), \\
h^{(3)} &= f^{(3)}\left(h_1^{(2)}, h_2^{(2)}, h_3^{(2)}; w_{(4)}, w_{(5)}, w_{(6)}\right).
\end{aligned}
$$

Virhefunktio on $L = -\log\left[h^{(3)}\right]$. Tilanteessa, jossa polkuja on useita, gradientin laskeminen tapahtuu käyttäen moniulotteista ketjusääntöä (*multivariate chain rule*). Kuvion 6 graafissa on polut $h^{(1)} \to h_1^{(2)} \to h^{(3)}$, $h^{(1)} \to h_2^{(2)} \to h^{(3)}$ ja $h^{(1)} \to h_3^{(2)} \to h^{(3)}$. Moniulotteisen ketjusäännön mukaan gradientti painon $w_{(0)}$ suhteen saadaan summaamalla jokaiselta polulta lasketut paikalliset (lokaalit) gradientit yhteen:

$$
\begin{aligned}
\frac{\partial L}{\partial w_{(0)}} &= \frac{\partial L}{\partial h^{(3)}}\frac{\partial h^{(3)}}{\partial h_1^{(2)}}\frac{\partial h_1^{(2)}}{\partial h^{(1)}}\frac{\partial h^{(1)}}{\partial w_{(0)}} + \frac{\partial L}{\partial h^{(3)}}\frac{\partial h^{(3)}}{\partial h_2^{(2)}}\frac{\partial h_2^{(2)}}{\partial h^{(1)}}\frac{\partial h^{(1)}}{\partial w_{(0)}} + \frac{\partial L}{\partial h^{(3)}}\frac{\partial h^{(3)}}{\partial h_3^{(2)}}\frac{\partial h_3^{(2)}}{\partial h^{(1)}}\frac{\partial h^{(1)}}{\partial w_{(0)}} \\
&= \frac{\partial L}{\partial h^{(3)}}\left[\frac{\partial h^{(3)}}{\partial h_1^{(2)}}\frac{\partial h_1^{(2)}}{\partial h^{(1)}} + \frac{\partial h^{(3)}}{\partial h_2^{(2)}}\frac{\partial h_2^{(2)}}{\partial h^{(1)}} + \frac{\partial h^{(3)}}{\partial h_3^{(2)}}\frac{\partial h_3^{(2)}}{\partial h^{(1)}}\right]\frac{\partial h^{(1)}}{\partial w_{(0)}}.
\end{aligned}
$$

(Aggarwal 2018, 107–109.)

Moniulotteinen ketjusääntö yleistyy vastaavasti tilanteeseen, jossa polkuja on paljon enemmän. Neuroverkkorakenteessa polkujen lukumäärä kuitenkin kasvaa eksponentiaalisesti neuronien ja kerrosten määrän kasvaessa. Esimerkiksi, jos neuroverkossa on viisi piilokerrosta ja kullakin kerroksella on 20 neuronia, niin tällöin eri polkuja syötteen yhdeltä neuronilta ulostulokerroksen yhdelle neuronille on $20^5 = 3\,200\,000$ kappaletta. Näin ollen, gradientin laskeminen suoraan summaamalla on niin hidasta, että käytännössä syvien neuroverkkojen käyttö tällä tavoin on mahdotonta.

Gradientin laskemista voidaan nopeuttaa käyttämällä dynaamisen ohjelmoinnin periaatteita. Dynaaminen ohjelmointi (*dynamic programming*) on optimointimenetelmä, jossa optimointiongelma jaetaan pienemmiksi osa-ongelmiksi ja näille haetaan optimaalisin ratkaisu. Osa-ongelmien ratkaisut tallennetaan muistiin ja niitä käytetään sekä muiden osa-ongelmien että myös lopulta alkuperäisen optimointiongelman ratkaisussa. Osa-ongelmien on oltava luonteeltaan yhteneviä sekä keskenään että myös alkuperäisen optimointiongelman kanssa, jolloin jokainen mahdollinen ratkaisu alkuperäiseen optimointiongelmaan sisältää myös jokaisen osa-ongelman ratkaisun. Toisin sanoen, etsimällä optimaalisia ratkaisuja pienemmille osa-ongelmille, voidaan rekursiivisesti löytää ratkaisu myös alkuperäiselle optimointiongelmalle. (Aggarwal 2020, 248.)

Määritelmästään huolimatta dynaamisen ohjelmoinnin periaatetta sovelletaan myös muidenkin kuin optimointiongelmien ratkaisussa. Yleisesti ottaen kaikkia sellaisia me-

netelmiä, joissa alkuperäinen ongelma jaetaan osa-ongelmiin, joiden ratkaisujen avulla ratkaistaan alkuperäinen ongelma, voidaan pitää dynaamisena ohjelmointina. (Aggarwal 2020, 250.) Ketjusääntö neuroverkossa tukeutuu vahvasti dynaamisen ohjelmoinnin periaatteeseen, sillä gradientti useiden neuroverkon kerrosten ylitse voidaan toteuttaa lisäämällä kerroksia mukaan ketjusääntöön yksi kerrallaan. Ideana on laskea virhefunktion gradientteja jokaisella neuroverkon kerroksella ulostulokerrokselta lähtien. Kukin laskettu gradientti voidaan tallentaa muistiin, minkä jälkeen uuden kerroksen gradientin laskemisessa voidaan hyödyntää tallennettuja tuloksia aiempien kerrosten gradienteista. Näin aikaa ei tarvitse käyttää samojen gradienttien laskemiseen toistuvasti yhä uudelleen.

Tarkastellaan seuraavaksi gradienttien laskentaa neuroverkossa dynaamisen ohjelmoinnin periaatteella. Seuraava esitys noudattaa julkaisun Aggarwal (2018) luvun 3.2 esitystä.

Olkoon neuroverkossa k kerrosta, joiden indeksit ovat $i = 1, \ldots, k$. Oletetaan, että paino $w_{(i)}$ vaikuttaa kerroksen i neuroniin $h^{(i)}$, josta lähtee useita polkuja ulostulokerroksen neuronia $h^{(k)}$ kohti. Moniulotteisen ketjusäännön mukaan virhefunktion gradientti neuronin $h^{(i)}$ suhteen on

$$\frac{\partial L}{\partial h^{(i)}} = \frac{\partial L}{\partial h^{(k)}} \left[\sum_{h^{(i)} \to h^{(k)}} \prod_{j=i}^{k-1} \frac{\partial h^{(j+1)}}{\partial h^{(j)}} \right], \tag{10}$$

jossa summa käy kerralla vain niiden neuronien yli, jotka muodostavat yhden yhtenäisen polun lähtien neuronista $h^{(i)}$ ja päättyen neuroniin $h^{(k)}$ (eli polun $h^{(i)} \to h^{(i+1)} \to \cdots \to h^{(k)}$). Tällä tavalla summassa neuroneiksi $h^{(j+1)}$ ja $h^{(j)}$ tulee valituksi vain yhden tietyn polun muodostavat neuronit. Tulon laskemisen jälkeen summa siirtyy seuraavaan polkuun. Nyt virhefunktion gradientti painon $w_{(i)}$ suhteen on:

$$\frac{\partial L}{\partial w_{(i)}} = \frac{\partial L}{\partial h^{(i)}} \frac{\partial h^{(i)}}{\partial w_{(i)}}. \tag{11}$$

Kaavan (11) jälkimmäinen termi $\frac{\partial h^{(i)}}{\partial w_{(i)}}$ voidaan kirjoittaa auki, kun muistetaan yksittäisen piilokerroksen neuronin ja piilokerroksen aktivaatiofunktion välinen yhteys:

$$h^{(i)} = \sigma\left(a^{(i)}\right) = \sigma\left(w_{(i)} h^{(i-1)}\right).$$

Tällöin suoralla laskulla saadaan

$$\frac{\partial h^{(i)}}{\partial w_{(i)}} = \frac{\partial \sigma\left(a^{(i)}\right)}{\partial w_{(i)}} = \frac{\partial \sigma\left(w_{(i)} h^{(i-1)}\right)}{\partial w_{(i)}} = \sigma'\left(w_{(i)} h^{(i-1)}\right) h^{(i-1)} = \sigma'\left(a^{(i)}\right) h^{(i-1)}.$$

Gradientti siis voidaan kirjoittaa neuroverkon painojen, neuronien ja aktivaatiofunktion derivaatan avulla, jotka ovat kaikki tunnettuja.

Kaavan (11) ensimmäisessä termissä $\frac{\partial L}{\partial h^{(i)}}$ lasketaan virhefunktion gradientti neuronin $h^{(i)}$ suhteen. Tarkoituksena on kirjoittaa myös tämä gradientti neuroverkon painojen, neuronien ja aktivaatiofunktion avulla. Ideana on käyttää dynaamisen ohjelmoinnin periaatetta: Gradientit lasketaan rekursiivisesti siten, että neuroverkon polkuja edetään

taaksepäin ulostulokerroksen neuronista $h^{(k)}$ lähtien ja gradientti lasketaan vuorollaan vain yhdellä kerroksella. Rekursiossa uuden kerroksen suhteen laskettu gradientti saadaan siten kirjoitettua edellisten kerrosten gradienttien avulla. Rekursion kolme ensimmäistä vaihetta, joissa gradientit on laskettu suhteessa ulostulokerroksen $h^{(k)}$ neuroniin sekä kahden viimeisen piilokerroksen $h^{(k-1)}$ ja $h^{(k-2)}$ neuroneihin, ovat näkyvillä alla:

0) Rekursion alustus: Lasketaan $\dfrac{\partial L}{\partial h^{(k)}}$,

1) $$\frac{\partial L}{\partial h^{(k-1)}} = \sum_{h^{(k-1)} \to h^{(k)}} \frac{\partial L}{\partial h^{(k)}} \frac{\partial h^{(k)}}{\partial h^{(k-1)}} = \frac{\partial L}{\partial h^{(k)}} \left[\sum_{h^{(k-1)} \to h^{(k)}} \frac{\partial h^{(k)}}{\partial h^{(k-1)}} \right],$$

2) $$\begin{aligned} \frac{\partial L}{\partial h^{(k-2)}} &= \sum_{h^{(k-2)} \to h^{(k)}} \frac{\partial L}{\partial h^{(k)}} \frac{\partial h^{(k)}}{\partial h^{(k-1)}} \frac{\partial h^{(k-1)}}{\partial h^{(k-2)}} \\ &= \left[\sum_{h^{(k-2)} \to h^{(k-1)}} \frac{\partial h^{(k-1)}}{\partial h^{(k-2)}} \right] \left[\sum_{h^{(k-1)} \to h^{(k)}} \frac{\partial L}{\partial h^{(k)}} \frac{\partial h^{(k)}}{\partial h^{(k-1)}} \right] \\ &= \frac{\partial L}{\partial h^{(k-1)}} \left[\sum_{h^{(k-2)} \to h^{(k-1)}} \frac{\partial h^{(k-1)}}{\partial h^{(k-2)}} \right]. \end{aligned}$$

Summamerkinnät huomioivat jälleen sen, että gradientit lasketaan vuorollaan sellaisilta poluilta, jotka sijaitsevat summamerkinnässä ilmoitettujen neuronien välissä.

Edellistä rekursiota jatkamalla voidaan lopulta laskea gradientti minkä tahansa polun minkä tahansa neuronin suhteen. Oletetaan seuraavaksi, että tietyssä polussa gradientit on laskettu rekursiivisesti neuroniin $h^{(i+1)}$ saakka. Toisin sanoen, $\frac{\partial L}{\partial h^{(i+1)}}$ on tiedossa. Kun seuraavassa rekursion vaiheessa mukaan otetaan neuroni $h^{(i)}$, saadaan gradientille rekursiokaava:

$$\begin{aligned} \frac{\partial L}{\partial h^{(i)}} &= \sum_{h^{(i)} \to h^{(i+1)}} \frac{\partial L}{\partial h^{(i+1)}} \frac{\partial h^{(i+1)}}{\partial h^{(i)}} \\ &= \frac{\partial L}{\partial h^{(i+1)}} \left[\sum_{h^{(i)} \to h^{(i+1)}} \frac{\partial h^{(i+1)}}{\partial h^{(i)}} \right]. \end{aligned} \tag{12}$$

Vielä on laskettava kaavan (12) summassa esiintyvä gradientti $\frac{\partial h^{(i+1)}}{\partial h^{(i)}}$. Olisi myös tässä vaiheessa toivottavaa, että gradientti voitaisiin esittää neuroverkon painojen, neuronien ja aktivaatiofunktion derivaatan avulla. Kuten aiemmin, neuroni voidaan kirjoittaa aktivaatiofunktion avulla seuraavasti:

$$h^{(i+1)} = \sigma\left(a^{(i+1)}\right) = \sigma\left(w_{(i+1)} h^{(i)}\right).$$

Tällöin suoralla laskulla gradientiksi saadaan

$$\frac{\partial h^{(i+1)}}{\partial h^{(i)}} = \frac{\partial h^{(i+1)}}{\partial a^{(i+1)}} \frac{\partial a^{(i+1)}}{\partial h^{(i)}} = \sigma'\left(a^{(i+1)}\right) w_{(i+1)}.$$

Sijoittamalla tämä takaisin kaavaan (12) saadaan

$$\frac{\partial L}{\partial h^{(i)}} = \frac{\partial L}{\partial h^{(i+1)}} \left[\sum_{h^{(i)} \to h^{(i+1)}} \sigma' \left(a^{(i+1)} \right) w_{(i+1)} \right], \tag{13}$$

jossa $a^{(i+1)}$ ja $w_{(i+1)}$ jälleen summataan siten, että summassa on vastaavat arvot kerrallaan vain tietyltä polulta. Edelleen, kun saatu lauseke sijoitetaan jälleen takaisin kaavaan (11), saadaan lopulta lauseke virhefunktion gradientille suhteessa mihin tahansa neuroverkon painoon

$$\frac{\partial L}{\partial w_{(i)}} = \frac{\partial L}{\partial h^{(i+1)}} \left[\sum_{h^{(i)} \to h^{(i+1)}} \sigma' \left(a^{(i+1)} \right) w_{(i+1)} \right] \sigma' \left(a^{(i)} \right) h^{(i-1)}. \tag{14}$$

Edellinen kaava pätee myös neuroverkossa, jossa vakioneuronit ovat käytössä. Tällöin asetetaan $h^{(i-1)} = 1$, sillä vakioneuroni on aina yksi.

Kaava (14) on näin ollen valmis käytettäväksi vastavirta-algoritmissa. Yksittäisen painon $w_{(i)}$ päivitykselle stokastisella gradienttimenetelmällä saadaan edellisten tulosten nojalla yksinkertainen päivityskaava

$$w_{(i)} \leftarrow w_{(i)} - \eta \frac{\partial L}{\partial w_{(i)}}.$$

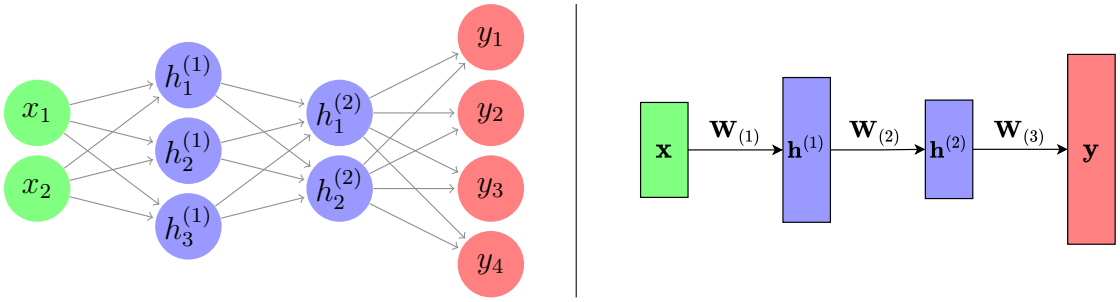
4.4.3 Peruutusvaihe: vektorikeskeinen neuroverkkorakenne

Edellä peruutusvaiheen gradienttien laskuja monimutkaistaa erityisesti neuroverkossa olevien polkujen suuri määrä, minkä takia jo yksittäisen painon suhteen laskettu gradientti vaatii runsaasti laskutyötä. Tarkastellaan seuraavaksi peruutusvaihetta siten, että painot ja neuronit käsitellään vektoreina ja matriiseina. Tällaista neuroverkon rakennetta kutsutaan vektorikeskeiseksi rakenteeksi. Vektorikeskeisessä rakenteessa neuroverkon peruutusvaiheen operaatioita voidaan laskea yhdellä kertaa useita, mikä nopeuttaa peruutusvaiheen laskuja. Kuviossa 7 on esitetty tavallinen neuroverkkograafi ja sitä vastaava vektorikeskeinen rakenne.

Vektorikeskeinen rakenne tuo mukanaan kaksi olennaista hyötyä verrattuna tavanomaiseen graafirakenteeseen: Ensiksi, vektorikeskeisessä neuroverkkorakenteessa esiintyy vain yksi polku, joten vastavirta-algoritmin peruutusvaihetta riittää tarkastella vain tällä yhdellä polulla. Toiseksi, nykyaikaiset tietokoneet, erityisesti varta vasten koneoppimisongelmia varten suunnitellut suorittimet (graafiikkasuorittimet (GPU) ja tensorisuorittimet (TPU)), pystyvät laskemaan todella tehokkaasti vektorien, matriisien ja muiden suurempiulotteisten tensorien laskutoimituksia. (Aggarwal 2020, 471.)

Kuten aiemmin, myös vektorikeskeisen neuroverkkorakenteen vastavirta-algoritmi käyttää gradientin laskennassa ketjusääntöä. Ketjusääntöä pitää kuitenkin laajentaa vielä edelleen sisältämään vektoriarvoisia muuttujia: Olkoon vektorit $\boldsymbol{u} \in \mathbb{R}^m$ ja $\boldsymbol{v} \in \mathbb{R}^n$ sekä kuvaukset $f : \mathbb{R}^n \to \mathbb{R}$ ja $g : \mathbb{R}^m \to \mathbb{R}^n$. Olkoon lisäksi $\boldsymbol{v} = g(\boldsymbol{u})$ ja $z = f(\boldsymbol{v}) = f(g(\boldsymbol{u}))$. Nyt gradientti u_i:n suhteen on

$$\frac{\partial z}{\partial u_i} = \frac{\partial z}{\partial \boldsymbol{v}} \frac{\partial \boldsymbol{v}}{\partial u_i} = \frac{\partial z}{\partial (v_1, \ldots, v_n)} \frac{\partial (v_1, \ldots, v_n)}{\partial u_i} = \sum_{j=1}^{n} \frac{\partial z}{\partial v_j} \frac{\partial v_j}{\partial u_i}.$$



Kuvio 7: *Sama neuroverkkorakenne esitettynä perinteisenä graafirakenteena (vasemmalla) sekä vektorikeskeisenä rakenteena (oikealla), jossa on vain yksi polku.*

Edellinen voidaan kirjoittaa myös vektorimuodossa seuraavasti

$$\nabla_{\boldsymbol{u}} z = \boldsymbol{J}^T \nabla_{\boldsymbol{v}} z,$$

jossa $\boldsymbol{J} = [\partial \boldsymbol{v} / \partial \boldsymbol{u}]$ on niin sanottu Jacobin matriisi, $\nabla_{\boldsymbol{u}} z$ on kuvauksen z gradientti vektorin $\boldsymbol{u}$ suhteen ja $\nabla_{\boldsymbol{v}} z$ on vastaavasti kuvauksen z gradientti vektorin $\boldsymbol{v}$ suhteen. Jacobin matriisi on $n \times m$ -matriisi, jossa $\boldsymbol{J}_{ij} = [\partial v_i / \partial u_j]$. Toisin sanoen, Jacobin matriisiin on koottu kaikki mahdolliset gradientit, jotka voidaan vektoreiden $\boldsymbol{u}$ ja $\boldsymbol{v}$ alkioiden välillä laskea. (Goodfellow et al. 2016, 203.)

Vektorikeskeisessä rakenteessa neuroverkon yhdellä kerroksella tapahtuvia operaatioita (neuronien arvojen $\boldsymbol{a}^{(i)}$ laskeminen ja aktivaatiofunktion käyttäminen: $\boldsymbol{h}^{(i)} = \sigma\left(\boldsymbol{a}^{(i)}\right)$) tarkastellaan erikseen. Etenemisvaiheessa operaatioiden järjestys on selvä: ensin lasketaan $\boldsymbol{a}^{(i)}$, minkä jälkeen lasketaan $\boldsymbol{h}^{(i)}$. Vastavirta-algoritmin peruutusvaiheessa operaatioiden järjestys on käänteinen, mikä pitää huomioida myös peruutusvaiheen gradienttien laskennassa. Ensimmäisenä operaationa tapahtuu siten gradientin laskeminen vektorin $\boldsymbol{h}^{(i)}$ suhteen. (Aggarwal 2018, 118–119.)

Tarkastellaan vastavirta-algoritmin peruutusvaihetta vektorikeskeisessä neuroverkkorakenteessa peräkkäisten kerrosten (i) ja $(i+1)$ välillä. Näiden kerrosten neuronit ovat vektoreissa $\boldsymbol{h}^{(i)}$ ja $\boldsymbol{h}^{(i+1)}$, joiden pituuden ovat vastaavasti p_i ja p_{i+1}. Olkoon $\boldsymbol{g}_{(i+1)}$ kerroksen $(i+1)$ jokaisen neuronin suhteen laskettu virhefunktion gradientit sisältävä p_{i+1}-pituinen vektori. Lisäksi, olkoon $\boldsymbol{J}_{(i+1)}$ kerrosten välinen $p_i \times p_{i+1}$ -kokoinen Jacobin matriisi. Toisin sanoen vektori $\boldsymbol{g}_{(i+1)}$ ja Jacobin matriisi ovat muotoa

$$\boldsymbol{g}_{(i+1)} = \begin{bmatrix} \frac{\partial L}{\partial h_1^{(i+1)}} \\ \frac{\partial L}{\partial h_2^{(i+1)}} \\ \vdots \\ \frac{\partial L}{\partial h_{p_{i+1}}^{(i+1)}} \end{bmatrix} \quad \text{ja} \quad \boldsymbol{J}_{(i+1)} = \frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}} = \begin{bmatrix} \frac{\partial h_1^{(i+1)}}{\partial h_1^{(i)}} & \frac{\partial h_2^{(i+1)}}{\partial h_1^{(i)}} & \cdots & \frac{\partial h_{p_{i+1}}^{(i+1)}}{\partial h_1^{(i)}} \\ \frac{\partial h_1^{(i+1)}}{\partial h_2^{(i)}} & \frac{\partial h_2^{(i+1)}}{\partial h_2^{(i)}} & \cdots & \frac{\partial h_{p_{i+1}}^{(i+1)}}{\partial h_2^{(i)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_1^{(i+1)}}{\partial h_{p_i}^{(i)}} & \frac{\partial h_2^{(i+1)}}{\partial h_{p_i}^{(i)}} & \cdots & \frac{\partial h_{p_{i+1}}^{(i+1)}}{\partial h_{p_i}^{(i)}} \end{bmatrix}.$$

Tällöin kerroksen (i) vastaavaksi p_i-pituiseksi gradienttivektoriksi $\boldsymbol{g}_{(i)}$ saadaan vektorimuotoisen ketjusäännön nojalla

$$\boldsymbol{g}_{(i)} = \boldsymbol{J}_{(i+1)}^T \boldsymbol{g}_{(i+1)}. \tag{15}$$

Vektorikeskeisessä neuroverkkorakenteessa peruutusvaihe on kaikessa yksinkertaisuudessaan gradienttivektoreiden kertomista Jacobin matriisilla, kunnes syöte saavutetaan (Goodfellow et al. 2016, 203). Kaavan (15) ja Jacobin matriisin käytön etu on, että sitä voidaan käyttää myös tavallista monimutkaisemmissa neuroverkoissa, joiden operaatioita ei voida toteuttaa ainoastaan matriisien ja vektorien laskutoimituksilla. Täysin kytketyn neuroverkon tapauksessa kaikki operaatiot ovat aina toteutettavissa tavallisin lineaarialgebran keinoin, jolloin kaavan (15) sijaan voidaan käyttää yksinkertaisempia menetelmiä.

Osoittautuu, että gradienttivektori $\boldsymbol{g}_{(i)}$ voidaan kirjoittaa aktivaatiofunktion derivaatan avulla

$$\boldsymbol{g}_{(i)} = \boldsymbol{g}_{(i+1)} \odot \sigma' \left(\boldsymbol{a}^{(i)} \right).$$

Operaattori $\odot$ tarkoittaa kertolaskua, jossa vain vektorien kohdakkain olevat alkiot kerrotaan keskenään (kaavan (4) periaatteella). (Aggarwal 2018, 120; Aggarwal 2020, 471–473.)

Nyt voidaan laskea peruutusvaiheen oleellisin tulos, eli $p_{i-1} \times p_i$ -kokoinen matriisi $\nabla_{\boldsymbol{W}_i}$, joka sisältää jokaisen painomatriisin $\boldsymbol{W}_i$ painojen suhteen lasketut virhefunktion gradientit:

$$\nabla_{\boldsymbol{W}_i} = \boldsymbol{g}_{(i)} \left(\boldsymbol{h}^{(i-1)} \right)^T. \tag{16}$$

Jälleen, jos neuroverkossa on käytössä vakioneuronit, niin tällöin $\boldsymbol{h}^{(i-1)} = \boldsymbol{1}$, jolloin vakioneuronien suhteen laskettu p_i-pituinen gradienttivektori on

$$\nabla_{\boldsymbol{b}_i} = \boldsymbol{g}_{(i)}. \tag{17}$$

(Goodfellow et al. 2016, 209; Aggarwal 2020, 471–473.) Kaavoilla (16) ja (17) laskettuja gradientteja voidaan käyttää suoraan vastaavien parametrien päivityksessä, kun käytetään esimerkiksi stokastista gradienttimenetelmää.

Lopuksi neuroverkossa valmistaudutaan siirtymään edelliselle kerrokselle $(i-1)$, joten lasketaan p_{i-1}-pituinen gradienttivektori $\boldsymbol{g}_{(i-1)}$ neuronien arvojen $\boldsymbol{h}^{(i-1)}$ suhteen:

$$\boldsymbol{g}_{(i-1)} = \boldsymbol{W}_i \, \boldsymbol{g}_{(i)}.$$

Painomatriisi $\boldsymbol{W}_i$ on transpoosi etenemisvaiheessa käytetystä matriisista $\boldsymbol{W}_i^T$. (Aggarwal 2018, 120.)

Vastavirta-algoritmin vaiheet voidaan esittää lyhyesti myös algoritmimuodossa. Algoritmit 1 ja 2 perustuvat teoksiin Goodfellow et al. (2016) s. 208–209 ja Aggarwal (2018) s. 120. Algoritmissa 2 laskettuja painojen gradientteja $\nabla_{\boldsymbol{b}_i}$ ja $\nabla_{\boldsymbol{W}_i}$ voi käyttää välittömästi (jopa kesken algoritmin suorituksen) stokastisessa gradienttimenetelmässä.

Algoritmi 1 Etenemisvaihe

Olkoon: neuroverkon syvyys k.
Olkoon: painomatriisit $\boldsymbol{W}_i$, $i = 1, \ldots, k$.
Olkoon: vakioneuronit $\boldsymbol{b}_i$, $i = 1, \ldots, k$.
Olkoon: syöte $(Y, \boldsymbol{x})$.
Olkoon: ulostulo $\boldsymbol{y}$.

Alustetaan syötekerros: $\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $i = 1, 2, \ldots, k-1, k$ **do**
    $\boldsymbol{a}^{(i)} = \boldsymbol{W}_i^T \boldsymbol{h}^{(i-1)} + \boldsymbol{b}_i$
    $\boldsymbol{h}^{(i)} = \sigma\left(\boldsymbol{a}^{(i)}\right)$
end for
Muodostetaan $\boldsymbol{y}$ softmax-todennäköisyyksien $\boldsymbol{h}^{(k)}$:n perusteella.
Lasketaan virhe: $L(\boldsymbol{\theta})$

Algoritmi 2 Peruutusvaihe

Olkoon: Etenemisvaihe (algoritmi 1) suoritettu.

Alustetaan $\boldsymbol{g}$: $\boldsymbol{g} = \nabla_{\boldsymbol{h}^{(k)}} L$
for $i = k, k-1, \ldots, 2, 1$ **do**
    $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(i)}} L = \boldsymbol{g} \odot \sigma'\left(\boldsymbol{a}^{(i)}\right)$
    $\nabla_{\boldsymbol{b}_i} L = \boldsymbol{g}$
    $\nabla_{\boldsymbol{W}_i} L = \boldsymbol{g}\left(\boldsymbol{h}^{(i-1)}\right)^T$
    $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(i-1)}} L = \boldsymbol{W}_i \boldsymbol{g}$
end for

4.5 Neuroverkon opetukseen liittyviä ongelmia

Kuten minkä tahansa tilastollisen mallin kanssa, myös neuroverkkojen parametrien estimointiin pitää kiinnittää erityistä huomiota, jotta neuroverkkomallista saadaan toimiva. Tässä luvussa esitellään kolme tyypillistä ongelmaa, jotka neuroverkkojen opetuksen ja mallin valinnan aikana pitää ratkaista.

4.5.1 Ali- ja ylioppiminen

Ylioppimisella (*overfitting*) tarkoitetaan sellaista tilannetta, jossa tilastollinen malli toimii liian tehokkaasti ja siten oppii opetusaineistosta myös sellaisia piirteitä, jotka ovat peräisin muualta kuin tuntemattomasta funktiosta f. Malli voi esimerkiksi oppia opetusaineistossa olevia, täysin satunnaisvaihtelusta peräisin olevia piirteitä. Jos näin on tapahtunut, niin vastaavanlaisia satunnaisia piirteitä ei välttämättä tunnisteta testiaineistosta lainkaan. Luokitin toimii siten huonosti sekä testiaineistolle että myös yleisesti. (James et al. 2021, 32.)

Etenkin, jos laskentakapasiteettia on käytössä paljon, niin ylioppiminen voi tapahtua neuroverkkoa laadittaessa melko helposti ja huomaamatta. Koska neuroverkoissa on jo

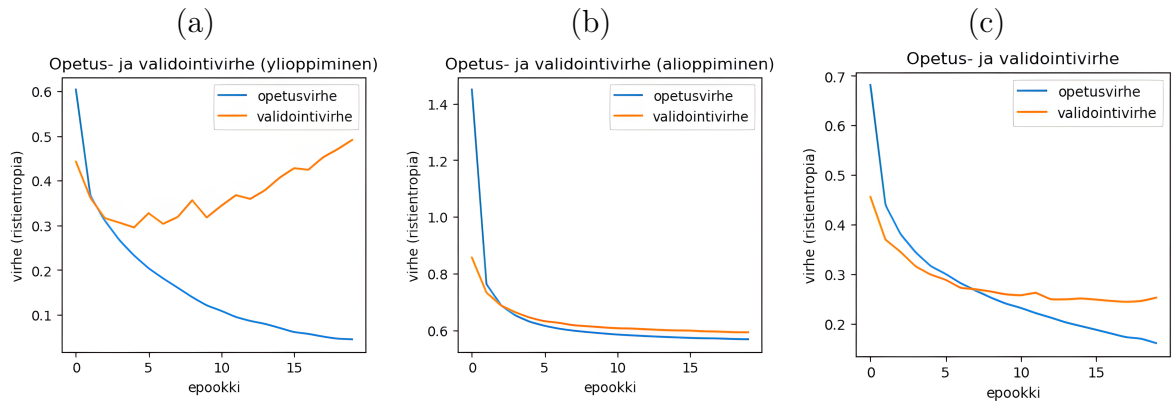
luonnostaan paljon parametreja (jopa useita miljoonia), niin ajatus siitä, että mahdollisimman monimutkainen neuroverkkorakenne ratkaisee minkä tahansa luokitteluongelman täydellisesti, on houkutteleva. Siispä onkin tärkeää tarkastella mahdollista ylioppimista validointiaineiston avulla jo neuroverkon opetuksen aikana eikä vasta opetuksen jälkeen. Perusidea on laskea neuroverkon opetus- ja validointivirheet jokaisen epookin jälkeen käyttäen sekä opetus- että validointiaineistoja. Jos malli on ylioppinut, niin mallin validointivirhe alkaa kasvamaan jossain vaiheessa epookkien lukumäärän kasvaessa. Opetusvirhe sen sijaan ainoastaan pienenee. Jos ylioppimista ei ole, niin tällöin hyvin opetetun mallin validointivirhe tasaantuu eikä enää pienene, vaikka epookkeja laskisi lisää.

Riippumatta siitä, mitä tilastollista mallia käytetään, mallin ylioppimisen estäminen tunnetaan termillä regularisaatio (*regularization*). Neuroverkkojen tapauksessa ilmeinen regularisaatiomenetelmä on yksinkertaistaa neuroverkon rakennetta esimerkiksi neuroverkon syvyyttä tai neuronien lukumäärää pienentämällä. Tällöin kuitenkin pitää pitää mielessä, että liian yksinkertainen neuroverkko ei välttämättä pysty ratkaisemaan monimutkaista luokitteluongelmaa. Neuroverkko voi siten olla myös alioppinut (*underfitting*). Parhaimmassa tilanteessa neuroverkko ei ole yli- eikä alioppinut.

Kuviossa 8 on esimerkki testi- ja validointivirheiden käyttäytymistä, kun neuroverkko on joko ylioppinut tai alioppinut tai kun se on opetettu riittävästi. Kaikissa tilanteissa neuroverkko luokittelee harmaasävykuvia kymmeneen luokkaan, kun käytetään samoja opetus-, testi- ja validointiaineistoja. Epookkeja lasketaan kaikissa tilanteissa 20, ja ainoastaan neuroverkon rakennetta on muutettu. Kuvaajassa (a) validointivirhe ensin pienenee, mutta kääntyy kasvuun epookkien lukmäärän kasvaessa, mikä on merkki ylioppimisesta. Kuvaajan (a) neuroverkkorakenne on monimutkaisin (yhteensä noin 3.7 miljoonaa parametria). Kuvaajassa (b) validointivirhe pienenee samaan tahtiin opetusvirheen kanssa, ja virhefunktion arvo on suhteellisen suuri epookkien lukumäärästä huolimatta. Tämä on merkki neuroverkon alioppimisesta, joten neuroverkon rakennetta voisi monimutkaistaa pienemmän virheen (ja siten paremman luokittelijan) saamiseksi. Tämä neuroverkko on yksinkertaisin (yhteensä alle 500 parametria). Kuvaajassa (c) ei ole havaittavissa ali- eikä ylioppimista, sillä validointivirhe pienenee epookkien lukumäärän kasvaessa, mutta tasaantuu noin 15 epookin jälkeen. Siispä, kuvaajan (c) neuroverkko on sopivin (noin 122 000 parametria).

Toinen neuroverkkojen tehokas ja suosittu regularisaatiomenetelmä on niin sanottu neuronikato (*dropout*). Neuronikato on yksinkertainen ja tehokas menetelmä, jonka keskeinen idea on satunnaisesti pudottaa yksittäisiä neuroneita neuroverkon syöte- tai piilokerroksilta ennalta määrätyillä todennäköisyyksillä. Jokaisella neuronilla, joissa neuronikato on otettu käyttöön, on mahdollisuus tulla pudotetuksi muista neuroneista riippumattomasti. Käytännössä neuronin pudotuksessa neuronille asetetaan arvo nolla, jolloin sekä neuroniin vaikuttavat että neuronista lähtevät painot mitätöityvät. Samalla kaikki polut, jotka kulkevat pudotetun neuronin kautta, katkeavat. Seurauksena vain neuronikadosta selvinneiden neuronien kautta kulkevat ehjät polut tarvitsee huomioida neuroverkon opetuksessa. (Aggarwal 2018, 189–190.)

Neuronikatoa voi katsoa myös toisesta näkökulmasta: Jos alkuperäisessä neurover-



Kuvio 8: *Esimerkit neuroverkon testi- ja validointivirheistä, kun neuroverkko ylioppinut, alioppinut tai riittävästi oppinut.*

kossa ilman neuronikadon käyttöä on yhteensä N neuronia, niin tällöin on olemassa 2^N kappaletta sellaisia neuroverkkoja, jotka koostuvat joistain alkuperäisen neuroverkon neuroneista. Tällöin, kun neuroverkon painoja päivitetään SGD-menetelmällä, näistä 2^N eri neuroverkosta arvotaan jokin neuroverkkorakenne, jonka painot päivitetään. Arvonta tehdään siten, että jokaista SGD:n erää kohti arvotaan uusi rakenne. Lopulta arvottujen rakenteiden painot yhdistetään, jolloin saadaan neuronikadolla opetettu neuroverkko. Näin saadaan tehokkaasti ehkäistyä ylisovittumista. On kuitenkin huomattava, että neuroverkon opettamiseen käytetty aika kasvaa 2–3-kertaiseksi verrattuna neuroverkkoon, jossa ei käytetä neuronikatoa. (Aggarwal 2018, 189; Srivastava et al. 2014.)

4.5.2 Katoavan ja räjähtävän gradientin ongelma sekä sammuneet neuronit

Luvun 3.3.1 aktivaatiofunktioiden yhteydessä jo mainittu katoavan gradientin ongelma (*vanishing gradient problem*) ja sille hyvin samankaltainen räjähtävän gradientin ongelma (*exploding gradient problem*) ovat kaksi yleistä ongelmaa, jotka tapahtuessaan vaikeuttavat neuroverkon opetusta.

Vastavirta-algoritmin peruutusvaiheen yhteydessä näytettiin, että virhefunktion gradienttiin kerroksella (i) vaikuttaa aktivaatiofunktion derivaatta $\sigma'(\cdot)$ ja vastaava paino kerroksella $(i+1)$ (kaavat (13) ja (14)). Tällöin tilanteessa, jossa kyseinen derivaatta on hyvin lähellä nollaa, gradienttien laskeminen rekursiivisesti ei etene juuri lainkaan, vaikka iteraatioiden lukumäärää kasvattaisi. Virhefunktion minimiä ei siten saavuteta ja neuroverkkoluokitin toimii kehnosti. Räjähtävän gradientin ongelmassa tilanne on päinvastainen. Tällöin aktivaatiofunktion derivaatta on suhteellisen suuri, minkä seurauksena gradientti voi muuttua liikaa ja ohittaa virhefunktion minimin. Tällöinkään minimiä ei saavuteta.

Katoavan ja räjähtävän gradientin ongelmat voi kohdata myös säätämällä SGD:n oppimisnopeuden η liian pieneksi tai suureksi. Kuitenkin, toisin kuin oppimisnopeus, joka on käyttäjän päätettävissä oleva parametri, vastavirta-algoritmin aikana gradientteihin liittyvää epästabiiliutta esiintyy luonnollisesti jokaisessa neuroverkossa. Siihen, kuinka suureksi ongelmaksi gradienttien käyttäytyminen tulee, voi käyttäjä vaikuttaa esimerkik-

si sopivaa aktivaatiofunktiota käyttämällä. Optimitilanteessa aktivaatiofunktion derivaatan ja derivaattaa vastaavan kerroksen painojen yhteinen vaikutus vastavirta-algoritmin peruutusvaiheessa tapahtuvaan gradientin rekursiokaavaan olisi tasan yksi. Tällöin, kun gradientteja lasketaan rekursiivisesti, eri kerrosten gradientit pysyvät kaikki samassa suuruusluokassa. Rekursiosta johtuen, gradienttien pieneneminen tai suureneminen riippuu eksponentiaalisesti neuroverkon kerrosten lukumäärästä. Jos yhteinen vaikutus on alle yhden, niin tällöin gradientit alkavat pienentyä (katoavan gradietin ongelma, esim. $0.9^5 = 0.59 < 1$). Jos taas yhteinen vaikutus on suurempaa kuin yksi, niin gradientit alkavat kasvaa (räjähtävän gradientin ongelma, esim. $1.1^5 = 1.61 > 1$). (Aggarwal 2018, 130.)

Luvussa 3.3.1 esitetyt perinteiset aktivaatiofunktiot sigmoidi ja hyperbolinen tangentti kärsivät katoavan gradientin ongelmasta. Ongelma johtuu näiden funktioiden tavasta kuvata reaaliluku joko välille $(0, 1)$ (sigmoidi) tai välille $(-1, 1)$ (tanh). Funktioiden arvojen suurin vaihtelu tapahtuu kohdan $x = 0$ läheisyydessä, ja suurilla positiivisilla tai negatiivisilla arvoilla funktioiden arvot eivät vaihtele enää juuri lainkaan. Toisin sanoen, sigmoidifunktion ja hyperbolisen tangentin derivaatat ovat itseisarvoltaan suurilla arvoilla mitättömän pieniä, mikä näkyy opetuksen aikana katoavan gradientin ongelmana. Voidaan lisäksi helposti osoittaa, että hyperbolisen tangentin derivaattafunktion suurin mahdollinen arvo on 1, ja sigmoidifunktion derivaatan suurin mahdollinen arvo on vain 0.25. Näin ollen, jo näiden aktivaatiofunktioiden perusominaisuuksien nojalla on odotettavissa, että derivaatta on alle yhden, ja katoavan gradientin ongelma voi esiintyä. Neuroverkon opettaminen voi siis hidastua, mutta opetus voi siitä huolimatta onnistua, jos vain aikaa ja laskentatehoa on käytettävissä tarpeeksi.

ReLU-aktivaatiofunktio sen sijaan ennaltaehkäisee erittäin hyvin sekä katoavan että räjähtävän gradientin ongelmia. ReLU on kuvaus

$$\sigma(x) = \begin{cases} x, & \text{kun } x \geq 0 \\ 0, & \text{kun } x < 0 \end{cases},$$

josta nähdään, että ReLU:n derivaatta on aina vakio 1, kun $x \geq 0$ ja nollaa muualla. Siispä, kun derivaatta on vakio, se ei tällöin voi kasvaa mielettömän suureksi tai kutistua mitättömän pieneksi. Tällöin sekä räjähtävän että katoavan gradientin ongelmilta vältytään. Ongelmia kuitenkin seuraa negatiivisilla ReLU:n syötteen arvoilla. Alueella $x < 0$ derivaatta on nolla, joten selkeästi katoavan gradientin ongelma esiintyy. Toinen ongelma syntyy, kun myös neuroni saa arvon nolla. Tällöin gradientti ei pysty päivittymään lainkaan riippumatta kyseiseen neuroniin vaikuttavien painojen arvoista. Kun tällainen tilanne saavutetaan, neuroni pysyy aina nollana koko neuroverkon opetuksen ajan. Sanotaan että neuroni on sammunut, ja neuroverkko on tällöin ”vaurioitunut” tai kärsinyt biologisin termein sanottuna ”aivovauriosta”. (Aggarwal 2018, 133.)

Sammuneita neuroneita ja neuroverkon vaurioita ei pidä automaattisesti pitää ongelmana neuroverkon toiminakyvyn kannalta. Jos neuroverkon opettamisen aikana muutamia neuroneita sammuu neuroverkon eri osissa, niin tätä voi pitää osana neuroverkon

opetusprosessia: Jos tiettyä polkua pitkin kulkeva signaali syötteeltä ulostulolle nollautuu sammuneen neuronin takia, on tämä samaan tapaan informaatiota kuin ehjiä polkuja pitkin kulkevat signaalitkin. Näin neuroverkon vaurioita voi pitää myös eräänlaisena regularisaationa, mutta sillä erolla, että neuroverkon käyttäjä ei suoraan voi vaikuttaa sammuneista neuroneista johtuvan regularisoivan vaikutuksen suuruuteen. On myös huomattava, että liian suuri määrä sammuneita neuroneita voi huonontaa neuroverkon toimintaa niin paljon, ettei neuroverkko enää toimi ollenkaan. (Aggarwal 2018, 134.)

ReLU:n ja muiden sellaisten aktivaatiofunktioiden kanssa, joiden derivaatta joillain alueilla on nollaa tai hyvin lähellä sitä, neuroverkon vaurioiden liiallista esiintymistä voidaan välttää alustamalla neuroverkon opetettavat parametrit sopivasti. Esimerkiksi ReLU:n tapauksessa parametrien on toivottavaa olla positiivisia, jotta neuronit ovat aktiivisia heti opetuksen alussa. Yleisesti, neuroverkon painot voidaan alustaa esimerkiksi standardinormaali- tai tasajakaumasta arpomalla. Positiivisuuden voi varmistaa lisäämällä jokaiseen arvottuun neuronin arvoon jonkin pienen positiivisen vakion.

Parametrien oikeanlainen alustaminen ei kuitenkaan takaa sitä, etteivätkö parametrit jossain vaiheessa opetusta siirtyisi aktivaatiofunktion näkökulmasta epäedulliselle alueelle. Mitä syvempi neuroverkko on, sitä enemmän neuroverkossa on muita neuroverkon sellaisia neuroneita, jotka vaikuttavat muilla kerroksilla oleviin neuroneihin. Jos neuronien arvot vaihtelevat paljon aiemmilla kerroksilla, sama arvojen vaihtelu näkyy neuroverkon viimeisillä kerroksilla vielä suurempana. Tähän ongelmaan eräs tehokas ratkaisu on normalisoida jokainen SGD:n erä jokaisella tai joillain piilokerroksilla, jolloin eri kerrosten välinen neuronien arvojen vaihtelu on pienempää. Menetelmää kutsutaan eränormalisoinniksi (*batch normalization*, Ioffe et al. (2015)).

Eränormalisoinnin jälkeen neuronit on keskistetty ja skaalattu. Se on tehokas tapa vähentää neuroverkon vaurioita, mutta samalla myös katoavan ja räjähtävän gradientin ongelmaa, sillä erän skaalaus ja keskistäminen vaikuttaa myös gradienttien suuruuteen. Gradientti ei näin ollen voi räjähtää mielettömän suureksi, sillä normalisointi palauttaa arvot takaisin sopivaan suuruusluokkaan (Goodfellow et al. 2016, 315.) Kun lisäksi käytetään sopivaa aktivaatiofunktiota (esim. ReLU), niin myös katoavan gradientin ongelmasta on huolehdittu.

Neuroverkon peruutusvaiheessa eriä on useita ja eränormalisoinnissa yksittäinen havainto voi kuulua mihin tahansa normalisoitavaan erään. Jokaisella havainnolla on siten mahdollisuus vaikuttaa neuronien arvoihin hieman eri tavalla riippuen siitä, mihin erään havainto kuuluu. Tällainen vaikutus toimii ylimääräisenä kohinan lähteenä neuroverkon parametrien päivityksen aikana. Ylimääräisellä kohinalla on puolestaan epäsuora regularisoiva vaikutus neuroverkossa. (Aggarwal 2018, 156.) He et al. (2016) osoittivatkin, että aiemmin esitetyllä neuronikadolla ei ole regularisoivaa vaikutusta, jos eränormalisointia on jo käytetty neuroverkon kerroksilla.

5 Konvoluutioneuroverkot

Tyypillinen luvussa 3 esitetty, vain täysin kytketyistä kerroksista koostuva neuroverkkorakenne, toimii hyvin sellaisessa tilanteessa, jossa aineisto on tavallinen kaksiulotteinen datamatriisi. Tällöin toiminta syötekerroksella on ilmeistä, sillä jokaiselle muuttujalle voidaan varata oma neuroni (tai aineiston esikäsittelystä riippuen useampi neuroni). Toiseksi, aineiston muuttujien välillä ei tarvitse olettaa olevan minkäänlaisia riippuvuuksia, jotka pitäisi huomioida neuroverkossa. Toisin sanoen, jokaisella muuttujalla on mahdollisuus vaikuttaa (tai olla vaikuttamatta) yhtä paljon kaikkiin muihin neuroneihin sekä lopputulokseen. Näin ei aina kuitenkaan ole. Esimerkiksi kuvien kaltaisten spatiaalisten rakenteiden luokittelussa, kuvan yhden tietyn kuvapisteen ympärillä olevat muut kuvapisteet vaikuttavat enemmän siihen, mitä kuvan tietty alue esittää.

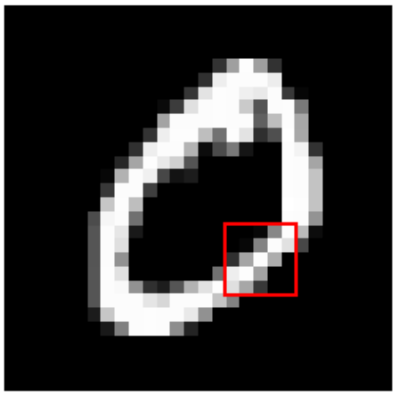
Konvoluutioneuroverkko (*convolutional neural network*, CNN) on syväoppimismenetelmä, joka on suunniteltu käytettäväksi erityisesti kuvien kaltaisten kaksiulotteisten spatiaalisten rakenteiden luokittelussa (Aggarwal 2018, 315). Konvoluutioneuroverkon tarkoituksena on tarkastella syötteen eri alueita (ja alueiden ympäristöjä) ja huomioida syötteen spatiaalinen rakenne. CNN oppii syötteestä erilaisia rakenteita, kuten esineiden reunoja, kulmia ja eri muotoja, joiden perusteella luokittelu voidaan lopuksi tehdä. (Goodfellow et al. 2016, 330.)

Tässä luvussa määritellään ensin tämän tutkielman kannalta riittävällä tasolla tensori sekä tarkastellaan kaksi- ja yksiulotteisten rakenteiden tensoriesityksiä. Kaksiulotteisista rakenteista käsitellään sekä harmaasävy- että värikuvia, sillä niiden luokittelu kuuluu yleisimpien konvoluutioneuroverkon sovelluskohteiden joukkoon. Yksiulotteisista rakenteista tarkastelu rajataan DNA-sekvensseihin, mutta esitys yleistyy helposti myös muihin sekvenssiaineistoihin. Tämän jälkeen esitetään konvoluutioneuroverkolle ominaiset matemaattiset operaatiot konvoluutio ja tiivistys. Lopuksi tarkastellaan kokonaisen konvoluutioneuroverkon rakennetta.

5.1 Harmaasävy- ja värikuvien tensoriesitykset

Yksinkertainen ja tyypillinen CNN:n sovellus on harmaasävykuvien luokittelu. Tällöin syötteenä on kaksiulotteinen harmaasävykuva, jossa jokainen kuvapiste saa kokonaislukuarvon väliltä $[0, 255]$. Luku 0 vastaa täysin mustaa kuvapistettä, luku 255 puolestaan täysin valkoista kuvapistettä ja muut arvot vastaavat harmaan eri sävyjä mustan ja valkoisen väliltä. Tällainen kuva voidaan esittää kaksiulotteisena matriisina, mitä on havainnollistettu kuviossa 9. Kuvien esittäminen matriiseina toimii harmaasävykuvien tapauksessa ymmärrettävästi, sillä yhden kuvapisteen kaikki mahdolliset eri värisävyt voidaan esittää yhtenä kokonaislukuna väliltä $[0, 255]$.

Värikuvat koostuvat nekin harmaasävykuvien tapaan kuvapisteistä, mutta värikuvassa yksi värillinen kuvapiste muodostuu useammasta kuin yhdestä värisävystä. Kuvien yhteydessä näitä värisävyjä kutsutaan kuvan kanaviksi (*channel*). Tyypillisesti värikuvan kanavien lukumäärä on kolme ja kanavat ovat punainen, vihreä ja sininen. Näiden



$$\begin{bmatrix} 0 & 0 & 7 & 135 & 253 \\ 0 & 7 & 131 & 252 & 225 \\ 48 & 165 & 252 & 173 & 0 \\ 238 & 253 & 162 & 0 & 0 \\ 223 & 167 & 56 & 0 & 0 \end{bmatrix}$$

Kuvio 9: *Käsinkirjoitettuja numeroita sisältävän MNIST-aineiston (LeCun et al. 1998) numeroa nolla esittävä* 28×28 *-kokoinen harmaasävykuva. Neliöllä rajattua* 5×5 *-kokoista aluetta vastaavassa matriisissa on kuvassa esiintyvien kuvapisteiden lukuarvot.*

värien englanninkielisistä nimistä myös muodostuu värikuviin viittaava lyhenne *RGB*. Myös muunlaisia värikanavajakoja käytetään eri sovelluksissa (esimerkiksi monet tulostimet käyttävät neljää värikanavaa värikuvien tulostamisessa: syaani, magneta, musta ja keltainen), mutta tässä tutkielmassa keskitytään vain RGB-värikuviin. RGB-kuvan jokainen kanava voidaan nyt esittää kaksiulotteisena matriisina vastaavasti kuin harmaasävykuvan yksi ainoa kanava. Matriisin arvo kuvaa kyseisen kanavan värin intensiteettiä: arvo 0 vastaa täysin mustaa kuvapistettä ja arvo 255 puolestaan kyseisen värin suurinta mahdollista intensiteettiä. Kun kanavia tarkastellaan yhdessä, syntyy lopullinen värikuva. Värikuvan eri kanavia on havainnollistettu kuviossa 10.

Kuvio 10: *Värikuva auringonkukasta ja sen hajotelma punaiseksi, vihreäksi ja siniseksi kanavaksi.*

Kokonainen RGB-kuva muodostuu siis kolmesta kaksiulotteisesta matriisista. Tämä rakenne on moniulotteinen taulukko eli tensori (*tensor*). Tensori on tavallisen kaksiulotteisen matriisin yleistys kolmeen tai useampaan ulottuvuuteen (Bi et al. 2021). Edellä esitetty kolmesta kaksiulotteisesta matriisista muodostuva RGB-värikuva on esimerkki eräästä kolmiulotteisesta tensorista. Tensoreita merkitään (tässä tutkielmassa) siten, että tensorista esitetään pelkästään sen dimensio. Dimensio puolestaan esitetään sulkumerkinnällä: Esimerkiksi $(28, 28, 3)$ on merkintä kolmiulotteiselle tensorille, jonka dimensiot ovat 28, 28 ja 3. Tällainen tensori esittää esimerkiksi resoluutioltaan 28×28 -kokoista värikuvaa. Sulkujen sisällä on järjestyksessä ensin kuvan resoluutio (kuvan

leveys 28 ja korkeus 28), ja lopuksi kuvan kanavien lukumäärä (3). Vastaavan harmaasävykuvan tensori on kaksiulotteinen tensori $(28, 28)$ (eli $(28, 28, 1)$), joka on tavallinen matriisi.

Luvussa 4.3 stokastisen gradienttimenetelmän yhteydessä määritetty eräkoko näkyy myös tensoriesityksissä siten, että valitun eräkoon mukainen määrä syötteen havaintoja "niputetaan" yhteen tensoriin. Tällöin tensorin dimensio kasvaa yhdellä. Esimerkiksi eräkoon ollessa 64, neliulotteinen tensori $(64, 28, 28, 3)$ sisältää nyt 64 kappaletta 28×28 -kokoisia värikuvia.

5.2 DNA-sekvenssien tensoriesitys

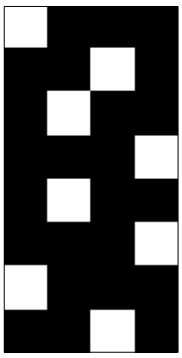
DNA-sekvenssit (olettaen, että jokaisen nukleotidin emäsosa on täysin havaittu ja puuttuvaa tietoa ei ole lainkaan) ovat ennen esikäsittelyä merkkijonoja, jotka koostuvat kirjaimista `A`, `T`, `G` ja `C`. Merkkijonoa ei voi käyttää sellaisenaan käytännössä minkään tilastomenetelmän syötteenä, vaan sekvenssi on ensin esikäsiteltävä numeeriseen muotoon. Tässä alaluvussa DNA-sekvenssien esikäsittelyyn esitetään kaksi tapaa. Ensimmäisessä tavassa DNA-sekvenssin edellä mainitut neljä emäsvaihtoa koodataan numeeriseen muotoon, minkä jälkeen koko DNA-sekvenssi voidaan muuttaa numeeriseksi koodauksen perusteella. Toisessa tavassa DNA-sekvenssille johdetaan numeerinen tunnuslukuvektori, jonka tavoite on kuvata alkuperäistä sekvenssiä mahdollisimman hyvin. Tämä tunnuslukuvektori edustaa alkuperäistä DNA-sekvenssiä analyyseissa.

Emäksien uudelleenkoodaukseen perustuva ensimmäinen tapa perustuu indikaattorimuuttujiin, jotka kuvaavat, onko tietyssä kohdassa DNA-sekvenssiä tietty emäs vai ei. Koska emäsvaihtoehtoja on neljä kappaletta, tarvitaan neljä indikaattorimuuttujaa. Toisin sanoen, jokainen emäs koodataan sellaiseksi vektoriksi, jonka pituus on neljä, ja jossa on luku yksi kyseessä olevan emäksen kohdalla ja muuten nollaa. Tällaista vektoria kutsutaan indikaattorivektoriksi (*one-hot encoding, one-hot vector*). Voidaan esimerkiksi sopia, että adeniini (`A`) koodataan indikaattorivektoriksi $(1, 0, 0, 0)$, tymiini (`T`) vektoriksi $(0, 1, 0, 0)$, guaniini (`G`) vektoriksi $(0, 0, 1, 0)$ ja sytosiini (`C`) vektoriksi $(0, 0, 0, 1)$. Kun koodaus on valittu, sitä voidaan käyttää jokaiseen DNA-sekvenssin emäkseen. Indikaattorivektorit voidaan lopuksi yhdistää, jolloin saadaan $bp \times 4$ -kokoinen matriisi, jossa bp on DNA-sekvenssin pituus. Saatu matriisi vastaa mustavalkokuvaa, jossa nolla vastaa täysin mustaa kuvapistettä ja ykkönen täysin valkoista kuvapistettä. Kuviossa 11 on esimerkki erästä DNA-sekvenssiä vastaavista indikaattorivektoreista sekä indikaattorivektorien muunnoksesta mustavalkokuvaksi.

Indikaattorivektoriesityksen yksi oleellinen ongelma konvoluutioneuroverkossa liittyy DNA-sekvenssien pituuksiin. Jos DNA-sekvenssit ovat eri pituisia keskenään, indikaattorivektoreista muodostetut matriisit ovat myös eri kokoisia. Tämä on ongelma, sillä konvoluutioneuroverkon syötteiden on oltava kaikkien samankokoisia. Toisin sanoen, CNN:lle syötteenä annettujen tensorien on oltava dimensioiltaan samoja. Eräs ongelman ratkaisu on DNA-sekvenssien keinotekoinen pidentäminen siten, että kaikista DNA-sekvensseistä tehdään yhtä pitkiä. Tässä tavassa DNA-sekvensin indikaattorivektoriesityksen perään

A	1	0	0	0
G	0	0	1	0
T	0	1	0	0
C	0	0	0	1
T	0	1	0	0
C	0	0	0	1
A	1	0	0	0
G	0	0	1	0

$\Rightarrow$



Kuvio 11: *Keksittyä DNA-sekvenssiä* `AGTCTCAG` *vastaava indikaattorivektoreihin perustuva matriisi ja sitä vastaava* 8×4 *-kokoinen mustavalkokuva.*

lisätään $(0, 0, 0, 0)$-vektoreita, kunnes pituudet ovat samoja koko aineistossa. Keinotekoinen pidentäminen kasvattaa kuitenkin samalla myös syötteen kokoa, joten neuroverkon opettamiseen tarvittava aika kasvaa.

Toisessa, DNA-sekvenssien tunnuslukuvektoreihin perustuvassa vaihtoehdossa konvoluutioneuroverkkoa voidaan soveltaa suoraan, kunhan sopivat tunnuslukuvektorit on ensin johdettu. Tällaisia tunnuslukuvektoreita kutsutaan DNA:n perimätunnuksiksi (*genome signature*). Eräs tapa johtaa perimätunnus on laskea DNA-sekvenssin k-mer-frekvenssit (*k-mer counting, k-mer frequences*). (Gatherer 2007.) Termillä k-mer tarkoitetaan $k = 1, 2, \ldots$ pituisia DNA-sekvenssin peräkkäisten emästen muodostamia osajonoja. Jälleen, koska emäsvaihtoehtoja on neljä kappaletta, niin erilaisia k-pituisia osajonovaihtoehtoja on 4^k kappaletta. Jonot muodostetaan siten, että ensimmäinen jono alkaa DNA-sekvenssin ensimmäisestä emäksestä ja jonoon otetaan mukaan k emästä (ensimmäinen emäs mukaan lukien) järjestyksessä. Toinen jono aloitetaan DNA-sekvenssin toisesta emäksestä ja mukaan otetaan jälleen k emästä. Näin jatketaan, kunnes muodostetun osajonon viimeinen emäs on koko DNA-sekvenssin viimeinen emäs. Tällöin uusia osajonoja ei enää voida muodostaa. Jos DNA-sekvenssin pituus on bp, niin osajonoja voidaan muodostaa $bp - k + 1$ kappaletta. Tyypillisessä tilanteessa $bp \gg k$, joten osajonojen lukumäärä on lähellä koko DNA-sekvenssin pituutta. Osajonojen muodostumista on havainnollistettu kuvion 12 esimerkissä.

Kun osajonot on muodostettu, voidaan eri osajonojen frekvenssit laskea. Frekvenssit kootaan vektoriin, jonka pituus on 4^k, seuraavasti: Jokainen vektorin alkio vastaa yhtä eri k-mer-osajonovaihtoehtoa, ja kyseisen alkion arvo kertoo, kuinka monta kertaa kyseinen osajono esiintyy DNA-sekvenssin k-mer-osajonojen joukossa. Jos osajonoa ei esiinny lainkaan, niin sen frekvenssi on nolla. Vektorin pituus on tällöin aina muotoa 4^k. Syntynyt vektori on DNA-sekvenssin tunnuslukuvektori eli perimätunniste. Esimerkiksi kuvion 12 tilanteessa $k = 2$ kaikkien mahdollisten osajonovaihtoehtojen (AA, AT, …, CG, CC) lukumäärä on $4^2 = 16$, joista seitsemän löytyy DNA-sekvenssistä `AGTCTCAG`. Jonot AG ja TC sisältyvät sekvenssiin kahdesti, joten näiden jonojen kohdalle tulee perimätunnistevektoriin luku kaksi. Muut esiintyvät osajonot sisältyvät sekvenssiin kerran, joten niiden kohdalla perimätunnusvektorssa on luku yksi. Loput alkiot perimätunnuksessa ovat

#	A	G	T	C	T	C	A	G
1	A	G	T					
2		G	T	C				
3			T	C	T			
4				C	T	C		
5					T	C	A	
6						C	A	G

#	A	G	T	C	T	C	A	G
1	A	G						
2		G	T					
3			T	C				
4				C	T			
5					T	C		
6						C	A	
7							A	G

Kuvio 12: *DNA-sekvenssin* `AGTCTCAG` *k-mer-osajonojen muodostuminen tilanteissa, kun* $k = 3$ *(vasemmalla) ja* $k = 2$ *(oikealla). Tilanteessa* $k = 3$ *muodostuu kuusi osajonoa, ja tilanteessa* $k = 2$ *muodostuu seitsemän osajonoa.*

nollia, sillä niitä vastaavia osajonoja ei sisälly DNA-sekvenssiin ollenkaan. Taulukossa 2 on esitetty edellisen esimerkin kaikki k-mer-osajonovaihtoehdot sekä niiden frekvenssien muodostama perimätunnus.

Taulukko 2: *DNA-sekvenssin* `AGTCTCAG` *kaikki k-mer-osajonovaihtoehdot ja perimätunnusvektori, kun* $k = 2$.

k-mer:	AA	AT	AG	AC	TA	TT	TG	TC	GA	GT	GG	GC	CA	CT	CG	CC
perimätunnus:	0	0	2	0	0	0	0	2	0	1	0	0	1	1	0	0

Vastaavasti muodostetaan perimätunnukset myös suuremmilla k:n arvoilla tai tilanteessa, jossa emäsvaihtoehtoja on enemmän kuin neljä kappaletta. On huomattava, että perimätunnuksen pituus kasvaa eksponentiaalisesti osajonon pituuden k kasvaessa.

5.3 Konvoluutio

Tässä luvussa esitetään konvoluutio (*convolution*), joka on konvoluutioneuroverkon yksi keskeisimmistä matemaattisista operaatioista. Konvoluutio esitetään tässä luvussa vain vektoreille ja matriiseille, joihin molempiin konvoluutiota tullaan soveltamaan konvoluutioneuroverkossa. Jatkuvien funktioiden konvoluutio sen sijaan sivuutetaan tässä tutkielmassa. Seuraava esitys perustuu pääosin Goodfellow et al. (2016) lukuun 9.1.

Matemaattisesti tarkasteltuna konvoluutio on lineaarinen operaatio, jossa syötefunktiosta f ja niin sanotusta ydinfunktiosta (*kernel function*) g muodostuu uusi funktio s. Konvoluutiota merkitään tähdellä: $s = f * g$. Konvoluutiossa perusidea on painottaa syötteen arvoja ydinfunktion määräämällä tavalla. Vektorien ja matriisien konvoluutiossa ydin, syöte ja konvoluution lopputulos ovat kaikki joko vektoreita tai matriiseja.

Motivaatio konvoluution käyttöön luokittelussa löytyy konvoluution tavasta painottaa syötteen arvoja. Konvoluution lopputuloksena saadussa konvoluutiovektorissa tai -matriisissa alkiot ovat nimittäin sitä suurempia mitä enemmän syöte on muistuttanut ydintä syötteen eri kohdissa. Tästä syystä ydintä kutsutaan etenkin konvoluutioneuro-

verkkojen yhteydessä myös suodattimeksi (*filter*) ja konvoluution tuloksena saatua matriisia tai -vektoria piirrekartaksi (*feature map*). Konvoluutiossa alkiot asettuvat lähelle nollaa, jos jokin kohta syötteessä ei muistuta ydintä. Siispä konvoluutio ikään kuin suodattaa syötteen tietyn kohdan pois, jos se ei muistuta ydintä. Piirrekartta puolestaan kuvaa, millaisia rakenteita (esimerkiksi erilaisia muotoja, kulmia tai reunoja) syötteestä on konvoluutiossa ytimien avulla havaittu ja missä syötteen kohdissa löydetyt rakenteet sijaitsevat. Nämä molemmat ominaisuudet ovat erinomaisia luokittelun kannalta, sillä kun konvoluutio lasketaan käyttäen useita erilaisia ytimiä, voidaan eri luokkia edustavista havainnoista havaita erilaisia piirrekarttoja. Luokittelematon havainto voidaan luokitella, kun havainnon piirteet yhdistetään niitä eniten muistuttaviin piirrekarttoihin. Tämän jälkeen tarkastellaan, mihin luokkaan kyseiset piirrekartat yhdistettin CNN:n opetusvaiheessa, minkä perusteella voidaan määrätä lopullinen havainnon luokka.

Esitetään ensin yksiulotteinen konvoluutio (eli 1D-konvoluutio) kohdassa t. Tällöin syöte f ja ydin g ovat vektoreita, ja 1D-konvoluutio on summa

$$s(t) = \sum_{a \geq 0} f(a)g(t-a). \tag{18}$$

1D-konvoluutiossa syötteen ja ytimen arvot kohdissa a ja $t-a$ määräytyvät vastaavien syöte- ja ydinvektoreiden alkioiden indeksien mukaan. Jos syöte on esimerkiksi vektori $(a,b,c)^T$ ja ydin on esimerkiksi vektori $(\alpha,\beta)^T$, niin tällöin $f(0)=a$, $f(1)=b$, $f(2)=c$ ja $g(0)=\alpha$, $g(1)=\beta$. Indeksöinti alkaa aina indeksistä nolla. Konvoluutiossa ydin on määritelty tyypillisesti vain muutamassa kohdassa (esimerkiksi kolmessa tai viidessä), kun taas syöte voi olla moninkertainen pituudeltaan. Ydinfunktion määrittelyjoukon koko rajoittaa siten summattavien termien määrän.

Konvoluutiolle voidaan helposti johtaa ekvivalentti muoto hyödyntäen muuttujanvaihtoa $u = t-a$. Tällöin saadaan

$$s(t) = \sum_{a \geq 0} f(t-a)g(a).$$

Konvoluutio on siis kommutatiivinen. Lisäksi, jos termissä $f(t-a)$ syötteen arvo lasketaankin kohdassa $t+a$ kohdan $t-a$ sijaan, niin tällöin konvoluutiosta tulee ristikorrelaatio (*cross correlation*):

$$s(t) = \sum_{a \geq 0} f(t+a)g(a). \tag{19}$$

Ristikorrelaatioon (19) liittyy muutamia käytännön hyötyjä verrattuna kaavan (18) konvoluutioon. Erityisesti, ristikorrelaatiossa sekä ytimen että syötteen indeksit molemmat kasvavat kohdan t ja summausindeksin a kasvaessa, mikä tekee sekä konvoluution laskemisesta että myös sen graafisesta tulkinnasta yksinkertaisempaa. Ristikorrelaatio on tästä syystä implementoitu moniin neuroverkko-ohjelmistoihin ja ohjelmointikielten kirjastoihin alkuperäisen konvoluutio-operaation (kaava (18)) sijasta. CNN:n toiminnan kannalta molemmat vaihtoehdot ovat yhtä päteviä. Tässä tutkielmassa konvoluutio-operaationa jatkossa käytetään ristikorrelaatiota, ja termillä konvoluutio tarkoitetaan ristikorrelaatiota.

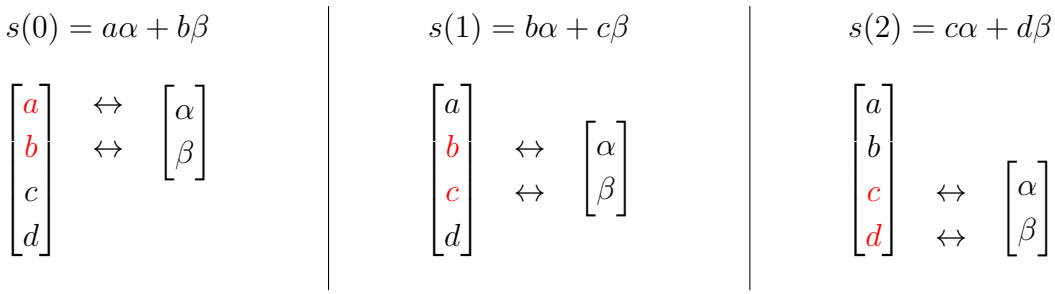
Esimerkkinä yksiulotteisesta konvoluutiosta tarkastellaan syötevektoria, jolle $f(0) = a, \quad f(1) = b, \quad f(2) = c, \quad f(3) = d$ ja ydinvektoria, jolle $g(0) = \alpha, \quad g(1) = \beta$. Ydinfunktio on määritelty vain kohdissa 0 ja 1, joten se rajoittaa summausindeksin. Konvoluutio voidaan laskea niin monessa kohdassa kuin syötteen ja ytimen määrittelyjoukot sallivat. Tässä esimerkissä konvoluutio voidaan laskea vain kohdissa $t = 0$, $t = 1$ ja $t = 2$:

$$s(0) = \sum_{a=0}^{1} f(0+a)g(a) = f(0)g(0) + f(1)g(1) = a\alpha + b\beta,$$
$$s(1) = \sum_{a=0}^{1} f(1+a)g(a) = f(1)g(0) + f(2)g(1) = b\alpha + c\beta,$$
$$s(2) = \sum_{a=0}^{1} f(2+a)g(a) = f(2)g(0) + f(3)g(1) = c\alpha + d\beta.$$

Tulos voidaan esittää vektorimuodossa seuraavasti:

$$f * g = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} * \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} a\alpha + b\beta \\ b\alpha + c\beta \\ c\alpha + d\beta \end{bmatrix}.$$

Konvoluutio-operaation voi laskea myös käyttäen graafista esitystapaa: Konvoluutiovektorin ensimmäinen alkio saadaan, kun ydin viedään ensin syötteen alkuun, minkä jälkeen ytimen ja syötteen kohdakkain olevat alkiot (a ja α sekä b ja β) kerrotaan keskenään ja lopuksi summataan yhteen. Konvoluutiovektorin toinen alkio saadaan vastaavasti, kun ydin ensin siirtyy yhden askeleen eteenpäin syötteellä. Viimeinen konvoluutio saadaan, kun syötteen ja ytimen viimeiset alkiot ovat kohdakkain (eli kun ydin on käynyt koko syötteen läpi). Tätä ideaa on havainnollistettu kuviossa 13.



Kuvio 13: *1D-konvoluution laskenta graafisesti.*

Edellä esimerkissä ydin siirtyi aina yhden askeleen eteenpäin konvoluution laskemisen jälkeen. Ytimen siirtymän pituutta kutsutaan askelpituudeksi (*stride*), ja se on tavallisesti yksi, mutta sitä voidaan myös muuttaa. Suurempi askelpituus johtaa lyhyempään konvoluutiovektoriin s. Jos askelpituus puolestaan on suurempi kuin ytimen pituus, niin tällöin ydin siirtyy liikaa yhdellä askeleella ja näin ohittaa syötteen joitain kohtia. Konvoluution

kaavassa (19) askelpituus r näkyy kohdan t kertoimena:

$$s(t) = \sum_{a \geq 0} f(rt + a)g(a).$$

Askelpituudesta riippumatta konvoluutiossa ulostulovektorin pituus aina pienenee verrattuna syötteen pituuteen. Yleisessä tilanteessa, jos syötevektorin pituus on I ja ydinvektorin pituus K, niin konvoluutiovektorin s pituus on $I - K + 1$. Lyheneminen voidaan estää, jos syötettä keinotekoisesti pidennetään lisäämällä sopiva määrä nollia syötteen eteen tai perään ennen konvoluution toteuttamista. Tämä tapa on niin sanottu nollatäyttö (*zero-padding*). Edellisessä esimerkissä syötevektorin pituuden olisi oltava viisi, jotta konvoluutiovektorin pituus olisi neljä.

Kaksiulotteinen konvoluutio (eli 2D-konvoluutio) on hyvin samankaltainen 1D-konvoluution kanssa. 2D-konvoluutiossa syöte ja ydin ovat matriiseja, joten ydin siirtyy syötteessä sekä vaaka- että pystyakselin suuntaisesti. Olkoon syötematriisi f ja ydinmatriisi g ja merkitään niiden alkioita kohdassa (i, j) merkinnöillä $f(i, j)$ ja $g(i, j)$. Indeksi i viittaa matriisin riviin ja indeksi j puolestaan matriisin sarakkeeseen. 2D-konvoluutio S syötteen kohdassa (i, j) on

$$S(i, j) = (f * g)(i, j) = \sum_{m \geq 0} \sum_{n \geq 0} f(m, n)g(i - m, j - n),$$

jossa indeksit m ja n summataan syötematriisin kaikkien alkioiden vaaka- ja pystysuuntaisten koordinaattien yli. Matriisin ajatellaan sijoittuvan koordinaatistoon siten, että matriisin jonkin kulman alkio sijoittuu kohtaan $(0, 0)$, ja loppujen alkioiden sijaintien koordinaatit ovat positiivisia.

Samoin kuin 1D-konvoluutio, myös 2D-konvoluutio on kommutatiivinen ja 2D-konvoluution sijasta käytetään usein kaksiulotteista ristikorrelaatiota

$$S(i, j) = \sum_{m \geq 0} \sum_{n \geq 0} f(i + m, j + n)g(m, n). \tag{20}$$

Koska ydinmatriisi g (tavallisesti 3×3 -matriisi) on jälleen syötettä pienempi, niin sen määrittelyjoukko rajoittaa summausindeksit m ja n.

Graafisesti tulkittuna 2D-konvoluutio toimii vastaavasti kuin 1D-konvoluutio, mutta ydintä siirretään kahdessa ulottuvuudessa. Konvoluution laskeminen aloitetaan syötematriisin yhdestä kulmasta (esimerkiksi vasemmasta ylänurkasta), joka sijaitsee kohdassa $(0, 0)$. 2D-konvoluutiossa syötematriisista haetaan kohdan $(0, 0)$ ympäriltä ydinmatriisin kokoinen alimatriisi, minkä jälkeen alimatriisin ja ydinmatriisin kohdakkain olevat alkiot kerrotaan keskenään ja tulot summataan yhteen. Tämän jälkeen ydintä siirretään oikealle ja lasketaan uusi konvoluutio kohdassa $(0, 1)$. Kun saavutaan syötematriisin ensimmäisen rivin loppuun, ydinmatriisi siirretään toisen rivin alkuun kohtaan $(1, 0)$ ja tämän rivin konvoluutiot lasketaan kuten edellä. Viimeinen konvoluutio saadaan, kun ydin saavuttaa syötteen oikean alanurkan.

2D-konvoluution tuloksena saadaan $(I_1 - K_1 + 1) \times (I_2 - K_2 + 1)$ -matriisi S, jossa I_1, I_2, K_1 ja K_2 ovat syöte- ja ydinmatriisin leveys ja korkeus. Syötettä keinotekoisesti kasvattamalla (nollatäyttö) voidaan jälleen mahdollistaa, että konvoluutiomatriisi ei

pienene. 2D-konvoluutiossa kasvatus tehdään molempien akselien suuntaan. Myös askelpituuden kasvatus toimii 2D-konvoluutiossa, mutta nyt askelpituutta voidaan muuttaa sekä vaaka- että pystyakselin suuntaisesti (kertoimet r_1 ja r_2, vastaavasti):

$$S(i,j) = \sum_{m\geq 0}\sum_{n\geq 0} f(r_1 i + m, r_2 j + n)g(m,n).$$

Tavallisesti $r_1 = r_2 = 1$.

Esimerkkinä 2D-konvoluutiosta tarkastellaan 3×3 -kokoista syötematriisia f ja 2×2 -kokoista ydinmatriisia g, joiden alkiot ovat määritelty alla olevien matriisien mukaisesti:

$$f := \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad \text{ja} \quad g := \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}.$$

Syötteessä on neljä ytimen kokoista alimatriisia

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix},$$

joissa konvoluutio lasketaan. Konvoluutiomatriisiksi siten saadaan

$$f * g = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \end{bmatrix}.$$

Jatketaan esimerkkiä käyttäen nollatäyttöä: Jotta konvoluution tulos on myös 3×3 -matriisi, niin syötematriisiin riittää lisätä yksi rivi ja yksi sarake nollia:

$$f_{pad.} := \begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Lasketaan esimerkkinä konvoluutiot $S(0,0)$ ja $S(2,1)$ kaavan (20) mukaisesti. Ydinmatriisin koko jälleen rajoittaa summausindeksit siten, että summa käy vain arvojen 0 ja 1 yli, jolloin

$$\begin{aligned} S(0,0) &= \sum_{m=0}^{1}\sum_{n=0}^{1} f_{pad.}(0+m, 0+n)g(m,n) \\ &= f_{pad.}(0,0)g(0,0) + f_{pad.}(0,1)g(0,1) + f_{pad.}(1,0)g(1,0) + f_{pad.}(1,1)g(1,1) \\ &= a\alpha + b\beta + d\gamma + e\delta \end{aligned}$$

ja

$$\begin{aligned} S(2,1) &= \sum_{m=0}^{1}\sum_{n=0}^{1} f_{pad.}(2+m, 1+n)g(m,n) \\ &= f_{pad.}(2,1)g(0,0) + f_{pad.}(2,2)g(0,1) + f_{pad.}(3,1)g(1,0) + f_{pad.}(3,2)g(1,1) \\ &= h\alpha + i\beta + 0\gamma + 0\delta = h\alpha + i\beta. \end{aligned}$$

Muut konvoluutiot lasketaan vastaavasti. Lopputuloksena saadaan 2D-konvoluutiomatriisi

$$f_{pad.} * g = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta & c\alpha + f\gamma \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta & f\alpha + i\gamma \\ g\alpha + h\beta & h\alpha + i\beta & i\alpha \end{bmatrix}.$$

Konvoluutio voidaan esittää myös lineaarialgebran merkinnöin vektorin ja matriisin kertolaskuna. Konvoluution matriisimuotoa monesti suositaan, sillä se on yhteensopiva konvoluutioneuroverkon opettamisessa käytettyjen laskujen kanssa. Annetaan johdatteleva esimerkki konvoluution matriisimuodosta käyttämällä samoja matriiseja (mutta ilman nollatäyttöä) kuin yllä olevassa esimerkissä. Olkoon siten syötematriisi $\boldsymbol{I}$ ja ydin $\boldsymbol{K}$:

$$\boldsymbol{I} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad \text{ja} \quad \boldsymbol{K} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}.$$

Konvoluution laskenta matriisimuodossa aloitetaan tasoittamalla syöte vektoriksi. Siispä muodostetaan sarakevektori $\bar{\boldsymbol{i}}$, jonka pituus on sama kuin syötteen alkioiden lukumäärä, ja joka sisältää kaikki syötteen alkiot. Yläviiva symboloi tasoitettua vektoria.

$$\boldsymbol{I} \xrightarrow{\text{tasoitus}} (a, b, c, d, e, f, g, h, i)^T = \bar{\boldsymbol{i}}.$$

Edellisen lisäksi luodaan uusi matriisi $\boldsymbol{C}$

$$\boldsymbol{C} = \begin{bmatrix} \alpha, \beta, 0, \gamma, \delta, 0, 0, 0, 0 \\ 0, \alpha, \beta, 0, \gamma, \delta, 0, 0, 0 \\ 0, 0, 0, \alpha, \beta, 0, \gamma, \delta, 0 \\ 0, 0, 0, 0, \alpha, \beta, 0, \gamma, \delta \end{bmatrix},$$

joka on muodostettu seuraavalla periaatteella: Matriisin $\boldsymbol{C}$ jokaisella rivillä lasketaan konvoluutio yhden syötematriisin $\boldsymbol{I}$ alimatriisin kanssa. Alimatriiseja on neljä kappaletta, joten matriisissa $\boldsymbol{C}$ on siten neljä riviä. Jokainen matriisin $\boldsymbol{C}$ sarake puolestaan vastaa yhtä syötematriisin alkiota, joten sarakkeita on yhdeksän. Ensimmäisellä rivillä lasketaan konvoluutio matriisin $\boldsymbol{I}$ ensimmäisen alimatriisin kanssa. Tässä alimatriisissa on alkiot a, b, d ja e. Matriisin $\boldsymbol{C}$ ensimmäisen rivin kolmas alkio nolla vastaa vektorin $\bar{\boldsymbol{i}}$ kolmatta alkiota c. Koska alkio c ei ole alimatriisissa, se ei ole myöskään mukana konvoluutiossa, jolloin sen kohdalla on luku nolla. Vastaavasti kaksi ensimmäistä alkiota ovat α ja β, sillä niitä vastaavat alkiot a ja b puolestaan ovat mukana konvoluutiossa. Jokainen rivi matriisissa $\boldsymbol{C}$ vastaa yhden alimatriisin kanssa laskettua konvoluution vaihetta, joten vastaavasti voidaan muodostaa myös muut matriisin $\boldsymbol{C}$ rivit.

Lopullinen konvoluutiomatriisi saadaan kertolaskun $\boldsymbol{C}\bar{\boldsymbol{i}}$ tuloksena. $\boldsymbol{C}\bar{\boldsymbol{i}}$ on vektori, joka sisältää konvoluutiomatriisin $\boldsymbol{I} * \boldsymbol{K}$ alkiot. Kun tämä vektori kootaan takaisin matriisimuotoon vastaavassa järjestyksessä syötteen tasoituksen kanssa, saadaan varsinainen konvoluutiomatriisi:

$$\boldsymbol{C}\,\bar{\boldsymbol{i}} = (a\alpha + b\beta + d\gamma + e\delta, b\alpha + c\beta + e\gamma + f\delta, d\alpha + e\beta + g\gamma + h\delta, e\alpha + f\beta + h\gamma + i\delta)$$

$$\xrightarrow{\text{palautus matriisiksi}} \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \end{bmatrix} = \boldsymbol{I} * \boldsymbol{K}.$$

(Aggarwal 2018, 335–337.)

1D-konvoluution voi myös toteuttaa vastaavasti vektorimuodossa. Tällöin syötettä ei tarvitse tasoittaa, sillä 1D-konvoluutiossa ydin ja syöte ovat vektoreita jo alusta alkaen. Olkoon esimerkiksi syötevektori $f = (a, b, c, d)^T$ ja ydinvektori $g = (\alpha, \beta)^T$. Tällöin 1D-konvoluutiovektori saadaan laskemalla $\boldsymbol{C}f$, jossa

$$\boldsymbol{C} = \begin{bmatrix} \alpha, \beta, 0, 0 \\ 0, \alpha, \beta, 0 \\ 0, 0, \alpha, \beta \end{bmatrix}.$$

Matriisi $\boldsymbol{C}$ on muodostettu samalla periaatteella kuin 2D-konvoluution esimerkissä esitettiin.

5.4 Tiivistys

Tiivistys (*pooling*) on toinen konvoluutioneuroverkon keskeinen operaatio, ja se liittyy läheisesti konvoluutioon. Konvoluutioneuroverkossa tyypillisesti ensin toteutetaan yksi tai useampia konvoluutioita, minkä jälkeen saatu tulos tiivistetään tiivitysoperaatiolla.

Tiivistys ottaa syötteeksi vektorin (1D-tiivistys) tai matriisin (2D-tiivistys) ja palauttaa vastaavasti vektorin tai matriisin, jonka kokoa on pienennetty. Kuten konvoluutiossa, myös tiivistyksessä käytetään ydintä, jota siirretään syötteessä käyden sen kokonaan läpi. Tyypillinen ytimen koko 2D-tiivistyksessä on 2×2 -matriisi. Tiivistyksessä ydin itse ei kuitenkaan sisällä alkioita, vaan ytimen koko on avainasemassa. Tiivistyksessä jokaiselta ytimen määräämältä alueelta valitaan tämän alueen syötteen alkioista suurin. Tämän jälkeen ydin siirtyy seuraavalle alueelle. Kuten konvoluutiossa, myös tiivistyksessä ytimen siirtymiseen vaikuttaa askelpituus. Kuitenkin toisin kuin konvoluutiossa, jossa askelpituus on yleensä yksi, tiivistyksessä askelpituus on tyypillisesti ytimen leveyden suuruinen. Ytimen leveyden pituinen askelpituus johtaa siihen, että ydin siirtyy aina sellaiselle alueelle, jota ei ole vielä käsitelty. Ydin ei tällöin sijoitu päällekkäin edellisen, jo käsitellyn, alueen kanssa. Edellisessä esimerkissä, jossa ytimen koko on 2×2 ja askelpituus on 2, lopputuloksena saadaan matriisi, jonka leveys ja korkeus ovat puolittuneet. Yksiulotteisen syötteen tapauksessa tiivistys toimii samoin, mutta ydin (esimerkiksi 2-pituinen vektori) siirtyy vain yhden akselin suuntaisesti.

Edellä kuvattu tiivistys on niin sanottu maksimitiivistys (MaxPool). Vaihtoehtoinen tiivistys on keskiarvotiivistys, jossa maksimin sijaan lasketaan alkioiden keskiarvo. Esimerkkinä maksimitiivistyksestä tarkastellaan 4×4 -kokoista syötettä, jolle tehdään edellä

kuvattu maksimitiivistys (ytimen koko 2×2 ja askelpituus 2):

$$\text{MaxPool} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} = \begin{bmatrix} \max\{a,b,e,f\} & \max\{c,d,g,h\} \\ \max\{i,j,m,n\} & \max\{k,l,o,p\} \end{bmatrix}.$$

Tiivistys puolittaa syötteen leveyden ja korkeuden, joten saadaan 2×2 -matriisi.

Kuvan tunnistuksen kannalta tiivistys nimensä mukaisesti tiivistää kuvan informaatiota. Jokaiselta 2×2 -kokoiselta alueelta tiivistyksessä lasketaan tätä aluetta parhaiten kuvaava tunnusluku, ja kuvaavimman tunnusluvun omaava alue jatkaa eteenpäin CNN:ssä.

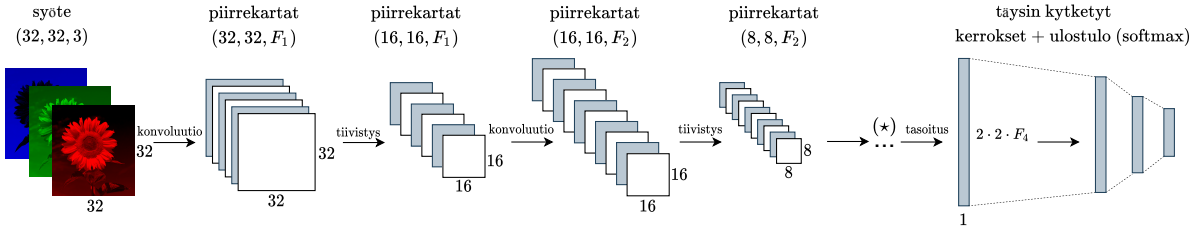
5.5 Konvoluutioneuroverkon rakenne

Konvoluutioneuroverkon rakenne on samanlainen kuin minkä tahansa muunkin eteenpäinsyöttävän neuroverkon rakenne. CNN:ssä on syötekerros, piilokerrokset sekä ulostulokerros, josta syötteen ennustettu luokka selviää. Seuraavaksi esitetään konvoluutioneuroverkon rakenne, kun syöte on värikuva. Tästä rakenteesta voidaan helposti johtaa myös yksiulotteiselle syötteelle sopiva konvoluutioneuroverkko, mitä tarkastellaan luvun lopussa. Esitys perustuu julkaisun James et al. (2021) lukuun 10.3.

CNN:ssä ensimmäiset piilokerrokset koostuvat konvoluutio- ja tiivistysoperaatioista, joiden järjestys ja lukumäärä ovat neuroverkon laatijan päätettävissä. Konvoluutiooperaatioita sisältävä piilokerros on nimeltään konvoluutiokerros (*convolutional layer*) ja tiivistyksiä sisältävä piilokerros on puolestaan tiivistyskerros (*pooling layer*). Konvoluutio- ja tiivistyskerroksia sijoitetaan konvoluutioneuroverkkoon siten, että tyypillisesti tiivistyskerros seuraa aina konvoluutiokerrosta. Konvoluutiokerroksia voi kuitenkin myös sijoittaa useampia peräkkäin. Tiivistyskerroksia sen sijaan ei sijoiteta peräkkäin.

Kuviossa 14 on esitetty esimerkki eräästä 2D-konvoluutioneuroverkon (2D-CNN) rakenteesta. Kuviossa konvoluutioneuroverkon syöte on $(I_1, I_2, 3)$-kokoinen värikuva, ja olkoon kuvan koko esimerkissä 32×32. Syöte on siis $(32, 32, 3)$-kokoinen tensori. CNN:n käyttäjä on etukäteen päättänyt jokaisella konvoluutiokerroksella olevien ydinten lukumäärän. Ydinten lukumäärä vaikuttaa siihen, kuinka monta piirrekarttaa CNN:n konvoluutiokerroksella lasketaan. Ydinten dimensio riippuu aina konvoluutiokerroksen syötteestä: Värikuvalle ytimet ovat kolmiulotteisia tensoreita, harmaasävykuvalle kaksiulotteisia matriiseja ja DNA-sekvenssille yksiulotteisia vektoreita. Värikuvan tapauksessa ydin on siis $(K_1, K_2, 3)$-kokoinen tensori. Ytimen ja syötteen kanavien lukumäärät ovat siis samat. Tavanomaiset ytimen leveys ja korkeus ovat 3, joten esimerkissä ydin olkoon $(3, 3, 3)$-tensori.

Annetaan esimerkki ensimmäisen konvoluutiokerroksen toiminnasta: Päätetään, että ensimmäisellä konvoluutiokerroksella on F_1 kappaletta $(3, 3, 3)$-kokoisia ytimiä. Tällöin konvoluutiokerroksella lasketaan F_1 kappaletta sellaisia konvoluutioita, joissa sekä syöte ja ydin ovat kolmiulotteisia tensoreita. Nämä konvoluutiot lasketaan käyttäen 2D-



Kuvio 14: *Konvoluutioneuroverkon rakenne, kun syöte on värikuva. Tähdellä* $(\star)$ *merkityssä kohdassa on kaksi konvoluutiokerrosta ja kaksi tiivistyskerrosta, jolloin piirrekartat muodostavat* $(2, 2, F_4)$*-kokoisen tensorin ennen tasoitusta.*

konvoluutiota jokaiselle kolmiulotteisen tensorin kaksiulotteiselle matriisille. Kaiken kaikkiaan kaavan (20) mukaisia 2D-konvoluutioita lasketaan $3F_1$ kappaletta. Kun kaikki konvoluutiot on laskettu, on saatu yhteensä F_1 kappaletta kolmiulotteisia tensoreita. Jokainen kolmiulotteinen tensori sisältää tässä vaiheessa kolme kaksiulotteista piirrekarttaa.

Tensoreille tehdään tässä vaiheessa dimension pienennys, jossa kolmiulotteiset tensorit muutetaan kaksiulotteisiksi tensoreiksi. Pienennys tehdään summaamalla kolmiulotteisen tensorin kolme kaksiulotteista piirrekarttaa yhteen. Dimension pienennyksen jälkeen konvoluutiokerroksella on nyt F_1 kappaletta kaksiulotteisia piirrekarttoja. Nämä piirrekartat kootaan jälleen uuteen kolmiulotteiseen tensoriin, joka on nyt kooltaan $(32, 32, F_1)$. Tämä tensori syötetään CNN:ssä seuraavalle kerrokselle syötteenä:

$$\text{syöte } (32, 32, 3) \xrightarrow{\text{2D-konvoluutio } (F_1 \text{ ydintä})} \text{piirrekartat } (32, 32, F_1).$$

Dimension pienentämisen seurauksena konvoluutiokerroksen ulostulona saadussa tensorissa on sama määrä kaksiulotteisia piirrekarttoja kuin konvoluutiokerroksella on ytimiä (ytimiä on F_1 kappaletta, jolloin tensorin dimensio on $(32, 32, F_1)$), mikä on toivottu lopputulos. Myös seuraavilla konvoluutiokerroksilla käytetään dimension pienennystä vastaavan lopputuloksen saamiseksi.

Konvoluutiokerroksella käytetään konvoluution lisäksi myös aktivaatiofunktiota, jota käytetään konvoluutio-operaation jälkeen. CNN:n aktivaatiofunktion valinta on vastaava prosessi kuin tavallisten eteenpäinsyöttävien neuroverkkojen tapauksessa. ReLU on näin ollen hyvä valinta myös CNN:n aktivaatiofunktioksi. CNN:ssä aktivaatiofunktiolla operoidaan jokainen piirrekartan alkio erikseen.

Konvoluution jälkeen CNN:ssä voi olla joko toinen konvoluutiokerros tai tiivistyskerros. Esitellään seuraavaksi tiivistyskerros, kun ytimen koko on 2×2, valittu tiivistys on maksimitiivistys ja nollatäyttö on käytössä. Tiivistyskerros ottaa syötteenään $(32, 32, F_1)$-kokoisen tensorin edelliseltä konvoluutiokerrokselta, jolle tiivistysoperaatio toteutetaan. Tensorin tiivistyksessä jokainen 32×32 -kokoinen kaksiulotteinen piirrekartta (joita on F_1 kappaletta) tiivistetään. Näin tiivistyskerroksen ulostulo on $(16, 16, F_1)$-kokoinen tensori:

$$\text{piirrekartat } (32, 32, 3) \xrightarrow{\text{MaxPool } (2 \times 2)} \text{piirrekartat } (16, 16, F_1).$$

Seuraavana kerroksena CNN:ssä on jälleen konvoluutiokerros, jossa olkoon F_2 kappaletta $(3, 3, F_1)$-kokoisia ytimiä. Konvoluution jälkeen konvoluutiokerroksella on F_2 kappaletta

$(16, 16, F_1)$ kokoisia tensoreita. Käytetään jälleen dimension pienennystä, jolloin saadaan yksi $(16, 16, F_2)$-tensori.

Tämän jälkeen voidaan taas käyttää joko tiivistyskerrosta tai myös toista konvoluutiokerrosta. Tällaista konvoluutio-tiivistys-sykliä toistetaan konvoluutioneuroverkossa neuroverkon käyttäjän valintojen mukaisesti. Koska piirrekarttojen koko pienenee tiivistyksessä, on tyypillistä, että ytimien määrää kasvatetaan seuraavilla konvoluutiokerroksilla (ts. $F_2 > F_1$). On myös huomattava, että tiivistyksiä ei voi tehdä loputtomasti, sillä pienimmillään piirrekartan koko on 1×1 -kokoinen piste.

Jatketaan edellä aloitettua konvoluutioneuroverkkoa vielä kahdella konvoluutiokerroksella ja kahdella tiivistyskerroksella, jolloin sekä konvoluutio- että tiivistyskerroksia on molempia neljä kappaletta. Olkoon kolmannella ja neljännellä konvoluutiokerroksilla F_3 ja F_4 kappaletta ytimiä ja tehdään tiivistys samoin kuin edellä. Tällöin CNN:n rakenne jatkuu seuraavasti:

$$\begin{aligned}
\text{piirrekartat } (16, 16, F_1) &\xrightarrow{\text{2D-konvoluutio } (F_2 \text{ ydintä})} \text{piirrekartat } (16, 16, F_2) \\
&\xrightarrow{\text{MaxPool } (2 \times 2)} \text{piirrekartat } (8, 8, F_2) \\
&\xrightarrow{\text{2D-konvoluutio } (F_3 \text{ ydintä})} \text{piirrekartat } (8, 8, F_3) \\
&\xrightarrow{\text{MaxPool } (2 \times 2)} \text{piirrekartat } (4, 4, F_3) \\
&\xrightarrow{\text{2D-konvoluutio } (F_4 \text{ ydintä})} \text{piirrekartat } (4, 4, F_4) \\
&\xrightarrow{\text{MaxPool } (2 \times 2)} \text{piirrekartat } (2, 2, F_4).
\end{aligned}$$

Konvoluutio- ja tiivistyskerrosten jälkeen alkuperäinen värikuva on tiivistynyt niin, että sen leveys ja korkeus ovat pienentyneet huomattavasti ja samalla värikuvan kanavien määrä on kasvanut suuremmaksi ($3 \to F_1 \to F_2 \to F_3 \to F_4$). Esimerkissä kuvan koko on enää vain 2×2, ja kanavien lukumäärä on lopulta F_4. Tässä vaiheessa konvoluutioneuroverkkoa piirrekartoille tehdään tasoitus (*flattening*). Tasoituksessa useampiulotteinen tensori muutetaan vektoriksi järjestämällä tensorin alkiot yhteen vektoriin. Tavalla, jolla alkiot järjestetään, ei ole merkitystä, sillä tässä vaiheessa tensorin spatiaalinen rakenne hajoaa. Esimerkin tapauksessa tasoituksen seurauksena saadun vektorin pituus on $2 \cdot 2 \cdot F_4$, ja tämä vektori annetaan syötteenä lopuille konvoluutioneuroverkon kerroksille:

$$\text{piirrekartat } (2, 2, F_4) \xrightarrow{\text{tasoitus}} (2 \cdot 2 \cdot F_4)\text{-vektori.}$$

Konvoluutioneuroverkon loput kerrokset ovat tavallisia täysin kytkettyjä kerroksia, jotka toimivat samalla tavalla kuin luvussa 3 esitettiin: Kun piirrekarttojen tasoitus on tehty, niin saatu vektori syötetään nyt täysin kytketyille kerroksille ja lopulta ulostulokerrokselle. Jokainen vektorin alkio käsitellään omana neuroninaan, joista lähtevät painot vaikuttavat seuraavan täysin kytketyn kerroksen neuroneihin. Lopulta ulostulokerroksella käytetään softmax-aktivaatiofunktiota, jonka laskemien softmax-todennäköisyyksien perusteella luokittelu tehdään.

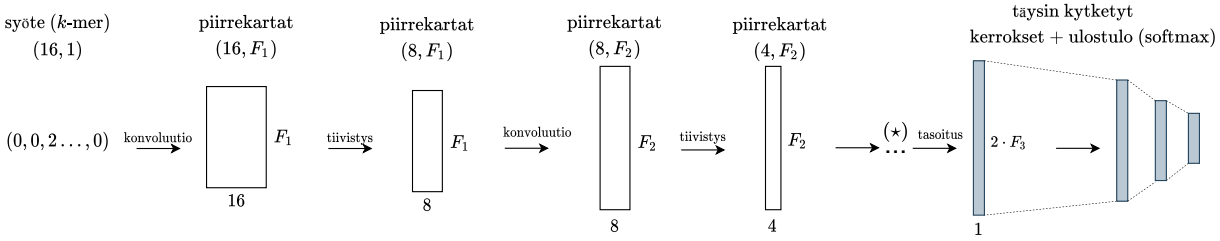
Jos syöte on harmaasävykuva värikuvan sijaan, niin ainoa ero edellä kuvattuun CNN:n rakenteeseen tulee ensimmäisellä konvoluutiokerroksella: Koska syöte on kaksiulotteinen matriisi, niin tällöin ytimetkin ovat matriiseja ja ensimmäisellä konvoluutiokerroksella

ei tarvitse tehdä dimension pienennystä. CNN:n rakenne on tästä eteenpäin sama kuin värikuvalla.

Jos syöte on vektori, niin tällöin puhutaan 1D-konvoluutioneuroverkosta (1D-CNN). 1D-CNN:n rakenne on muuten vastaava kuin 2D-CNN:n, mutta jokaisella kerroksella kaikkien tensoreiden (syöte ja ytimet) dimensiot ovat yhtä pienempiä. Toisin sanoen jokainen kolmiulotteinen tensori on kaksiulotteinen, ja jokainen kaksiulotteinen tensori on yksiulotteinen. Lisäksi 2D-konvoluution ja 2D-tiivistyksen sijaan käytetään vastaavien operaatioiden 1D-versioita. 1D-CNN:n tensorien dimensiot konvoluutio- ja tiivistyskerroksilla voidaan esittää seuraavasti:

$$
\begin{aligned}
\text{syöte } (16,1) &\xrightarrow{\text{1D-konvoluutio } (F_1 \text{ ydintä})} \text{piirrekartat } (16,F_1)\\
&\xrightarrow{\text{MaxPool}} \text{piirrekartat } (8,F_1)\\
&\xrightarrow{\text{1D-konvoluutio } (F_2 \text{ ydintä})} \text{piirrekartat } (8,F_2)\\
&\xrightarrow{\text{MaxPool}} \text{piirrekartat } (4,F_2)\\
&\xrightarrow{\text{1D-konvoluutio } (F_3 \text{ ydintä})} \text{piirrekartat } (4,F_3)\\
&\xrightarrow{\text{MaxPool}} \text{piirrekartat } (2,F_3).
\end{aligned}
$$

Viimeisen tiivistyskerroksen jälkeen piirrekartan dimensio on $(2, F_3)$, joka tasoitetaan vektoriksi. Tästä eteenpäin CNN:ssä käytetään täysin kytkettyjä kerroksia samalla tavalla kuin aiemmin esitettiin värikuvan yhteydessä. Esimerkki kokonaisen 1D-konvoluutioneuroverkon rakenteesta on esitelty kuviossa 15.



Kuvio 15: *Konvoluutioneuroverkon rakenne, kun syöte on DNA-sekvenssiä* `AGTCTCAG` *vastaava k-mer-vektori kuten taulukossa 2. Tähdellä* $(\star)$ *merkityssä kohdassa on yksi konvoluutiokerros ja yksi tiivistyskerros, jolloin piirrekartat muodostavat* $(2, F_3)$*-kokoisen tensorin ennen tasoitusta.*

5.6 Vastavirta-algoritmi konvoluutioneuroverkossa

Konvoluutioneuroverkossa opetettavat parametrit ovat konvoluutiokerrosten ydinten alkiot ja täysin kytkettyjen kerrosten painot. Kaikki parametrit opetetaan vastavirta-algoritmilla, joka muistuttaa monilta osin luvussa 4.4 esitettyä vastavirta-algoritmia. (Aggarwal 2018, 332.) CNN:n lopussa olevien täysin kytkettyjen kerrosten toiminnan osilta vastavirta-algoritmi on itse asiassa täysin sama. Lisäksi CNN:n tiivistysoperaatioon ei liity opetettavia parametreja lainkaan. Näin ollen, ainoastaan konvoluutiokerroksella vastavirta-algoritmia pitää säätää CNN:lle sopivaksi.

Konvoluutioneuroverkossa vastavirta-algoritmin toiminnan perusperiaate on tasoittaa kunkin konvoluutiokerroksen syöte ja ytimet sellaisiksi vektoreiksi ja matriiseiksi, joita voidaan suoraan soveltaa vastavirta-algoritmissa (Aggarwal 2018, 335). Toisin sanoen, jokaisella konvoluutiokerroksella useampiulotteinen syöte muunnetaan yksiulotteiseksi vektoriksi, jota käsitellään vastavirta-algoritmissa samalla tavalla kuin neuronivektoreita $\boldsymbol{h}^{(i)}$. Vastaavasti jokaisella konvoluutiokerroksella konvoluution ytimet muutetaan sellaisiksi matriiseiksi, joita käsitellään painomatriiseja $\boldsymbol{W}_i$ vastaavalla tavalla. Vastavirta-algoritmin suorituksen jälkeen syötteet ja ytimet muutetaan takaisin alkuperäiseen muotoon.

Aiemmin luvussa 5.3 konvoluutio-operaatio esitettiin matriisin $\boldsymbol{C}$ ja tasoitetun syötevektorin $\overline{\boldsymbol{i}}$ kertolaskuna $\boldsymbol{C}\,\overline{\boldsymbol{i}}$. Tämä on vastavirta-algoritmin etenemisvaiheessa laskettu konvoluutio. Peruutusvaiheessa konvoluutiota tarkastellaan käänteisessä järjestyksessä. Toisin sanoen, konvoluutiokerros saa peruutusvaiheessa syötteenään edelliseltä kerrokselta virhefunktion tasoitetun gradienttivektorin $\bar{\boldsymbol{g}}$, jossa gradientit on laskettu piirrekarttojen suhteen. Tällöin konvoluution syötteen suhteen lasketut virhefunktion gradientit sisältävä tasoitettu vektori on $\boldsymbol{C}^T\bar{\boldsymbol{g}}$. Edellä kuvattu tilanne vastaa tavallisen neuroverkon peruutusvaiheen vastavirta-algoritmin (Algoritmi 2) riviä 6, jossa etenemisvaiheen painomatriisin ($\boldsymbol{W}_i^T$) transpoosia ($\boldsymbol{W}_i$) käytetään peruutusvaiheessa siirryttäessä seuraavalle kerrokselle. Näin konvoluutioneuroverkon peruutusvaiheessa voidaan laskea virhefunktion gradienttivektori jokaisella konvoluutiokerroksella, minkä ansiosta konvoluutioiden ytimien alkiot voidaan jokaisella konvoluutiokerroksella päivittää esimerkiksi SGD:llä.(Aggarwal 2018, 337.)

6 Aineiston analyysi

Tässä luvussa ensin esitetään DNA-sekvenssien esikäsittelyyn liittyvät yksityiskohdat. Tämän jälkeen esitellään valitut neuroverkkoluokittimet sekä niiden hyperparametrien valinnat ja käytetyt ohjelmistot. Lopuksi tarkastellaan luokittelun tuloksia.

Luokittelua varten DNA-sekvenssit on muutettu tensoreiksi indikaattorivektoreihin ja k-mer-osajonoihin perustuvilla tavoilla, joita tarkasteltiin aiemmin luvussa 5.2. Valtaosa PROTAX-aineiston DNA-sekvensseistä on sellaisia, jotka koostuvat emäksistä `A`, `T`, `G`, `C` ja `N`. Viivalla (`-`) merkityt emäkset on myös muutettu `N`-emäkseksi. Aineistossa on myös yhteensä 34 DNA-sekvenssiä, joissa esiintyy myös muita emäsmerkintöjä (ks. Cornish-Bowden 1985). Yksinkertaisuuden vuoksi nämä sekvenssit on poistettu kokonaan aineiston esikäsittelyvaiheessa.

Koska emäsvaihtoehtoja on viisi kappaletta, niin kaikkien indikaattorivektoreiden pituus on viisi (aiemmin neljä) ja emäs `N` koodataan vektorilla $(0, 0, 0, 0, 1)$. Lisäksi kaikkia k-mer-osajonovaihtoehtoja on nyt 5^k kappaletta (aiemmin 4^k). DNA-sekvenssien muunnokset tensoreiksi tapahtuvat muuten samoin kuin aiemmin esitettiin.

DNA-sekvenssin esikäsittely määrää, millainen konvoluutioneuroverkko aineistoon sovitetaan. Indikaattorivektoreiden mukaan koodattujen sekvenssien luokittelu tehdään 2D-CNN:llä, ja k-mer-osajonoihin perustuva esikäsittely puolestaan johtaa 1D-CNN:n käyt-

töön. 2D- ja 1D-CNN ovat oleellisesti kaksi eri neuroverkkoluokitinta, joilla on molemmilla omanlaisensa arkkitehtuuri. Ei siis ole suoraan mahdollista käyttää täysin samanlaista rakennetta molemmissa neuroverkoissa ja näin verrata DNA-sekvenssin esikäsittelyn vaikutusta luokittelutarkkuuteen.

Ennen konvoluutioneuroverkkojen lopullisen rakenteen valintaa jo odotettiin, että 2D-CNN:n rakenteen on oltava monimutkaisempi kuin 1D-CNN:n rakenteen, jos molemmilla neuroverkoilla halutaan saavuttaa samankaltaisia tuloksia. Ennakkokäsitystä voi perustella esimerkiksi kaksiulotteisen syötteen suuremmalla koolla verrattuna yksiulotteiseen vektoriin, ja 2D-konvoluutiossa esiintyvien parametrien suuremmalla lukumäärällä verrattuna 1D-konvoluutioon. Tästä syystä 2D-CNN:n rakenteen monimutkaisuus nostettiin jopa tarpeettoman suureksi ja 1D-CNN:n rakenteen monimutkaisuus pidettiin kohtuullisena. 2D-CNN:n rakennetta kuitenkin edelleen rajoittivat käytössä oleva tietokoneen laskentakapasiteetti (erityisesti grafiikkasuorittimen muisti) sekä mahdollinen ylisovittuminen, jota ei sallittu.

Valitut neuroverkkorakenteet on esitetty liitteen A kuviossa 19. Konvoluutiot ja tiivistykset laskettiin molemmissa neuroverkoissa hyödyntämällä nollatäyttöä, jossa konvoluution ja tiivistyksen ulostulona saatu vektori tai matriisi on syötteen kanssa saman kokoinen. 1D-CNN:ssä tiivistyksen ja konvoluution ytimien pituuksiksi valittiin (3), kun 2D-CNN:ssä vastaavat ytimien koot olivat $(2, 2)$. Tiivistysten askelpituus (1D: r, 2D: r_1 ja r_2) oli kaksi, jolloin tiivistyksissä alueet eivät lomitu. 2D-CNN:ssä tiivistyksissä piti huomioida tarkasti syötteen leveys (5). Tällöin (nollatäytön kanssa) tiivistyskerroksia voi olla maksimissaan kolme kappaletta, ja tämän jälkeen syötteen leveys on enää vain yksi.

Neuroverkkojen opetuksessa käytettiin stokastista gradienttimenetelmää, kun eräkoot (U) olivat 64 (1D-CNN) ja 32 (2D-CNN) ja oppimisnopeus (η) oli 0.005. Neuroverkoissa käytettiin regularisaatiomenetelmänä eränormalisointia jokaisen tiivistyskerroksen jälkeen. SGD:ssä epookkien lukumäärä oli 30–75 välillä riippuen CNN:n rakenteen monimutkaisuudesta sekä siitä, kuinka monta luokkaa luokittelussa oli mukana. Epookkien lukumäärä osoittautui jokaisessa tilanteessa tarpeettoman suureksi, sillä opetusprosessin lopussa neuroverkon parametrit eivät enää päivittyneet, vaan konvergenssi oli saavutettu. Mikään neuroverkko ei kuitenkaan validointiaineiston perusteella ylisovittunut, vaikka epookkeja olikin ”liikaa”, joten liiallinen epookkien määrä ei näin ollen varsinaisesti haitannut.

Mallien opetusaikojen vertailu käyttäen kokonaisopetusaikaa (aika, joka kuluu mallin opetukseen alusta loppuun) ei ole edellä kuvattujen seikkojen vuoksi järkevää. Ensiksi, ajat ovat suurempia kuin optimaalisessa tilanteessa, sillä epookkeja laskettiin tarpeettoman paljon. Toiseksi, malleissa käytettiin eri määrä epookkeja, mikä oleellisesti vaikuttaa kokonaisopetusaikaan. Kokonaisopetusajan sijaan opetuksen aikana mitattiin jokaiseen epookkiin kulunut aika sekunteina (s/epookki). Tämän jälkeen näistä ajoista otettiin mediaani, joka on hyvä keskiluku yhden epookin laskemiseen kuluneelle ajalle. Mediaania käytettiin (esimerkiksi keskiarvon sijaan), sillä ensimmäisen epookin laskeminen saattaa kestää pidempään kuin muut riippuen, onko aineisto jo ladattu tietokoneen muistiin.

CNN:t opetettiin tilanteissa, joissa mukaan otettiin joko 10, 100, 500 tai 1000 lajia.

Lajit valittiin mukaan lajikohtaisten havaintojen lukumäärän suhteen suuruusjärjestyksessä, eli esimerkiksi ensimmäisessä tilanteessa mukaan otettiin sellaiset kymmenen lajia, joista oli eniten havaintoja koko aineistossa. 1D- ja 2D-konvoluutioneuroverkot on implementoitu TensorFlow-kirjastossa (TensorFlow Developers 2022), jota tässä tutkielmassa käytettiin Python-ohjelmointikielellä (Python Software Foundation 2022). Myös aineiston esikäsittely tehtiin pääosin Pythonilla. DNA-sekvenssien fasta-formaattien esikäsittelyssä hyödynnettiin Pythonin lisäksi R-ympäristön (R Core Team 2022) Biostrings-pakettia (Pagès et al. 2022).

TensorFlow tarjoaa tuen laitteistokiihdytykselle, jonka ansiosta neuroverkkojen opetusajat pysyivät pieninä. Opetuksessa käytössä oli kuluttajakäyttöön suunniteltu Nvidia GeForce GTX 3070 8GB -grafiikkasuoritin, jonka ainoaksi rajoittavaksi tekijäksi osoittautui videomuistin suhteellisen vähäinen määrä (8 GB). Vähäisen videomuistin takia 2D-CNN:ää opetettaessa eräkoko piti laskea 64:stä 32:een, jotta videomuisti riitti. Vertailun vuoksi ammattikäyttöön tarkoitetut ja syväoppimissovelluksiin suunnitellut grafiikkasuorittimet tarjoavat jopa kymmenkertaisen määrän videomuistia (esim. Nvidia A100 Tensor Core 80GB).

Lopullisen luokittelun tulokset on esitetty taulukossa 3. Taulukossa on DNA-sekvenssien kokonaismäärä kussakin tilanteessa (10, 100, 500 tai 1000 luokkaa) sekä lajikohtaiset havaintojen lukumäärät. 1D-CNN on opetettu tilanteissa joissa k-mer-osajonojen pituus k on neljä, viisi ja kuusi, ja tuloksista on raportoitu 1D-CNN:n opetettavien parametrien lukumäärä, yhden epookin laskemiseen kulunut aika (s/epookki) ja luokittelutarkkuus. Lisäksi taulukossa on raportoitu vastaavat tiedot myös 2D-CNN:llä toteutetusta luokittelusta. Luokittelutarkkuus, joka laskettiin käyttäen erillistä testiaineistoa, on oikein luokiteltujen sekvenssien osuus kaikista testiaineiston sekvensseistä. Taulukon lisäksi opetusaika ja luokitteluvirhe (väärin luokiteltujen DNA-sekvenssien osuus testiaineistossa) ovat esitetty graafisesti kuviossa 16. Testiaineisto muodostettiin valitsemalla testiaineistoon satunnaisesti 25 % kaikista luokittelussa mukana olleista sekvensseistä ja niitä vastaavista luokista. Opetuksen aikana sama testiaineisto toimi myös validointiaineistona. Neuroverkkojen opetusta seurattiin laskemalla luokittelutarkkuus ja ristientropia opetus- ja validointiaineistoilla jokaisen epookin kohdalla. Kuvaajien perusteella malleissa ei ole havaittavissa ali- eikä ylioppimista, joten neuroverkot opettuivat onnistuneesti. Kuvaajat on esitetty liitteessä B.

Yleisesti ottaen mallien luokittelutarkkuudet ovat korkeita; jopa 1000 lajin luokittelussa tarkkuudeksi saatiin suurimmillaan noin 95 % (1D-CNN) ja 93.5 % (2D-CNN). Näyttäisi siltä, että tarkasti valitut DNA-viivakoodit ovat luotettavia lajien tunnistamisen suhteen, vaikka lajikohtaisia DNA-sekvenssejä olisi saatavilla vain vähän. Lisäksi TensorFlow:n tarjoama tuki grafiikkasuorittimen laitteistokiihdytykselle piti molempien neuroverkkojen opetusajat kohtuullisina. Vertailun vuoksi kymmenen lajin luokittelu 1D-CNN:llä ilman laitteistokiihdytystä (ts. lasketaan vain prosessorilla (CPU), kun käytössä on Intel Core i5 12600K) tilanteessa $k = 5$ kestää noin 2.25 s/epookki (vrt. 0.25 s/epookki).

1D-CNN:ssä k-mer-osajonojen pituudella vaikuttaisi olevan luokittelutarkkuutta pa-

rantava vaikutus (etenkin 1000 lajin luokittelussa), mikä oli odotettavissa: Pidemmissä osajonoissa säilyy enemmän informaatiota emästen järjestyksestä. Kuitenkin pidempien k-mer-osajonojen käyttö neuroverkossa vaatii runsaasti enemmän laskentatehoa, sillä syötteen koko kasvaa eksponentiaalisesti k:n kasvaessa. Nimittäin, myös tilannetta $k = 7$ yritettiin opettaa samalla neuroverkkorakenteella, jolloin neuroverkon opetus kuitenkin epäonnistui grafiikkasuorittimen muistin loppuessa. Tilannetta voisi yrittää korjata madaltamalla eräkokoa.

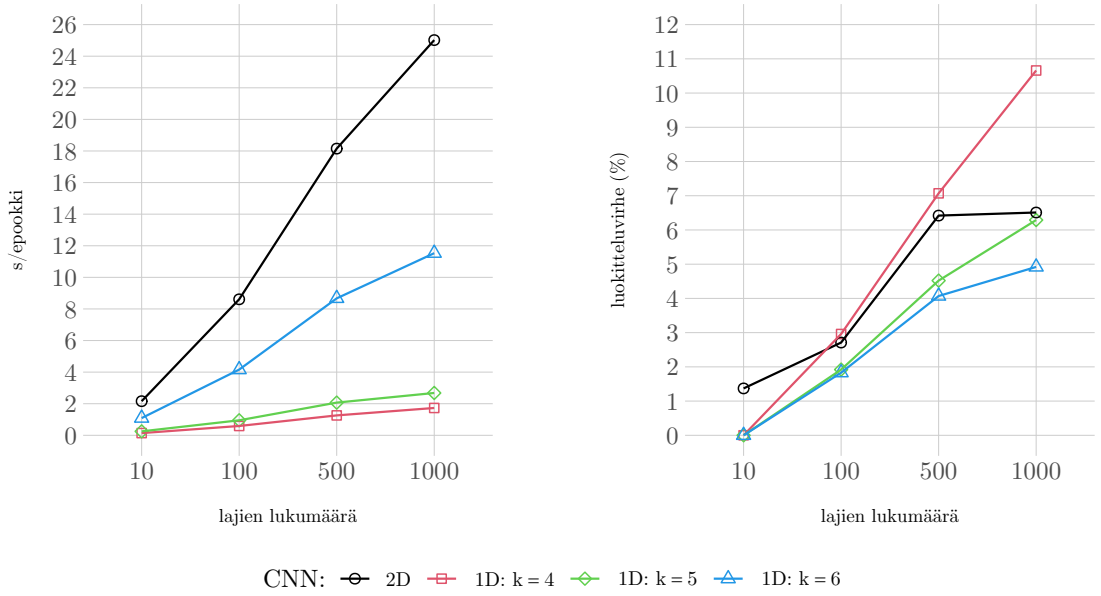
2D-CNN:n luokittelutarkkuudet ovat hieman alhaisempia kuin vastaavat 1D-CNN:n tarkkuudet, mutta kuitenkin samassa suuruusluokassa. Kuitenkin suurempi ero näkyy 2D-CNN:n opetusajassa yhtä epookkia kohti sekä neuroverkon rakenteen monimutkaisuudessa: Jotta saavutetaan edes jokseenkin samankaltainen luokittelutarkkuus 1D-CNN:n kanssa, niin 2D-CNN:n rakenteen pitää olla paljon monimutkaisempi ja opetukseen tarvitaan enemmän aikaa (tai laskentakapasiteettia). Siispä, 2D-CNN ja indikaattorivektoreihin perustuva esikäsittelymenetelmä ei tämän tuloksen perusteella ole kilpailukykyinen luokitin DNA-sekvensseille.

Taulukko 3: *DNA-sekvenssien luokittelun tuloksia 1D- ja 2D-CNN:llä.*

luokkien lkm		10 lajia	100 lajia	500 lajia	1000 lajia
DNA-sekvenssien lkm		1168	4587	9726	13 093
lajikohtaiset havainnot [min; max]		[74; 198]	[23; 198]	[8; 198]	[6; 198]
1D-CNN					
k-mer $k = 4$:	parametrien lkm	694 922	707 236	758 964	823.464
	s/epookki	0.14	0.59	1.26	1.73
	luokittelutarkkuus (%)	100	97.04	92.93	89.34
k-mer $k = 5$:	parametrien lkm	1 710 602	1 722 980	1 774 644	1 839 144
	s/epookki	0.25	0.95	2.07	2.68
	luokittelutarkkuus (%)	100	98.08	95.48	93.71
k-mer $k = 6$:	parametrien lkm	6 756 746	6 769 188	6 820 852	6 885 352
	s/epookki	1.09	4.16	8.67	11.53
	luokittelutarkkuus (%)	100	98.17	95.93	95.08
2D-CNN					
	parametrien lkm	63 307 658	64 450 532	64 860 532	65 373 032
	s/epookki	2.16	8.61	18.15	25.02
	luokittelutarkkuus (%)	98.63	97.29	93.58	93.49

7 Yhteenveto ja pohdintaa

Tässä pro gradu -tutkielmassa tarkasteltiin kahta syväoppimismenetelmää luokitteluongelman ratkaisemiseksi. Menetelmät olivat täysin kytketty neuroverkko ja



Kuvio 16: *1D- ja 2D-CNN:n opetusaineiston opetusajat ja testiaineistolla lasketut luokitteluvirheet 10, 100, 500 ja 1000 lajin tilanteissa.*

konvoluutioneuroverkko, joista jälkimmäisestä esitettiin sovellus DNA-sekvenssien luokittelussa. Vaikka täysin kytketty neuroverkko ja konvoluutioneuroverkko ovat kaksi eri menetelmää, on molemmissa neuroverkoissa myös paljon samankaltaisuuksia. Tämän vuoksi, samoin kuten kirjallisuudessa, ensin esiteltiin huolellisesti täysin kytketty neuroverkko ja sen parametrien estimointiin liittyvät seikat, minkä jälkeen konvoluutioneuroverkko esiteltiin täysin kytketyn neuroverkon pohjalta.

Konvoluutioneuroverkon sovelluksena tutkielmassa esitettiin DNA-sekvenssien luokittelu, jossa hyönteislajeja pyritään tunnistamaan niiden DNA-sekvenssien perusteella. Sovelluksesta johtuen konvoluutioneuroverkkoa tarkasteltiin kahdessa tilanteessa. Ensimmäisessä tilanteessa konvoluutioneuroverkon syöte oli kaksiulotteinen kuva, eli esimerkiksi harmaasävy- tai värikuva. Toisessa tilanteessa syöte oli yksiulotteinen, eli esimerkiksi DNA-sekvenssi. Konvoluutioneuroverkkoja kutsuttiin lyhenteillä 1D-CNN ja 2D-CNN, vastaavasti.

DNA-sekvenssit eivät sellaisenaan olleet sopivia neuroverkkojen syötteiksi, joten luokittelua varten sekvenssit piti esikäsitellä. Tässä tutkielmassa esikäsittelyyn esitettiin kaksi eri tapaa: Ensimmäinen tapa perustui indikaattorivektoreihin ja toinen tapa perustui DNA-sekvenssin perimätunnukseen. Indikaattorivektoreiden mukaisesti tehty esikäsittely johti kaksiulotteiseen CNN:n syötteeseen. Kun esikäsittelyssä käytettiin perimätunnusta, CNN:n syöte pysyi yksiulotteisena. DNA-sekvenssien luokittelu voitiin näin ollen tehdä käyttäen sekä 1D-CNN:ää että 2D-CNN:ää.

Menetelmiä vertaillessa havaittiin, kuinka 2D-CNN vaatii huomattavasti monimutkaisemman neuroverkkorakenteen ja enemmän laskentatehoa haastaakseen 1D-CNN:llä saavutettuja luokittelutarkkuuksia. Vaikka 1D- ja 2D-CNN:n tulosten vertailu ei ollut tutkielman keskeinen tavoite, olisi vertailua varten silti voinut kehittää systemaattisemman tavan. Esimerkiksi aineistojen jako opetus- ja testiaineistoiksi tehtiin erikseen kum-

mallekin konvoluutioneuroverkolle. Tällöin erityisesti testiaineistot sisälsivät eri DNA-sekvenssit menetelmien välillä, vaikkakin testiaineistojen koko oli molemmissa menetelmissä sama. Nyt luokittelutarkkuudessa voi esiintyä ylimääräistä vaihtelua, joka on peräisin siitä, kuuluuko yksittäinen DNA-sekvenssi opetusaineistoon vai ei. Koska 1D- ja 2D-CNN opetettiin onnistuneesti, voidaan olettaa, että molemmat luokittimet yleistyvät hyvin. Tällöin voidaan todeta, että opetusaineistojen eroilla ei liene olleen merkittävää vaikutusta luokittelutarkkuuteen.

Toisen vertailun saisi vertailemalla tämän tutkielman CNN-luokittimilla saatuja tuloksia Roslin et al. (2022) esittämään FinPROTAX-luokittimeen ja sillä saatuihin tuloksiin. Lähtökohtana molempien luokittimien opetuksissa käytettiin samaa aineistoa (PROTAX-aineisto). Tämän tutkielman CNN-luokittimien sovellus oli kuitenkin yksinkertaisempi kuin FinPROTAX-luokittimen alkuperäinen sovellus: Ensiksi, kun tässä tutkielmassa DNA-sekvenssi luokiteltiin vain hyönteisen taksonomian alhaisimman tason (laji) mukaisesti, FinPROTAX kykenee luokittelemaan kerralla useita taksonomian tasoja (artikkelissa tasoja oli seitsemän). Toiseksi, luokittelussa olevien lajien lukumäärä rajattiin tässä tutkielmassa joko 10, 100, 500 tai 1000 lajiin, kun taas artikkelin luokittelussa minkään taksonomian tasojen edustajien lukumääriä ei rajoitettu. Kolmantena, FinPROTAX-luokittimessa yksi mahdollinen luokka DNA-sekvenssille on ”tuntematon”, jonka DNA-sekvenssi voi saada millä tahansa taksonomian tasolla. Tuntematon-luokka annettiin DNA-sekvenssille, jos sekvenssin todennäköisyys johonkin taksonomian tasoista kuulumiselle oli alle 0.5 tai 0.9: Jos ennustettu todennäköisyys taksonomian tasolle kuulumiselle oli korkeampi kuin edelliset rajat (vastaavassa järjestyksessä), niin tällöin mallin ennustamaa luokkaa pidettiin mahdollisena (*plausible*) tai luotettavana (*reliable*). Tämän tutkielman CNN-luokittelijoissa vastaavaa tuntematon-luokkaa ei käytetty ollenkaan, vaan laji luokiteltiin suoraan suurimman todennäköisyyden mukaisesti riippumatta siitä, kuinka suuri (tai pieni) todennäköisyys oli. Toisin sanoen, CNN-luokittelija aina luokitteli DNA-sekvenssin johonkin luokittelussa mukana olleista lajeista. Sellaiset tilanteet, joissa todennäköisyydet antavat aihetta epäillä luokittelun varmuutta, rajattiin tämän tutkielman ulkopuolelle. Luokan määrääminen ”pakotetusti” ei ole aina hyvä ratkaisu esimerkiksi automatisoidussa koneellisessa lajien tunnistuksessa, joten tämän tutkielman 1D- ja 2D-CNN-luokittimia on mahdollista vielä kehittää tässä kohdin.

Neuroverkkojen parametrien estimoinnin yhteydessä käytetty vastavirta-algoritmi esitettiin ensin vain täysin kytketyille neuroverkoille. Tarkastelu rajattiin vain yhteen neuroverkon painoon ja painon päivitykseen kerralla. Tämän jälkeen otettiin käyttöön vektorikeskeinen neuroverkkorakenne, jossa neuroverkon painoja voidaan käsitellä useita samanaikaisesti. Konvoluutioneuroverkon parametrien estimoinnin yhteydessä osoittautui, että vastavirta-algoritmi on hyvin samanlainen myös CNN:n parametrien estimoinnissa. Ainoastaan konvoluutio-operaatioiden kohdalla vastavirta-algoritmia piti muuttaa CNN:lle sopivaksi.

Tutkielman aihetta rajattiin erityisesti neuroverkkojen osalta. Syväoppimismenetelmät ovat olleet muutamien viime vuosien aikana kovassa nousussa, ja niitä tutkitaan paljon. Uusia menetelmiä luodaan jatkuvasti lisää ja olemassaolevia kehitetään eteen-

päin. Tässä tutkielmassa monia asioita piti aiheen laajuuden vuoksi rajata tutkielman ulkopuolelle. Tämä näkyi esimerkiksi luvussa 4, jossa esitettiin neuroverkkojen opettaminen käyttäen stokastista gradienttimenetelmää. SGD:n idea on suhteellisen helppo omaksua, joten sen esittäminen ensimmäisenä (ja ainoana) opetusalgoritmin vaihtoehtona on järkevää asian ymmärtämisen kannalta. SGD on opetusalgoritmien joukossa kuitenkin perusmenetelmä, jolle on olemassa monia muita vaihtoehtoja. Muut, tässä tutkielmassa sivuutetut, vaihtoehdot ovat kehitetty neuroverkon opetusprosessia silmälläpitäen ja ne ratkaisevat monia ongelmia, joita SGD:n kanssa voi esiintyä. Vaihtoehtoisia algoritmeja ovat esimerkiksi vauhdikas SGD (*SGD with momentum*), Adam (*adaptive moment estimation*), AdaGrad (*adaptive gradient algorithm*) ja RMSProp (*root mean squared propagation*), joista voi lukea lisää esimerkiksi julkaisun Goodfellow et al. (2016) luvuista 8.3 ja 8.5 sekä julkaisun Aggarwal (2018) luvusta 3.5. Kaikkien neuroverkon opetusalgoritmien lopullinen päämäärä on joka tapauksessa neuroverkon parametrien päivittäminen virhefunktion minimointiin perustuen.

Konvoluutioneuroverkko on yksi monista mahdollisista DNA-sekvenssien luokittelumenetelmistä. Erityisesti, jos kiinnostuksen kohteena on vain yhden taksonomian tason (esimerkiksi lajin) luokittelu, niin käytännössä mitä tahansa luokittelumenetelmää voitaisiin soveltaa myös DNA-sekvensseille esimerkiksi k-mer-osajonojen mukaista perimätunnusvektoria hyödyntämällä. Perimätunnukset voitaisiin esimerkiksi koota yhdeksi havaintomatriisiksi. Matriisissa on tällöin 5^k muuttujaa ja se sopii syötteeksi moniin sekä syviin että myös pinnallisiin luokittelumenetelmiin. Yksi mielenkiintoinen ja yksinkertainen tämän tutkielman jatkotutkimuksen kohde olisi soveltaa tavallista täysin kytkettyä neuroverkkoa suoraan k-mer-osajonojen perimätunnusvektoriin, ja verrata tuloksia vastaaviin CNN:n tuloksiin. Myös pinnallisten menetelmien, kuten esimerkiksi tukivektorikoneen, käyttö voi yhtä lailla tuottaa hyviä tuloksia.

Yksi haaste, johon Roslin et al. (2022) on jo esittänyt ratkaisun, on useiden taksonomian tasojen luokittelu yhdellä kerralla. Täysin kytketty neuroverkko tai CNN eivät pysty tällaista luokittelua tekemään sellaisenaan, sillä ulostulokerroksella määräytyy aina vain yksi luokka, johon havainnon ennustetaan kuuluvan. Yksinkertainen ratkaisu olisi opettaa useita CNN-luokittimia, joista kukin luokittelisi tietyn taksonomian tason. Toisin sanoen, tässä tutkielmassa esitettyä CNN-luokitinta vastaavia konvoluutioneuroverkkoja opetettaisiin niin monta kappaletta kuin kiinnostuksen kohteena olevia taksonomian tasoja aineisto sisältää. Tässä tavassa on heti näkyvissä kaksi haastetta, joita tarkastellaan seuraavaksi.

Ensiksi, useiden CNN-luokittimien opettaminen alusta loppuun vie luonnollisesti moninkertaisesti aikaa verrattuna vain yhden CNN:n opettamiseen. On kuitenkin olemassa tekniikoita, jotka mahdollistavat saman syväoppimismallin hyödyntämisen monissa eri tarkoituksissa ilman, että opetusprosessia tarvitsee aloittaa jokaisella kerralla alusta. Niin sanotussa opetuksen siirtämisessä (*transfer learning*) neuroverkon tietyt parametrit kiinnitetään ja vain tietyt muut parametrit opetetaan uudestaan uudella aineistolla. Opetuksen siirtäminen on tyypillistä tilanteissa, joissa tutkijan oma tutkimusongelma ratkaistaan ottamalla käyttöön valmiiksi opetettu neuroverkkomalli sen sijaan, että it-

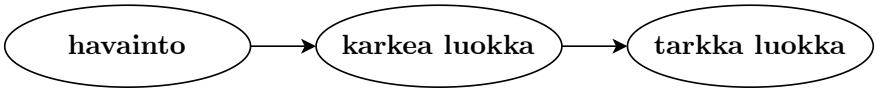
se luotu malli opetettaisiin alusta asti kokonaan omalla aineistolla. Valmiiksi opetetut mallit on usein opetettu niin suurilla opetusaineistoilla, että vain koneoppimismallien opettamista varten suunnitellut tietokoneet saavat mallit opetettua järkevässä ajassa. Esimerkki valmiiksi opetetusta neuroverkkomenetelmästä on kuvien luokittelua varten opetettu ResNet-konvoluutioneuroverkko. Tämä 152 kerrosta sisältävä konvoluutioneuroverkko opetettiin käyttäen ImageNet-aineiston (`https://www.image-net.org/`) eräs-tä versiota, jonka opetusaineisto sisälsi jopa 1.2 miljoonaa 224×224 -kokoista värikuvaa, ja jossa luokkia oli 1000 kappaletta (He et al. 2016). Voidaan hyvin kuvitella, että ResNet-neuroverkon opettaminen ei ole mahdollista kuluttajakäyttöön tarkoitetuilla tietokoneilla järkevässä ajassa. Opetuksen siirtämisen keinoin ResNetin ja muiden valmiiksi opetettujen neuroverkkojen käyttö omassa sovelluksessa on kuitenkin täysin mahdollista myös kotikoneella.

Kun jokin valmiiksi opetettu neuroverkko (esim. ResNet) on opetettu, sitä voidaan hienosäätää (*fine-tuning*) siten, että vain pieni osa parametreista opetetaan uudelleen tutkijan omalla aineistolla. Koska koko neuroverkon parametreista opetetaan uudestaan vain pieni osa, opettaminen on suhteellisen nopeaa ja on täysin tehtävissä tutkijan käytössä olevilla laskentaresursseilla. Hienosäädön jälkeen tutkijan käytössä on lopulta tehokas syväoppimismenetelmä, jonka opettaminen vei vain murto-osan alkuperäisen mallin tarvitsemasta opetusajasta. DNA-sekvenssien taksonomian tasojen luokittelun tapauksessa riittäisi kiinnittää kaikki konvoluutio- ja tiivistyskerrosten parametrit, sillä DNA-sekvenssien piirteiden hakeminen konvoluutiolla pysyy samana taksonomian tasosta riippumatta. Lopulta neuroverkosta riittäisi opettaa uudelleen vain ulostulokerroksen ja lisäksi kenties muutamien aiempien täysin yhdistettyjen kerrosten parametrit. Tämä säästää aikaa, sillä konvoluutiokerrosten parametrit riittää opettaa vain kerran. Opetuksen siirtäminen ja hienosäätö mahdollistaisi useiden CNN-luokittimien opettamisen suhteellisen nopeasti. Lisäksi ne kykenisivät luokittelemaan DNA-sekvenssejä taksonomian eri tasojen luokkiin ilman, että CNN pitäisi opettaa kokonaan alusta alkaen jokaisella kerralla.

Toinen haaste useiden erillisten luokittimien käytössä liittyy taksonomian tasojen keskinäiseen hierarkkisuuteen. Taksonomian tasoissa esiintyy luonnollista hierarkiaa siten, että ylempi taso määrää alempaan tasoon kuuluvien luokkien mahdolliset arvot. Esimerkkinä tarkastellaan kissaa, joka kuuluu kissaeläinten heimoon ja petoeläinten lahkoon. Hierarkiassa lahko on heimon yläpuolella. Tällöin heimon ennuste voi olla kissaeläimet vain silloin, jos lahkon luokaksi on ennustettu petoeläimet. Toisin sanoen, eri taksonomian tasojen luokittelijoiden on oltava keskenään tietoisia siitä, mitä ylempien tasojen ennusteet ovat olleet. Tästä syystä useiden erillisten CNN:ien käyttäminen ei sovellu hierarkkiseen tilanteeseen.

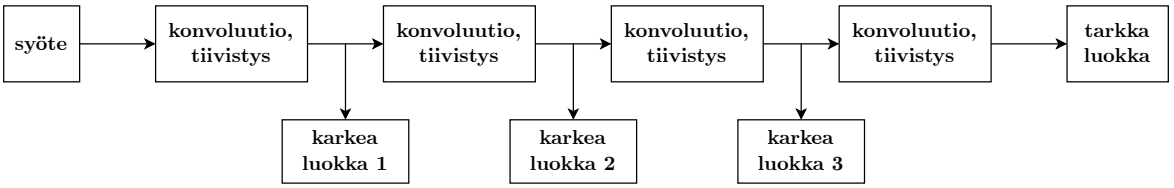
Parempi ratkaisu olisi käyttää sellaisia menetelmiä, jotka on alusta alkaen tarkoitettu hierarkkisten aineistojen luokitteluun. Esimerkiksi PROTAX-luokitin, jonka pohjalta myös FinPROTAX on kehitetty, huomioi taksonomian tasojen hierarkian puurakennetta hyväksikäyttäen: Puun juurisolmusta edetään kohti puun lehtiä, ja lehdet kuvaavat taksonomian alinta tasoa eli eri lajeja ja aiemmat puun solmut vastaavasti ylempiä tak-

sonomian tasoja. (Somervuo et al. 2016.) Myös vastaavaan puurakenteeseen perustuvia konvoluutioneuroverkkoluokittimia on kehitetty hierarkkiseen luokitteluun. Esimerkiksi Yan et al. (2015) esittivät hierarkisen syvän konvoluutioneuroverkon (*hierarchial deep convolutional neural network*, HD-CNN), joka soveltuu hierarkkiseen kuvien luokitteluun. Hierarkian tasojen lukumäärä on tosin rajattu vain kahteen tasoon. HD-CNN luokittelee havainnon ensin karkeaan luokkaan (*coarse category*), joka on hierarkian kahdesta tasosta ylempi. Tämän jälkeen karkeiden luokkien havainnot voidaan edelleen luokitella tarkkoihin luokkiin (*fine category*), joka on vuorostaan hierarkian alempi taso. Kuviossa 17 on havainnollistettu HD-CNN:n hierarkiaa puurakenteena. Organismin taksonomian tasojen luokittelussa HD-CNN soveltuisi siis vain kahden tason luokitteluun.



Kuvio 17: *Puukuvio, kun havainto luokitellaan HD-CNN:llä hierarkisesti kahdelle tasolle. Vasemmanpuoleisin solmu on puun juurisolmu.*

Toinen esimerkki hierarkkisesta CNN-luokittimesta on Seo et al. (2019) esittämä H-CNN (*hierarchial convolutional neural network*), joka laajentaa HD-CNN:n idean sisältämään useita hierarkian tasoja. Perusideana H-CNN:ssä on luokitella havainto useisiin karkeisiin luokkiin samassa konvoluutioneuroverkossa siten, että luokittelu tapahtuu CNN:ssä useissa kohdissa konvoluutiokerrosten jälkeen. Luokittelu tehdään ensimmäiseen karkeaan luokkaan käyttäen vain ensimmäisiä konvoluutiokerroksia, joten tämä luokittelu on nimensä mukaisesti kaikista karkein. Seuraavassa karkeassa luokittelussa konvoluutiokerroksia on mukana enemmän, joten H-CNN on tässä vaiheessa oppinut lisää piirteitä havainnosta ja luokittelu onnistuu paremmin. Aivan H-CNN:n lopussa ulostulokerroksella luokitellaan tarkka luokka. Kuviossa 18 on havainnollistettu H-CNN:n ideaa. H-CNN soveltuisi näin ollen myös organismin taksonomian tasojen luokitteluun, sillä tasoja on usein aina enemmän kuin kaksi.



Kuvio 18: *Hierarkkisen konvoluutioneuroverkon (H-CNN) idea, kun havainto luokitellaan useisiin karkeisiin luokkiin ja lopulta tarkkaan luokkaan.*

Neuroverkon rakenteen muuttamisen lisäksi myös virhefunktiota muokkaamalla voidaan huomioida aineiston hierarkkisuus. Tässä tutkielmassa käytetty ristientropia ei huomioi hierarkkisuutta lainkaan. Eräs yksittäisen hierarkkisen havainnon virhettä kuvaava

funktio on niin sanottu CSL-funktio (*context sensitive loss*), jossa virheeseen vaikuttaa ennustetun ja todellisen havainnon luokan välinen etäisyys hierarkiassa. CSL-funktion arvo kasvaa, jos havainnon luokaksi ennustetaan sellainen luokka, joka on ”kaukana” todellisesta luokasta hierarkian näkökulmasta katsottuna. Jokaiselle havainnolle laskettujen CSL-funktion arvojen perusteella voidaan määritellä myös koko aineiston CSE-virhe (*context sensitive error*), jota käytetään luokitteluvirhettä arvioitaessa varsinaisen luokittelun jälkeen. (Verma et al. 2012.)

Ärje et al. (2020) käyttivät tutkimuksessaan muunneltua CSE-virhettä (LCSE) arvioidessaan hierarkkisen aineiston luokitteluvirhettä. Luokittelussa pohjaeläimiä luokiteltiin konvoluutioneuroverkolla ja muilla menetelmillä hierarkkisesti siten, että jokaiselle pohjaeläinhavainnolle oli määrätty oma hierarkkinen rakenne pohjaeläimen taksonomian tasojen mukaisesti. Aineistona kyseisessä tutkimuksessa käytettiin kuva-aineistoa, joka sisälsi yksilöistä otettuja valokuvia. Näin ollen konvoluutioneuroverkko soveltui erinomaisesti luokittelijaksi myös kyseisessä tutkimuksessa. Kuten tässä tutkielmassa näytettiin, konvoluutioneuroverkko soveltuu hyvin myös DNA-sekvenssien luokitteluun. Tällöin eräs jatkotutkimuskohde tämän tutkielman CNN-luokittimille olisikin tutkia CLS-funktion toimivuutta CNN:n virhefunktiona, kun luokittelu tehdään DNA-sekvenssiaineistolle hierarkkisesti Ärje et al. (2020) esittämällä tavalla.

Viitteet

Kessy Abarenkov, Panu Somervuo, R. Henrik Nilsson, Paul M. Kirk, Tea Huotari, Nerea Abrego ja Otso Ovaskainen (2018). "Protax-fungi: a web-based tool for probabilistic taxonomic placement of fungal internal transcribed spacer sequences". *New Phytologist* 220.2. DOI: 10.1111/nph.15301.

Charu C. Aggarwal (2020). *Linear Algebra and Optimization for Machine Learning*. Springer Nature Switzerland. DOI: 10.1007/978-3-030-40344-7.

Charu C. Aggarwal (2018). *Neural Networks and Deep Learning*. Springer International Publishing. DOI: 10.1007/978-3-319-94463-0.

Dhammika Amaratunga ja Javier Cabrera (2004). *Exploration and Analysis of DNA Microarray and Protein Array Data*. 1. painos. John Wiley & Sons. ISBN: 0-471-27398-8.

Oswald T. Avery, Colin M. MacLeod ja Maclyn McCarty (1944). "Studies on the chemical nature of the substance inducing transformation of pneumococcal types : Induction of transformation by a desoxyribonucleic acid fraction isolated from pneumococcus type III". *Journal of Experimental Medicine* 79.2. DOI: 10.1084/jem.79.2.137.

Xuan Bi, Xiwei Tang, Yubai Yuan, Yanqing Zhang ja Annie Qu (2021). "Tensors in statistics". *Annual Review of Statistics and Its Application* 8.1. DOI: 10.1146/annurev-statistics-042720-020816.

Cameron Buckner (2019). "Deep learning: A philosophical introduction". *Philosophy Compass* 14.10. DOI: 10.1111/phc3.12625.

Athel Cornish-Bowden (1985). "Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984". *Nucleic Acids Research* 13.9. DOI: 10.1093/nar/13.9.3021.

Ralf Dahm (2005). "Friedrich Miescher and the discovery of DNA". *Developmental Biology* 278.2. DOI: 10.1016/j.ydbio.2004.11.028.

Derek Gatherer (2007). "Genome signatures, self-organizing maps and higher order phylogenies: A parametric analysis". *Evolutionary Bioinformatics* 3. DOI: 10.1177/117693430700300001.

Ian Goodfellow, Yoshua Bengio ja Aaron Courville (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org. Viitattu 20.1.2022.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai ja Tsuhan Chen (2018). "Recent

advances in convolutional neural networks". *Pattern Recognition* 77. DOI: 10.1016/j.patcog.2017.10.013.

Trevor Hastie, Robert Tibshirani ja Jerome Friedman (2009). *The Elements of Statistical Learning*. Springer New York. DOI: 10.1007/978-0-387-84858-7.

Kaiming He, Xiangyu Zhang, Shaoqing Ren ja Jian Sun (2016). "Deep residual learning for image recognition". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren ja Jian Sun (2015). "Delving deep into rectifiers: surpassing human-level performance on ImageNet classification". Teoksessa: *2015 IEEE International Conference on Computer Vision (ICCV)*. DOI: 10.1109/ICCV.2015.123.

Paul D. N. Hebert, Alina Cywinska, Shelley L. Ball ja Jeremy R. deWaard (2003). "Biological identifications through DNA barcodes". Teoksessa: *Proceedings of the Royal Society of London. Series B: Biological Sciences*. Vol. 270. 1512. DOI: 10.1098/rspb.2002.2218.

Paul D. N. Hebert ja T. Ryan Gregory (2005). "The promise of DNA barcoding for taxonomy". *Systematic Biology* 54.5. DOI: 10.1080/10635150500354886.

Sergey Ioffe ja Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". DOI: 10.48550/arXiv.1502.03167.

Gareth James, Daniela Witten, Trevor Hastie ja Robert Tibshirani (2021). *An Introduction to Statistical Learning*. Springer US. DOI: 10.1007/978-1-0716-1418-1.

Suomen Lajitietokeskus (2022). *Lajiluettelo 2021*. Luonnontieteellinen keskusmuseo, Helsingin yliopisto, Helsinki. https://urn.fi/URN:ISSN:2490-0907.

Nguyen Quoc Khanh Le, Quang-Thai Ho, Trinh-Trung-Duong Nguyen ja Yu-Yen Ou (2021). "A transformer architecture based on BERT and 2D convolutional neural network to identify DNA enhancers from sequence information". *Briefings in Bioinformatics*. DOI: 10.1093/bib/bbab005.

Yann LeCun, Yoshua Bengio ja Geoffrey Hinton (2015). "Deep learning". *Nature* 521.7553. DOI: 10.1038/nature14539.

Yann LeCun, Léon Bottou, Yoshua Bengio ja Patrick Haffner (1998). "Gradient-based learning applied to document recognition". Teoksessa: *Proceedings of the IEEE*. Vol. 86. 11. DOI: 10.1109/5.726791.

Yann LeCun, Léon Bottou, Genevieve B. Orr ja Klaus-Robert Müller (2012). "Efficient backprop". Teoksessa: *Neural Networks: Tricks of the Trade: Second Edition*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-35289-8_3.

Andrew L. Maas, Awni Y. Hannun ja Andrew Y. Ng (2013). "Rectifier nonlinearities improve neural network acoustic models". Teoksessa: *International Conference on Machine Learning (ICML)*. Vol. 30.

H. Pagès, P. Aboyoun, R. Gentleman ja S. DebRoy (2022). *Biostrings: Efficient Manipulation of Biological Strings*. Versio 2.64.0. https://bioconductor.org/packages/Biostrings.

Phil Picton (1994). "What is a neural network?" Teoksessa: *Introduction to Neural Networks*. Macmillan Education UK. DOI: 10.1007/978-1-349-13530-1_1.

Python Software Foundation (2022). *Python*. Versio 3.9.13. https://www.python.org/.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. Versio 4.2.1. R Foundation for Statistical Computing. Vienna, Austria. https://www.R-project.org/.

Prajit Ramachandran, Barret Zoph ja Quoc V. Le (2017). "Searching for activation functions". DOI: 10.48550/arXiv.1710.05941.

Tomas Roslin, Panu Somervuo, Mikko Pentinsaari, Paul D. N. Hebert, Jireh Agda, Petri Ahlroth, Perttu Anttonen, Jouni Aspi, Gergin Blagoev, Santiago Blanco, Dean Chan, Tom Clayhills, Jeremy deWaard, Stephanie deWaard, Tyler Elliot, Riikka Elo, Sami Haapala, Eero Helve, Jari Ilmonen, Petri Hirvonen, Chris Ho, Juhani Itämies, Vladislav Ivanov, Jevgeni Jakovlev, Aino Juslén, Reijo Jussila, Jere Kahanpää, Lauri Kaila, Jari-Pekka Kaitila, Ari Kakko, Iiro Kakko, Ali Karhu, Sami Karjalainen, Jostein Kjaerandsen, Janne Koskinen, Erkki M. Laasonen, Leena Laasonen, Erkka Laine, Petri Lampila, Valerie Levesque-Beaudin, Liuqiong Lu, Meri Lähteenaro, Pekka Majuri, Sampsa Malmberg, Ramya Manjunath, Petri Martikainen, Jaakko Mattila, Jaclyn McKeown, Petri Metsälä, Margarita Miklasevskaja, Meredith Miller, Renee Miskie, Arto Muinonen, Veli-Matti Mukkala, Suresh Naik, Nadia Nikolova, Kari Nupponen, Otso Ovaskainen, Ika Österblad, Lauri Paasivirta, Timo Pajunen, Petri Parkko, Juho Paukkunen, Ritva Penttinen, Kate Perez, Jaakko Pohjoismäki, Sean Prosser, Martti Raekunnas, Miduna Rahulan, Meeri Rannisto, Sujeevan Ratnasingham, Pekka Raukko, Aki Rinne, Teemu Rintala, Susana Miranda Romo, Jukka Salmela, Juha Salokannel, Riitta Savolainen, Leif Schulman, Pasi Sihvonen, Dina Soliman, Jayme Sones, Claudia Steinke, Gunilla Ståhls, Jukka Tabell, Mikko Tiusanen, Gergely Várkonyi, Eero J. Vesterinen, Esko Viitanen, Veli Vikberg, Matti Viitasaari, Jussi Vilen, Connor Warne, Catherine Wei, Kaj Winqvist, Evgeny Zakharov ja Marko Mutanen (2022).

"A molecular-based identification resource for the arthropods of Finland". *Molecular Ecology Resources* 22.2. DOI: 10.1111/1755-0998.13510.

Yian Seo ja Kyung-shik Shin (2019). "Hierarchical convolutional neural networks for fashion image classification". *Expert Systems with Applications* 116. DOI: 10.1016/j.eswa.2018.09.022.

Panu Somervuo, Sonja Koskela, Juho Pennanen, R. Henrik Nilsson ja Otso Ovaskainen (2016). "Unbiased probabilistic taxonomic classification for DNA barcoding". *Bioinformatics* 32.19. DOI: 10.1093/bioinformatics/btw346.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever ja Ruslan Salakhutdinov (2014). "Dropout: A simple way to prevent neural networks from overfitting". *Journal of Machine Learning Research* 15.56.

Ratnasingham Sujeevan ja Paul Hebert (2007). "BOLD: The barcode of life data system". *Molecular Ecology Notes* 7.3. https://www.boldsystems.org/. Viitattu 21.6.2022.

Ayalew Tefferi (2006). "Genomics basics: DNA structure, gene expression, cloning, genetic mapping, and molecular tests". *Seminars in Cardiothoracic and Vascular Anesthesia* 10.4. DOI: 10.1177/1089253206294343.

TensorFlow Developers (2022). *TensorFlow*. Versio v2.10.0. https://www.tensorflow.org.

James Watson ja Francis Crick (1953). "Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid". *Nature* 171. DOI: 10.1038/171737a0.

Nakul Verma, Dhruv Mahajan, Sundararajan Sellamanickam ja Vinod Nair (2012). "Learning hierarchical similarity metrics". Teoksessa: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2012.6247938.

Zhicheng Yan, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di ja Yizhou Yu (2015). "HD-CNN: Hierarchical deep convolutional neural networks for large scale visual recognition". Teoksessa: *Proceedings of the IEEE International Conference on Computer Vision*.

Cheng-Hong Yang, Kuo-Chuan Wu, Li-Yeh Chuang ja Hsueh-Wei Chang (2022). "DeepBarcoding: Deep learning for species classification using DNA Barcoding". *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 19.4. DOI: 10.1109/tcbb.2021.3056570.

Alireza Zamani, Zdenek Faltýnek Fric, Hugo F. Gante, Tapani Hopkins, Alexander B. Orfinger, Mark D. Scherz, Alena Sucháčková Bartoňová ja Davide Dal Pos (2022). "DNA

barcodes on their own are not enough to describe a species". *Systematic Entomology* 47.3. DOI: 10.1111/syen.12538.

Qingda Zhou, Qingshan Jiang ja Dan Wei (2011). "A new method for classification in DNA sequence". Teoksessa: *2011 6th International Conference on Computer Science & Education (ICCSE)*, s. 218–221. DOI: 10.1109/ICCSE.2011.6028621.

Johanna Ärje, Jenni Raitoharju, Alexandros Iosifidis, Ville Tirronen, Kristian Meissner, Moncef Gabbouj, Serkan Kiranyaz ja Salme Kärkkäinen (2020). "Human experts vs. machines in taxa recognition". *Signal Processing: Image Communication* 87. DOI: 10.1016/j.image.2020.115917.

Kuvat

Kuva 10 (vas.): "Sunflower sky backdrop" by "Fir0002/Flagstaffotos" is licenced under CC BY-NC 3.0. Viitattu 27.6.2022.

Liitteet

A 1D-CNN:n ja 2D-CNN:n rakenteet

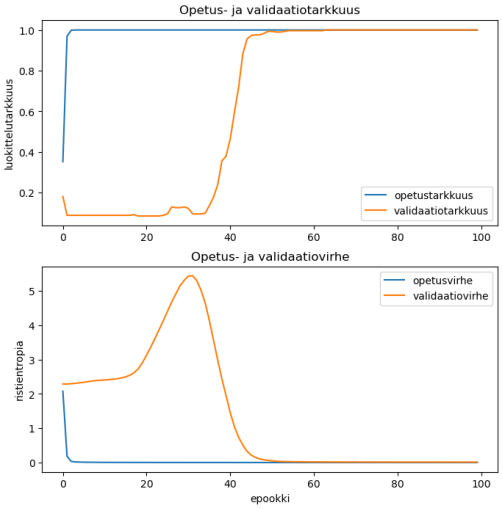
1D-CNN	**2D-CNN**
Conv1D (32)	Conv2D (128)
Conv1D (32)	Conv2D (128)
MaxPool1D	Conv2D (128)
	MaxPool2D
Conv1D (64)	Conv2D (256)
Conv1D (64)	Conv2D (256)
MaxPool1D	Conv2D (256)
	MaxPool2D
Conv1D (128)	Conv2D (512)
Conv1D (128)	Conv2D (512)
MaxPool1D	Conv2D (512)
	MaxPool2D
Conv1D (256)	
Conv1D (256)	
MaxPool1D	
Tasoitus	Tasoitus
Täysin kytketty kerros (128)	Täysin kytketty kerros (1024)
Täysin kytketty kerros (128)	Täysin kytketty kerros (1024)
Täysin kytketty kerros (#lajit)	Täysin kytketty kerros (#lajit)

Kuvio 19: *Valittujen yksi- ja kaksiulotteisten konvoluutioneuroverkkojen rakenteet. Kuviossa Conv1D ja Conv2D sekä MaxPool1D ja MaxPool2D tarkoittavat 1D- ja 2D-konvoluutioita ja -tiivistyksiä, vastaavasti. Suluissa on ilmoitettu konvoluutiokerroksen ydinten lukumäärä ja täysin yhdistetyn kerroksen neuronien lukumäärä.*

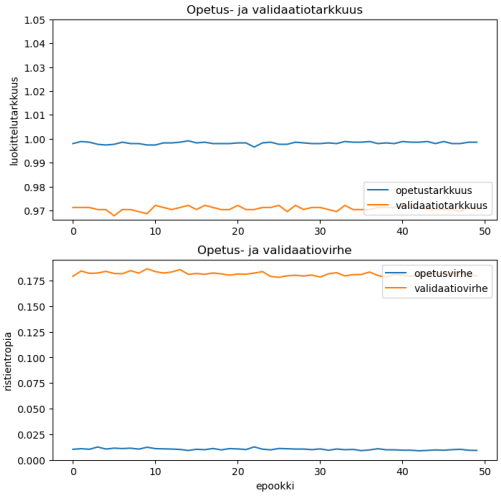
B Opetuksen aikaiset ristientropian ja luokittelutarkkuuden kuvaajat

Alla on kaikkien CNN-luokittimien opetuksen aikana opetus- ja validointiaineistolla lasketut ristientropian ja luokittelutarkkuuden kuvaajat epookkien lukumäärän suhteen. Ristientropian kuvaajista tarkasteltiin mallin ylioppimista.

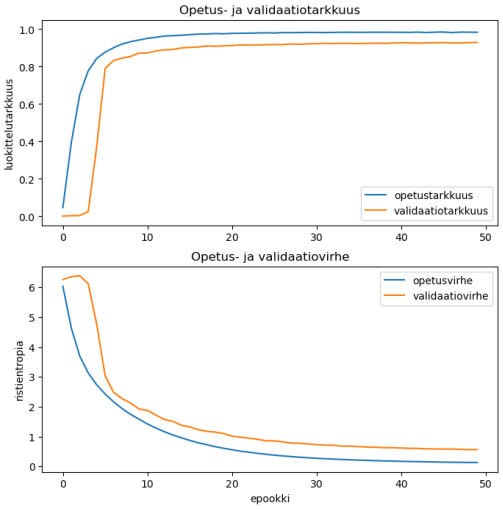
1D-CNN, $k = 4$, 10 lajia



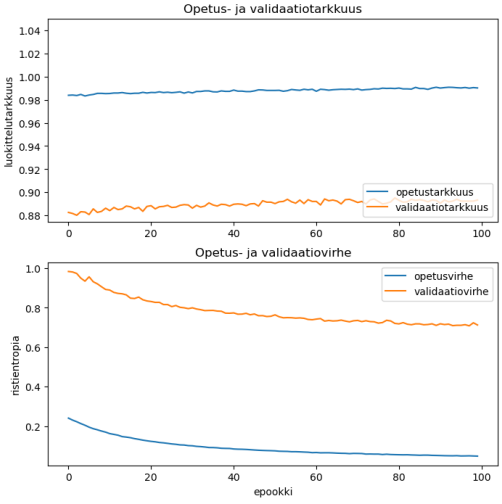
1D-CNN, $k = 4$, 100 lajia



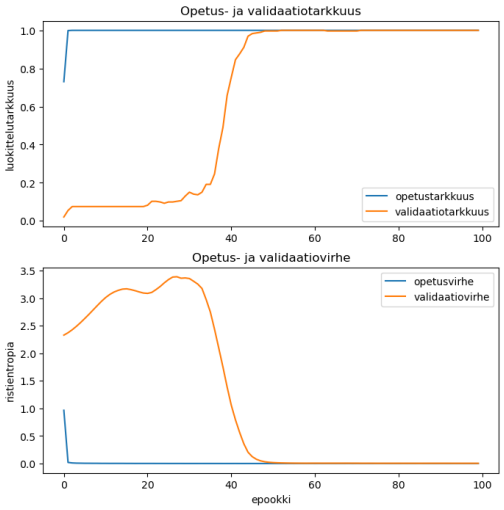
1D-CNN, $k = 4$, 500 lajia



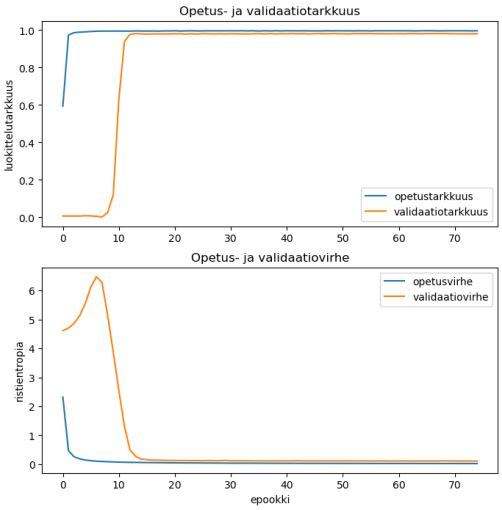
1D-CNN, $k = 4$, 1000 lajia



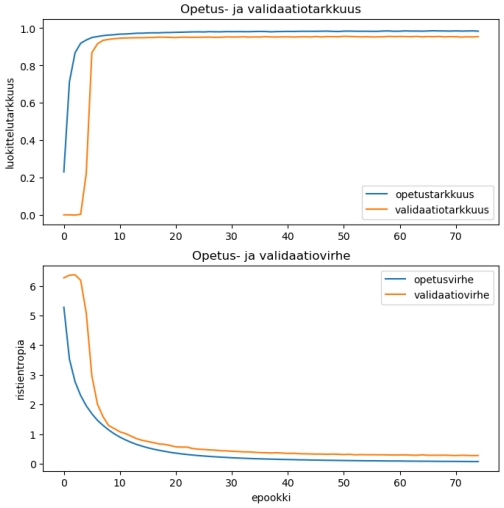
1D-CNN, $k = 5$, 10 lajia



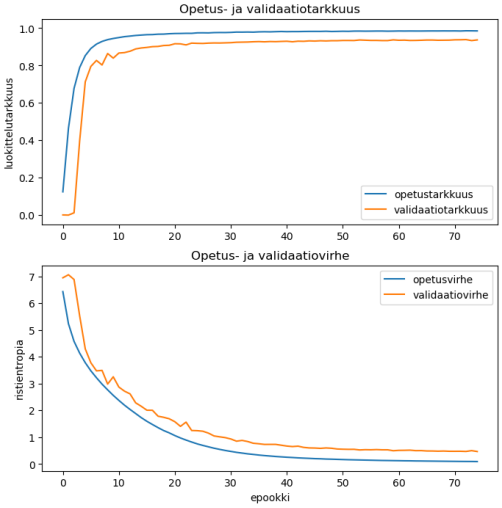
1D-CNN, $k = 5$, 100 lajia



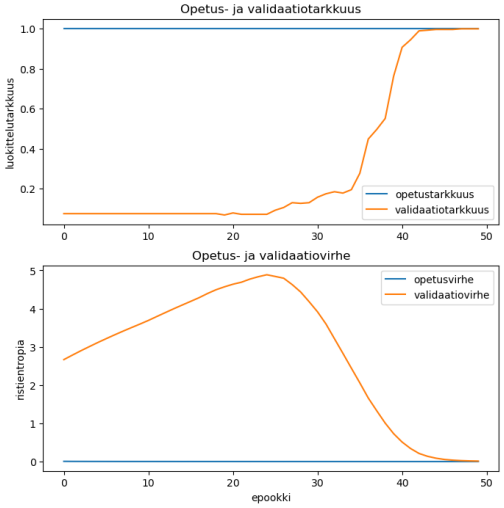
1D-CNN, $k = 5$, 500 lajia



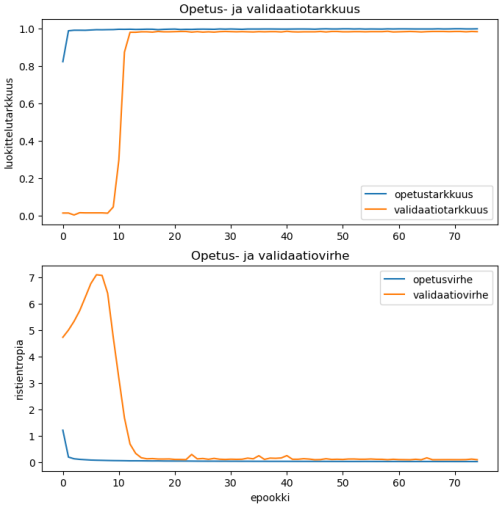
1D-CNN, $k = 5$, 1000 lajia



1D-CNN, $k = 6$, 10 lajia

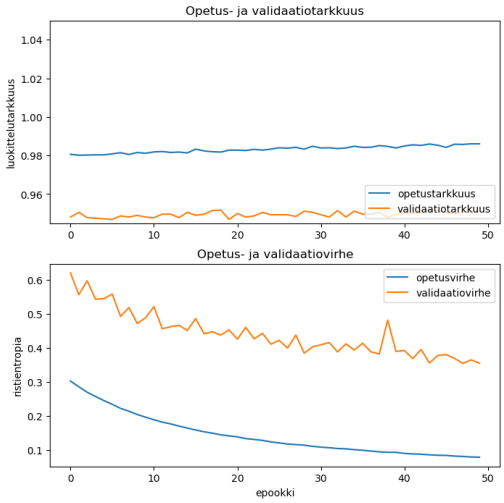


1D-CNN, $k = 6$, 100 lajia

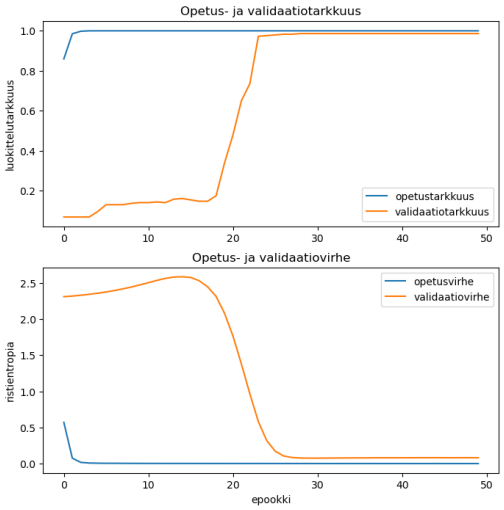


1D-CNN, $k = 6$, 500 lajia

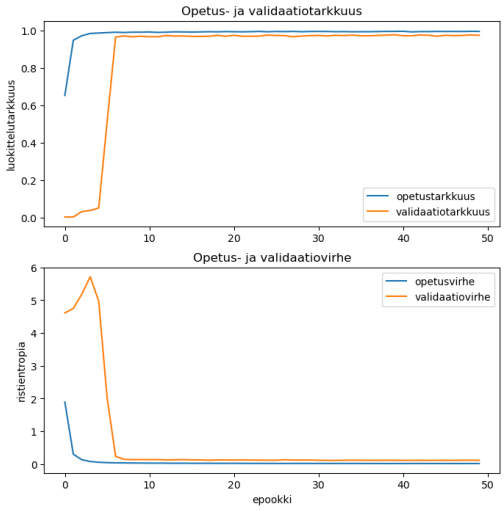
1D-CNN, $k = 6$, 1000 lajia



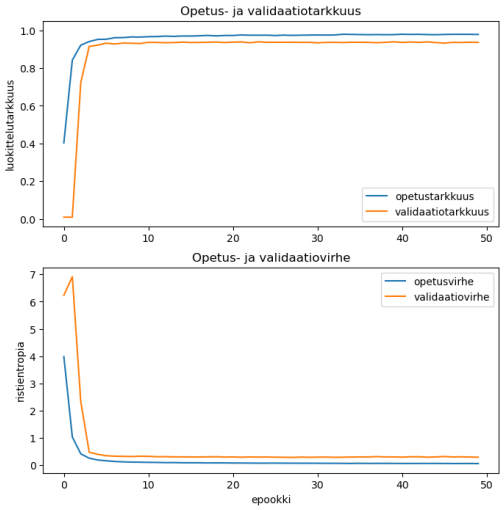
2D-CNN, 10 lajia



2D-CNN, 100 lajia



2D-CNN, 500 lajia



2D-CNN, 1000 lajia