Koneoppiminen molekyylidynamiikassa

LuK-tutkielma
Turun yliopisto
Fysiikka
2025
Erkki Rasmus
Tarkastaja:
Dosentti Johannes Niskanen

TURUN YLIOPISTO
Fysiikan laitos

Rasmus, Erkki Koneoppiminen molekyylidynamiikassa

LuK-tutkielma, 26 s., 3 liites.
Fysiikka
Huhtikuu 2025

Tutkielmassa tutustutaan koneoppimismenetelmien hyödyntämiseen molekyylidynamiikan simulaatioissa. Tässä rajoitutaan molekyyleihin kohdistuvien voimien ennustamiseen koneoppimisen menetelmien avulla. Tätä varten esitellään molekyylidynamiikan simulaatioiden perusperiaate yleisellä tasolla. Lisäksi tarkastellaan koneoppimisen perusteita, jolla pyritään antamaan yleiskuva siitä, mitä koneoppimismallin kouluttaminen tarkoittaa. Tässä rajoitutaan molekyylidynamiikan sovelluksissa laajasti käytettäviin neuroverkkoihin.

Tutkielman lopussa perehdytään voimien ennustamiseen. Tarkoitus on antaa yleiskuva tämän esittämisestä koneoppimistehtävänä. Lisäksi tarkastellaan ennustamisessa käytettävän koneoppimismallin kouluttamista. Lopussa esitellään tarkemmin rekursiiviset graafineuroverkot, jotka ovat yksi molekyylidynamiikan lupaavimmista koneoppimismenetelmistä. Tähän perustuvien koneopittujen voimien toimintaa käytännössä demonstroitiin ajamalla molekyylidynamiikan simulaatioita useissa eri lämpötilassa.

Asiasanat: koneoppiminen, neuroverkko, molekyylidynamiikka, emulaattori, koneoppimispotentiaali

Sisällys

Johdanto

Viime vuosikymmenenä nähtiin merkittävää kasvua koneoppimiseen perustuvien teknologioiden käytössä, mikä on herättänyt kiinnostusta tieteilijöiden parissa [1]. Yksi kiinnostava sovelluskohde on tekoälyn käyttäminen emulaattorien luomisessa nopeuttamaan simulaatioita [2]. Tällöin koneoppimiseen perustuva emulaattori koulutetaan syöttämällä sille simulaation lähtöarvoja ja näitä vastaavia lopputuloksia, joista se pyrkii etsimään datan taustalla olevia malleja [2]. Etenkin neuroverkkoihin perustuvat koneoppimissysteemit ovat osoittautuneet lupaaviksi emulaattoreiksi, koska ne pystyvät tekemään tarkkoja approksimaatiota selvästi aiempaa vähäisemmällä datamäärällä [2].

Tutkielmassa perehdytään koneopittuihin potentiaalienergiapintoihin, jotka ovat esimerkki emulaattorien käytöstä molekyylidynamiikan simulaatioissa. Koneoppimispotentiaaleilla pystytään approksimoimaan elektronirakennemenetelmistä saatavia potentiaalienergioita. Tämä mahdollistaa potentiaalienergioiden ja voimien laskemisen lähes alkuperäisen menetelmän tarkkuudella, mutta paljon pienemmällä laskennallisella vaativuudella. Tällä pystytään nopeuttamaan huomattavasti molekyylidynamiikan simulaatioita, jotka edellyttävät voimien ratkaisemisen jokaisella aika-askeleella. Tämä mahdollistaa aiempaa suurempien ja pidempien simulaatioiden ajamisen, mikä lisää molekyylidynamiikan simulaatioiden mahdollisia sovelluskohteita. [3, 4]

1 Molekyylidynamiikka

Molekyylidynamiikassa tarkastellaan keskenään vuorovaikuttavien molekyylien dynamiikkaa [5]. Molekyylidynamiikan simulaatiot ovat nykyään arvokas apuväline monilla eri aloilla, sillä ne mahdollistavat molekyylien ja kiinteiden aineiden ominaisuuksien ja reaktioiden ennustamisen atomitason rakenteellisen ja dynaamisen

informaation avulla. [4, 6]. Simulaatioissa pyritään mallintamaan molekyylien liikeratoja lähtien annetuista alkuehdoista [6]. Käytännössä tämä edellyttää voimien ja kiihtyvyyksien ratkaisemista simulaation jokaisella aika-askeleella, sillä kiihtyvyyksien avulla liikeratoja propagoidaan liikeyhtälöiden mukaisesti [5, 6].

1.1 Molekyylidynamiikan simulaatiot

Liikeyhtälöt ovat kappaleen liikettä kuvaavia differentiaaliyhtälöitä, joista kappaleen liikerata voidaan ratkaista. Karteesisissa koordinaateissa klassinen liikeyhtälö voidaan kirjoittaa muodossa

$$\boldsymbol{a}_i = \frac{\boldsymbol{f}_i}{m_i}, \tag{1}$$

missä $\boldsymbol{f}_i$ on kappaleeseen i vaikuttava voima, m_i on kappaleen massa ja $\boldsymbol{a}_i$ on sen kiihtyvyys. Kappaleeseen vaikuttava voima saadaan puolestaan systeemin potentiaalienergian gradientista ydinten paikkojen suhteen kaavalla

$$\boldsymbol{f}_i = -\nabla_{\boldsymbol{r}_i} V(\boldsymbol{r}_1, \boldsymbol{r}_2, ...), \tag{2}$$

missä V on potentiaalienergiapinnan arvo kyseisellä konfiguraatiolla, $\boldsymbol{r}_i$ kappaleen i paikkavektori ja $\nabla_{\boldsymbol{r}_i}$ gradientti tässä pisteessä. [5]

Molekyylidynamiikan simulaatioissa liikeyhtälöitä ei ratkaista analyyttisesti, vaan liikeradan propagoinnissa käytettävät algoritmit hyödyntävät approksimatiivisia menetelmiä. Molekyylidynamiikan simulaatioissa yleisesti käytettävässä differenssimenetelmässä approksimoidaan liikeyhtälöiden ratkaisuja lyhyillä aika-askeleilla. Menetelmän ideana on approksimoida (jatkuvaa) klassista liikerataa Taylorin polynomin avulla. Tällöin paikkojen, nopeuksien ja kiihtyvyyksien uudet arvot lyhyen aika-askeleen δt päässä on mahdollista laskea niiden nykyisten arvojen avulla. [5]

Uusien paikkojen avulla voidaan edelleen laskea uusi potentiaali parametrisoidusta funktiosta tai jopa elektronien Schrödingerin yhtälöstä käyttäen esimerkiksi tiheysfunktionaaliteoriaa (engl. density functional theory) [5, 6]. Potentiaalista saadaan uudet voimat kaavan (2) avulla ja voimista uudet kiihtyvyydet kaavan (1)

avulla [5]. Tällä tavoin liikeratoja voidaan approksimoida iteratiivisesti toistamalla edellä kuvattuja vaiheita [5].

Molekyylidynamiikan simulaatioissa käytettävällä algoritmilla pyritään siis toisintamaan klassista liikerataa mahdollisimman tarkasti riittävän pieniä aika-askelia käyttäen. Approksimaatiokyvyn lisäksi käytettävälle algoritmille on muitakin vaatimuksia, kuten aikareversiibelit liikeradat sekä systeemin kokonaisliikemäärän ja -energian säilyminen. Näiden ehtojen tärkeys seuraa siitä, että ne määräävät (mikrokanonisen) systeemin mahdolliset tilat. Näin ollen nämä ehdot toteuttavan simulaation jokainen vaihe on fysikaalisesti mahdollinen annetuilla alkuehdoilla. Tämä on tarpeellista, jos simulaatiolla halutaan generoida systeemin mahdollisia tiloja. [5]

1.2 Velocity Verlet -algoritmi

Tässä osiossa esitellään velocity Verlet, joka on eräs molekyylidynamiikan simulaatioissa käytettävä algoritmi. Siinä lasketaan ensin kullekin hiukkaselle i paikan uusi arvo kaavalla

$$\boldsymbol{r}_i(t+\delta t) = \boldsymbol{r}_i(t) + \delta t \boldsymbol{v}_i(t) + \frac{1}{2}\delta t^2 \boldsymbol{a}_i(t), \tag{3}$$

missä δt vastaa aika-askeleen pituutta. Samalla lasketaan vanhojen arvojen avulla nopeus puolivälissä aika-askelta kaavalla

$$\boldsymbol{v}_i(t+\frac{1}{2}\delta t) = \boldsymbol{v}_i(t) + \frac{1}{2}\delta t \boldsymbol{a}_i(t), \tag{4}$$

Uuden paikan arvon avulla saadaan ratkaistua edellisessä osiossa kuvatulla tavalla uusi voima ja kiihtyvyys $\boldsymbol{a}_i(t+\delta t)$, jota käytetään aiemmin lasketun $\boldsymbol{v}_i(t+\frac{1}{2}\delta t)$:n kanssa nopeuden uuden arvon $\boldsymbol{v}_i(t+\delta t)$ laskemisessa:

$$\boldsymbol{v}_i(t+\delta t) = \boldsymbol{v}_i(t+\frac{1}{2}\delta t) + \frac{1}{2}\delta t \boldsymbol{a}_i(t+\delta t). \tag{5}$$

Tämä jälkeen paikan, nopeuden ja kiihtyvyyden arvot on päivitetty ja algoritmin suoritus alkaa alusta. Algoritmista on syytä huomata, että nopeuden laskeminen

tehdään kahdessa osassa, joiden välissä lasketaan kiihtyvyyden uusi arvo. Kaavat (4) ja (5) yhdistämällä uusi nopeus $\boldsymbol{v}_i(t + \delta t)$ saadaan muotoon

$$\boldsymbol{v}_i(t + \delta t) = \boldsymbol{v}_i(t) + \frac{1}{2}\delta t[\boldsymbol{a}_i(t) + \boldsymbol{a}_i(t + \delta t)]. \tag{6}$$

Syy nopeuden laskemiseen kahdessa osassa on muistin käytön optimointi. Nopeuden laskeminen puolivälissä mahdollistaa kaavassa (6) esiintyvien nopeuden ja kiihtyvyyden vanhojen arvojen tallentamisen yhteen muuttujaan ennen kiihtyvyyden ylikirjoittamista. [5, 7]

2 Koneoppiminen

Koneoppimisen klassisen määritelmän esitti Arthur Samuel vuonna 1959: "Koneoppiminen on tutkimusala, jossa tietokoneet voivat oppia ilman, että ne on eksplisiittisesti ohjelmoitu"[8, 9]. Tällä viitataan siihen, että koneoppimisessa on tavoitteena kehittää algoritmeja, jotka pystyvät oppimaan datasta automaattisesti. Tällöin algoritmi pyrkii tunnistamaan datan taustalla olevia rakenteita sekä edelleen tekemään datan pohjalta ennustuksia uusia havaintoja koskien. Koneoppiminen voidaan siis nähdä pohjimmiltaan ennustusongelmana (engl. prediction problem), jossa tavoitteena on tehdä yleistyksiä datan pohjalta. [10]

Tässä luvussa käydään läpi koneoppimisen taustalla olevaa teoriaa yleisellä tasolla. Tarkastelussa rajoitutaan ohjattuun oppimiseen (engl. supervised learning) ja koneoppimismenetelmistä perehdytään erityisesti neuroverkkoihin sekä edelleen graafineuroverkkoihin, sillä nämä menetelmät ovat keskeisiä tutkielman kannalta.

2.1 Koneoppimisen vaiheet ja menetelmät

2.1.1 Koneoppimisen matemaattinen viitekehys

Tässä tutkielmassa keskityn ohjattuun oppimiseen, jossa koulutusdata on valmiiksi järjestetty ja koostuu pareista, joissa on selittävät muuttujat ja näitä vastaa-

vat selitettävät muuttujat. Tällöin koulutusdata D voidaan ilmaista matriisiparina $D = (\boldsymbol{X}, \boldsymbol{Y})$, jossa matriisi $\boldsymbol{X}$ sisältää riippumattomien muuttujien arvot ja matriisi $\boldsymbol{Y}$ näitä vastaavien riippuvien muuttujien arvot. Lisäksi rajoitun mallipohjaiseen oppimiseen, jolloin tehtävänä on löytää malli $f(\boldsymbol{x}; \boldsymbol{\theta})$, joka tuottaa mahdollisimman hyvän ennusteen $\hat{\boldsymbol{y}}$. Näin ollen malli f on funktio $f : \boldsymbol{x} \rightarrow \hat{\boldsymbol{y}}$. Ennusteen hyvyyttä mitataan tappiofunktiolla (loss function) $C(\boldsymbol{Y}, f(\boldsymbol{X}; \boldsymbol{\theta}))$, joka määritellään koneoppimistehtävän yhteydessä. Parhaan mallin etsiminen voidaan nähdä optimointitehtävänä, jossa etsitään sakkofunktion minivoivia mallin parametreja $\hat{\boldsymbol{\theta}} = \text{argmin}_{\boldsymbol{\theta}}\{C(\boldsymbol{Y}, f(\boldsymbol{X}; \boldsymbol{\theta}))\}$. Edellä esitellyt notaatiot vastaavat pääosin artikkelissa *A high-bias,low-variance introduction to Machine Learning for physicists* [10] käytettyjä merkintätapoja pienillä muutoksilla.

2.1.2 Data

Data on keskeisessä roolissa koneoppimisprosessissa, sillä sitä tarvitaan mallin kouluttamiseen [9, 11]. Koneoppiminen vaati yleisesti paljon dataa, mutta lisäksi käytettävän datan on oltava riittävän laadukasta ja edustavaa, sillä se vaikuttaa opitun mallin sisäiseen tarkkuuteen kohinan (engl. noise) kautta ja ulkoiseen tarkkuuteen mahdollisten vinoumien (engl. sampling bias) kautta [9, 10]. Vinoutunut koulutusdata ei ole edustava otos siitä joukosta, johon mallia pyritään yleistämään [9]. Jos koulutusdata on vinoutunutta, ei myöskään sen avulla koulutettu malli todennäköisesti anna hyviä ennusteita, vaan omaksuu koulutusdatan vinouman [9, 11]. Vinouma voi aiheutua joko virheellisestä otantatavasta (sampling bias) tai otantavirheestä (sampling noise), joka aiheutuu satunnaisuudesta [9].

Myös vinoutumaton data voi olla huonolaatuista, jos siinä on kohinaa, virheitä, poikkeavia havaintoja tai huonoja selittäviä muuttujia [9]. Tämän takia virheet ja poikkeavat havainnot kannattaa poistaa datasta tai korjata ennen koulutusprosessin aloittamista [9]. Tätä kutsutaan datan siivoamiseksi (data cleaning), mikä paran-

taa oppimisprosessia [9]. Myös selittäviä muuttujia voi poistaa, lisätä ja yhdistellä, jotta koulutusdata sisältäisi mallin kannalta oleelliset muuttujat [9]. Lisäksi ennen koulutusprosessia selittävät muuttujat kannattaa standardisoida [10].

2.1.3 Mallinvalinnan teoriaa

Koneoppimismallin kouluttaminen voidaan nähdä ennustusongelmana, jolloin kouluttamisella tarkoitetaan ennustamisessa tehtävän yleistysvirheen (engl. generalization error) minimoimista. Tätä varten kouluttamiseen käytettävä data jaetaan usein koulutusdataan, jota käytetään koneoppimismallin kouluttamiseen, sekä validaatiodataan, jota käytetään mallin yleistettävyyden testaamiseen. [9, 10]

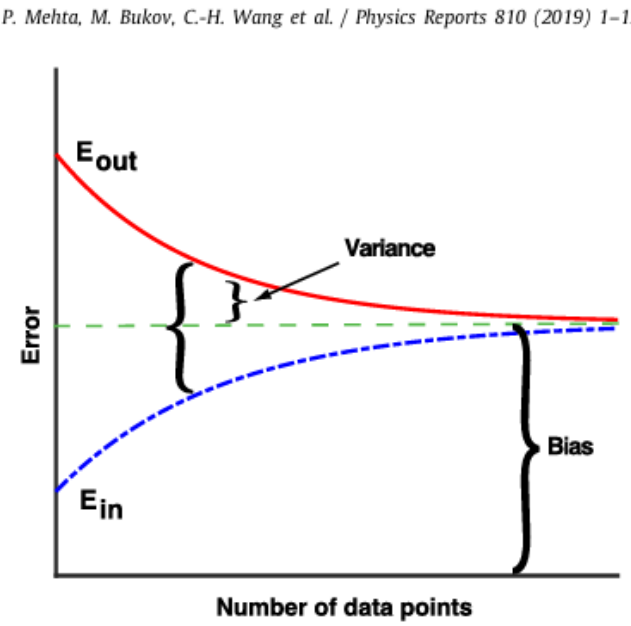
Jako suoritetaan, koska tyypillisesti mallin sisäinen virhe E_{in} (in-sample error) on pienempi kuin yleistysvirhe E_{out}. Sisäisellä virheellä tarkoitetaan mallin virhettä koulutusdatalle ja yleistysvirhellä puolestaan mallin virhettä validaatiodatalle. Näiden kahden virheen erotus riippuu koulutusdatan koosta ja on tyypillisesti suurempi datamäärän ollessa pieni. Datamäärän lähestyessä ääretöntä, lähestyvät virheet asympioottisesti toisiaan, mitä voidaan havainnollistaa kuvaajalla, jota kutsutaan oppimiskäyräksi (engl. learning curve). [9, 10]

Idealisoitu oppimiskäyrä on esitetty kuvassa 1, jossa on sisäinen virhe E_{in} ja yleistysvirhe E_{out} datapisteiden määrän funktiona. Kuva havainnollistaa virheiden välisen erotuksen merkitystä, sillä kuvassa näkyvä varianssi johtuu otoksen äärellisestä koostaa aiheutuvasta tilastollisesta kohinasta, joka pienenee otoskoon kasvaessa. Arvoa, jota virheet lähestyvä kutsutaan mallin vinoumaksi (engl. model bias), joka vastaa parasta teoreettista oppimistulosta, mihin kyseisellä mallilla voi päästä. [10]

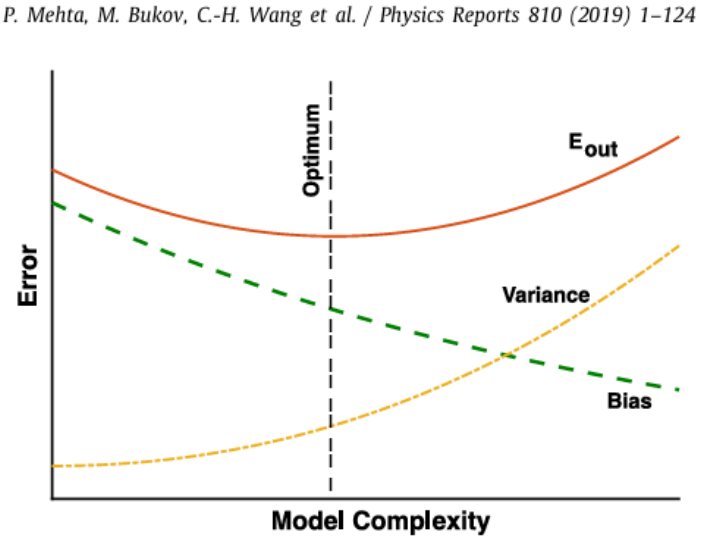
Kuvassa 2 on esitetty mallin paras teoreettinen yleistysvirhe E_{out} mallin kompleksisuuden funktiona jollakin äärellisellä datapisteiden määrällä [10]. Kuva havainnollistaa koneoppimismallin kouluttamisen kannalta tärkeää tulosta, jonka mukaises-

ti oppimistulokset eivät automaattisesti parane mallin kompleksisuuden kasvaessa, vaan on olemassa optimaalinen mallin kompleksisuuden arvo, jolla saavutetaan paras mahdollinen oppimistulos eli pienin mahdollinen yleistysvirhe [10]. Tämä selittyy sillä, että mallin varianssi lisääntyy kompleksisuuden kasvaessa, koska malli alkaa tunnistamaan myös koulutusdatassa esiintyvää tilastollista kohinaa, joka liittyy otantaprosessin satunnaisuuteen, eikä populaatioon todellisiin ominaisuuksiin [9]. Tällöin voi esiintyä ylisovittamista, jossa malli on liian monimutkainen suhteessa koulutusdatan datapisteiden määrään [9, 11]. Toisaalta jos mallia yksinkertaistetaan liikaa, voi ilmetä alisovittamista, jolloin käytettävä malli ei enää kykene kuvaamaan datan taustalla olevia rakenteita [9, 11].

Kuvan 2 havainnollistamaa tulosta kutsutaan vinouman ja varianssin kompromissiksi (engl. bias-variance tradeoff), sillä paras oppimistulos saavutetaan, kun otetaan molempien aiheuttamat virheet huomioon ja pyritään minimoimaan näiden virheiden summaa. On tärkeää tiedostaa, että näiden lisäksi käytännön sovelluksissa virhettä aiheuttaa myös datan itsensä sisältämä kohina, joka kasvattaa virhettä teoreettisesta minimistä. [9, 10]



Kuva 1: Oppimiskäyrä, joka havainnollistaa vinouman ja varianssin kompromissia. Kuva on lainattu artikkelista *"A high-bias, low-variance introduction to Machine Learning for physicists"*. [10]



Kuva 2: Yleistysvirhe mallin kompleksisuuden funktiona, kun datan koko pidetään vakiona. Kuva on lainattu artikkelista *"A high-bias, low-variance introduction to Machine Learning for physicists"*. [10]

2.1.4 Koulutus

Mallin kouluttaminen tarkoittaa optimointitehtävää, jossa tarkoituksena on löytää parametrit, jotka minimoivat suoritusta mittaavan sakkofunktion. Optimaalisten parametrien löytämiseen käytetään optimointialgoritmeja, joista esitellään yleisesti käytetty gradienttimenetelmä (engl. gradient descent) sekä sen variaatio stokastinen gradienttimenetelmä (engl. stochastic gradient descent). Gradienttimenetelmässä sakkofunktiolle lasketaan ensin gradientti parametrivektorin suhteen: $\nabla_{\boldsymbol{\theta}} C(\boldsymbol{Y}, f(\boldsymbol{X}; \boldsymbol{\theta}))$. [9]

Tämä kertoo moniulotteisessa parametriavaruudessa suunnan, jossa sakkofunktion arvo kasvaa nopeiten [9, 10]. Gradienttiin nähden päinvastaisessa suunnassa puolestaan funktion arvo pienenee nopeiten, joten sakkofunktion minimoimiseksi parametreja muutetaan liikkumalla gradienttivektoriin nähden päinvastaiseen suuntaan [9]. Tämä vastaa algoritmin seuraavaa vaihetta: $\boldsymbol{\theta}^{next} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} C(\boldsymbol{Y}, f(\boldsymbol{X}; \boldsymbol{\theta}))$, missä η on koulutusnopeus (engl. learning rate), joka on algoritmin hyperparametri [9]. Koulutusnopeus säätelee sitä, kuinka suuria askeleita gradientin suuntaan otetaan ja sillä on keskeinen rooli oppimistuloksen kannalta [9].

Koulutusnopeus vaikuttaa sakkofunktion ja samalla myös parametrien arvojen suppenemiseen, kun iteroidaan algoritmin mukaisesti. Jos oppimisaste on liian pieni, algoritmilla kestää kauan supeta sakkofunktion minimiin. Toisaalta liian suurella oppimisasteella, iteraatioiden jono ei välttämättä suppene minimiin, vaan se "hyppää"paikallisen minimin yli ja voi jopa hajaantua. Tämän vuoksi sopivan oppimisasteen valinta on keskeinen osa koulutusprosessia. [9]

Koulutusnopeutta ei yleensä pidetä vakiona koko prosessin ajan, vaan sitä pienennetään iteroinnin edetessä. Ideana on, että alussa otetaan suurempia askelia minimin suuntaan, jotta suppeneminen olisi nopeampaa. Puolestaan minimiä lähestyttäessä askeleet pienenevät, mikä auttaa funktiota suppenemaan minimiin. Oppimisastetta säädellään oppimisaikataululla (engl. learning schedule), joka kuvaa mi-

ten sitä muutetaan iteraation edetessä. Hyvin valittu oppimisaikataulu toimii edellä kuvatun idean mukaisesti, jolloin oppiminen on nopeampaa ja sillä päästään lähemmäksi minimiä. [9, 10]

Kuten edellisissä kappaleissa todettiin, ovat koneoppimisessa tarvittavat datamäärät usein suuria, jolloin gradientin laskeminen koko harjoitusdatan suhteen ei ole järkevää. Tämän vuoksi usein käytetään stokastista gradienttimenetelmää, jossa algoritmi toimii samalla idealla kuin tavallisessa gradienttimenetelmässä. Gradienttia ei kuitenkaan lasketa koko koulutusdatan yli, vaan koulutusdatasta satunnaisesti poimitun otoksen suhteen, jolloin funktio suppenee keskimääräisesti kohti minimiä. [9]

Kun mallia lähdetään kouluttamaan, tulee huomioida myös aiemmassa osiossa mainittu ylisovittaminen. Ylisovittamisen välttämiseksi mallien kouluttamisessa käytetään hyperparametreja (engl. hyperparameters), jotka lisätään käytettävään koulutusalgoritmiin. Tyypillisesti tämä tarkoittaa sitä, että koulutuksessa käytettävään sakkofunktioon lisätään regularisaatiotermi (engl. regularization term), joka rankaisee mallia liian suurista parametrien arvoista ja täten pienentää parametrien arvoja ehkäisten ylisovittamista. [9]

Hyperparametrien valinnassa hyödynnetään usein ristiinvalidointia. Tällöin koulutusdata jaetaan satunnaisesti koulutusdataan ja testidataan. Näitä yhdistelmiä vaihdellaan ja jokaisella kerralla malli koulutetaan koulutusdatana olevalla joukolla. Jäljelle jäävällä osalla testataan mallin yleistymistä, jota puolestaan käytetään hyperparametrien valitsemiseen. Lopulta malli validoidaan testaamalla sen virhettä varsinaisella testidatalla. [9]

2.2 Neuroverkot

Syvät neuroverkot ovat yksi ohjatun oppimisen käytetyimmistä ja tehokkaimmista koneoppimismenetelmistä [10]. Onkin osoitettu, että neuroverkot ovat universaaleja

approksimaattoreita eli ne pystyvät approksimoimaan mitä tahansa mielivaltaista jatkuvaa funktiota [12]. Vaikka tämä onkin puhtaasti teoreettinen tulos, kuvaa se neuroverkkojen esitysvoimaa approksimaattoreina [10].

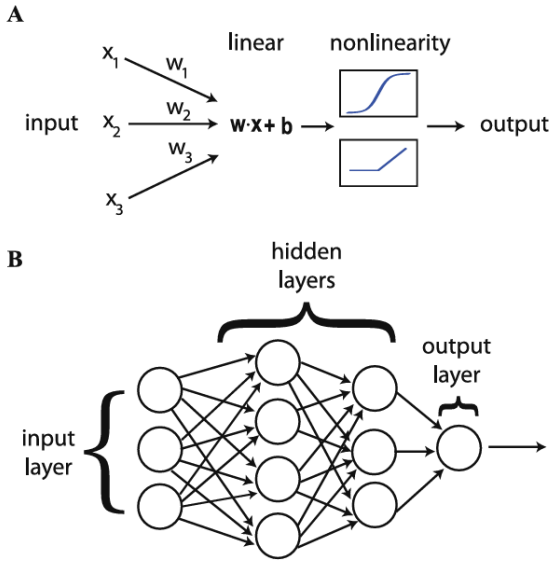
Aloitan neuroverkkojen käsittelyn tarkastelemalla niiden rakennetta ja yksittäisessä neuronissa suoritettavia laskuoperaatiota. Tämän jälkeen jatkan neuroverkkojen oppimiseen ja oppimisessa käytettäviin menetelmiin. Erityisesti keskityn vastavirta-algoritmiin (engl. backpropagation algorithm), joka mahdollistaa gradienttipohjaisten optimointimenetelmien käytön neuroverkon parametrien optimoinnissa. [9]

2.2.1 Eteenpäinsyöttävä neuroverkko

Neuroverkko koostuu kerroksista, jotka ovat päällekkäin pinottuja neuroneja. Edellisen kerroksen neuronit on yhdistetty seuraavan kerroksen neuroneihin ja yksinkertaisuuden vuoksi oletetaan, että jokaisesta edellisen kerroksen neuroneista on yhteys kaikkiin seuraavan kerroksen neuroneihin. Tällaisen yksinkertaisen neuroverkon rakenne on esitetty tarkemmin kuvan 3 alaosassa. [10]

Neuroverkko koostuu tyypillisesti syötekerroksesta (engl. input layer), joka välittää saamansa viestin eteenpäin seuraavalle kerrokselle. Neuroverkon toisessa päässä on tulostekerros (engl. output layer), joka välittää neuroverkon käsittelemän viestin eteenpäin. Näiden välissä olevat piilokerrokset (engl. hidden layer) tekevät saamalleen viestille epälineaarisen muunnoksen, jonka ne lähettävät eteenpäin seuraavalle kerrokselle. [9, 10]

Kuvan 3 yläosassa on esitetty, miten neuronit käsittelevät edellisen kerroksen neuroneilta saamiaan viestejä. Merkitään edellisen kerroksen neuroneiden lähettämiä viestejä vektorilla $\boldsymbol{x}^{l-1} = (x_1^{l-1}, x_2^{l-1},...,x_j^{l-1})$, missä yläindeksi $l-1$ viittaa kyseessä olevan kerroksen järjestysnumeroon ja alaindeksi indeksoi neuronin, josta viesti on lähtenyt. Tällöin seuraavan kerroksen neuroni suorittaa sille ensin lineaarisen muunnoksen $z_i^l = \boldsymbol{w}_i^l \cdot \boldsymbol{x}^{l-1} + b_i^l$, missä yläindeksi l viittaa kerrokseen ja



Kuva 3: Eteenpäinsyöttävä neuroverkon rakenne ja aktivaatiofunktio epälineaarisuuden mahdollistajana. Kuva on lainattu artikkelista *"A high-bias, low-variance introduction to Machine Learning for physicists"*. [10]

alaindeksi indeksoi kyseisen kerroksen neuronin. Vektori $\boldsymbol{w}_i^l$ voidaan kirjoittaa muodossa $\boldsymbol{w}_i^l = (w_{i1}^l, w_{i2}^l, ..., w_{ij}^l)$, jolloin sen komponentit vastaavat neuronien välisten yhteyksien painokertoimia. Alaindeksi ilmaisee miltä edellisen kerroksen neuronilta j viesti on saapunut tarkasteltavalle neuronille i. Summassa esiintyvä b_i^l on lineaarisen muunnoksen vakiotermi (engl. bias). Neuronista lähtevä viesti x_i^l saadaan, kun kyseinen lineaarinen muunnos syötetään epälineaariseen aktivaatiofunktioon (engl. activation function) σ, jolloin $x_i^l = \sigma(z_i^l)$. Kerroksen l neuroneista lähtevät viestit voidaan esittää vektorina $\boldsymbol{x}^l$, joka toimii kerroksen l+1 neuronien syötteenä. [10]

2.2.2 Vastavirta-algoritmi ja neuroverkkojen kouluttaminen

Neuroverkko voidaan ajatella monimutkaisena epälineaarisena muunnoksena. Tällöin neuroverkolla toteutetun mallin parametreina ovat sen neuronien välisten yhteyksien painokertoimet ja vakiotermit. Neuroverkon kouluttaminen puolestaan tar-

koittaa sellaisten painokertoimien ja vakiotermien löytämistä, jotka minimoivat sakkofunktion. Näiden etsimiseen voidaan käyttää gradienttimenetelmää, joka kuitenkin edellyttää gradienttivektorin laskemista neuroverkon painokertoimien ja vakiotermien suhteen kullakin iteraatiolla. [10]

Neuroverkkojen kouluttamiseen tarvittava gradienttivektori voidaan laskea vastavirta-algoritmin (engl. backpropagation) avulla[9, 13, 14]. Esittelen siitä yleisemmän muodon Michael A.Nielsenin kirjasta *Neural Networks and Deep Learning* [14], jossa sakkofunktion muotoa ei ole määritelty tarkasti. Tämä voidaan nähdä yleistettynä muotona alkuperäisestä vastavirta-algoritmista, jossa käytettiin neliöllistä sakkofunktiota $C = \frac{1}{2}\sum_j(\hat{y}_j - y_j^L)^2$[13]. Vaatimuksena näiden molempien suorittamiselle on, että aktivaatiofunktiot ovat derivoituvia [9, 13, 14]. Lisäksi toimivuuden kannalta käytetyn aktivaatiofunktion on oltava saturoitumaton [9, 14].

Algoritmin ensimmäinen vaihe on normaali eteenpäinsyöttö, jossa se laskee tapaukselle $(\boldsymbol{x},\boldsymbol{y}) \in D$ ennustuksen $\hat{\boldsymbol{y}}$. Laskemisen yhteydessä saadaan myös laskettua kullekin neuronille näiden tekemät lineaariset muunnokset z_i^l sekä niiden lähettämät viestit x_i^l, joita tarvitaan myöhemmin gradientin laskemisessa. Lasketuille ennustuksille lasketaan myös sakkofunktion arvo. [14]

Seuraavana saadulle sakkofunktiolle $C(\boldsymbol{y},\hat{\boldsymbol{y}})$ lasketaan osittaisderivaatta ulomman kerroksen neuroneiden tekemien linearisten muunnosten z_j^l suhteen. Tästä käytämme merkintää δ_j^l, jonka avulla saadaan osittaisderivaataksi

$$\delta_j^L = \frac{\partial C}{\partial x_j^L}\sigma'(z_j^L). \tag{7}$$

Tässä L viittaa neuroverkon kerrosten lukumäärään. Mainitsemisen arvoista on, että tämä ja seuraavana esitettävät kaavat voidaan johtaa osittaisderivaattojen avulla. Saadut virheet voidaan propagoida nykyistä kerrosta edeltävälle kerrokselle derivaatan ketjusäännön avulla. Tällöin kerroksen l (<L) neuronin j osittaisderivaataksi δ_j^l saadaan

$$\delta_j^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l). \tag{8}$$

Tämän kaavan avulla saatu virhe voidaan propagoida vastavirtaan neuroverkossa; tulostekerrokselta syötekerrokselle, mistä algoritmi on saanut nimensä. [14]

Kun sakkofunktion osittaisderivaatat neuronien tekemien lineaaristen muunnosten suhteen on laskettu, voidaan tämän avulla laskea osittaisderivaatat parametrien suhteen. Vakiotermien osittaisderivaataksi sakkofunktion suhteen saadaan

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l. \tag{9}$$

Yhteyksien painokertoimien osittaisderivaataksi sakkofunktion suhteen saadaan

$$\frac{\partial C}{\partial w_{jk}^l} = x_k^{l-1}\delta_j^l. \tag{10}$$

Painokertoimen osittaisderivaatta riippuu siis sen välittämän viestin x_k^{l-1} suuruudesta ja virheestä viestin vastaanottavalla neuronilla. [14]

2.3 Graafineuroverkot

Graafineuroverkot (engl. graph neural network) ovat neuroverkkoja, jotka käsittelevät graafeina esitettyä dataa [15, 16]. Graafit puolestaan ovat solmuista (engl. vertex) ja näitä yhdistävistä kaaristа (engl. edge) koostuva tietorakenne, jossa solmut edustavat itsenäisiä olioita ja kaaret kuvaavat näiden olioiden välisiä yhteyksiä [15, 16]. Jokaista solmua ja kaarta vastaa ominaisuusvektori (engl. feature vector). Tämä koostuu kvantitatiivisista ominaisuuksista, joilla pyritään kuvaamaan asiaa tai yhteyttä, jota solmut ja kaaret edustavat [15]. Solmun ominaisuusvektorista käytetään merkintää v_i^F, jossa i indeksoi kyseisen solmun v_i [15]. Kaarien ominaisuusvektoreista käytetään merkintää e_{ij}^F, jossa e_{ij} kertoo kyseessä olevan solmujen v_i ja v_j välinen kaari [15].

Tärkeä verkkoihin liittyvä käsite on solmun lähiympäristö (engl. neighborhood). Lähiympäristö koostuu yksinkertaisimmillaan keskussolmusta v_i, tästä lähtevistä kaarista sekä tähän yhdellä kaarella yhdistetyistä solmuista. Yleisemmin lähiympäristö määritellään solmujen joukkona, jonka jäsenet ovat korkeintaan jotain koko-

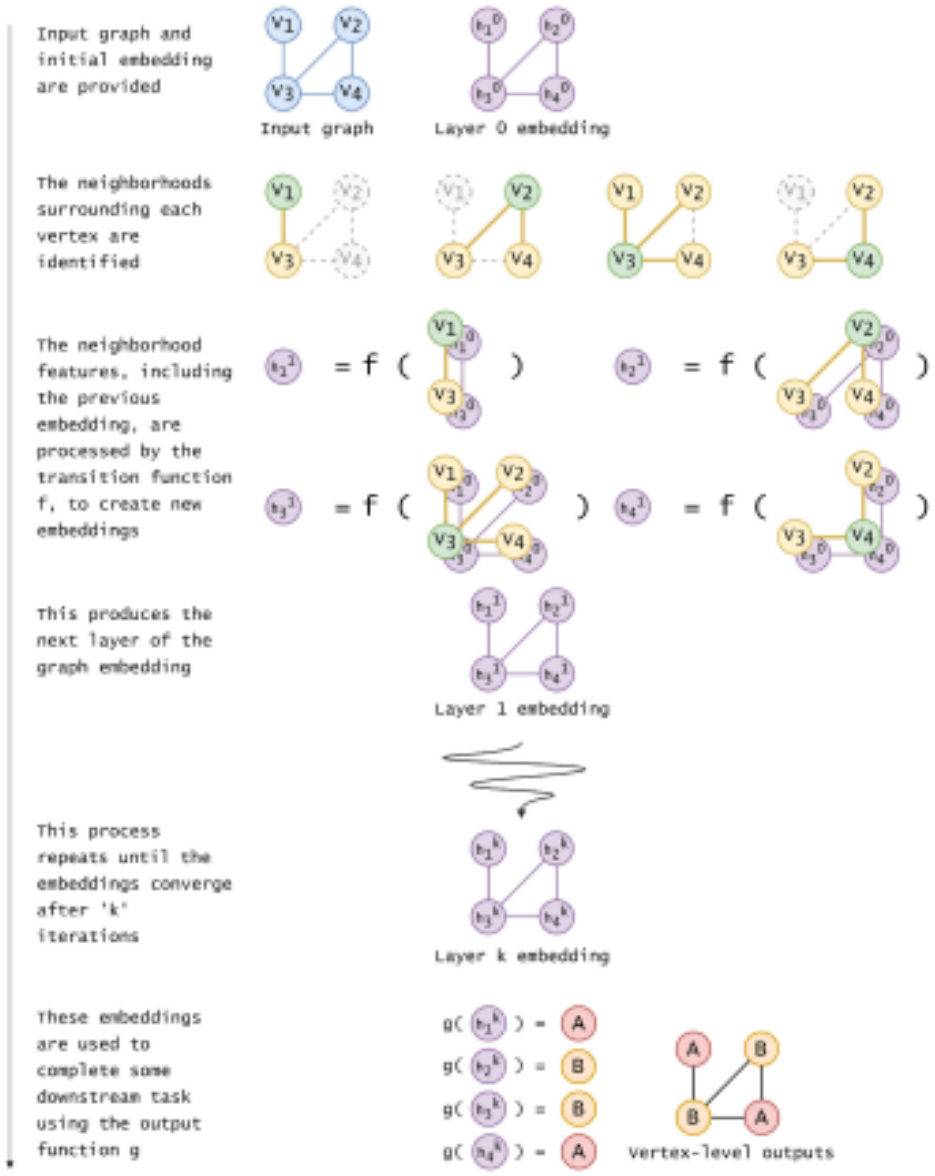
naislukumäärää vastaavan hyppäyksen (kaaren) päässä keskussolmusta. Solmun v_i lähiympäristöstä käytetään merkintää N_{v_i}. [15]

Seuraavaksi käsittelen yhtä molekyylidynamiikan simulaatioiden kannalta lupaavaa graafineuroverkkojen alalajia eli rekursiivista graafineuroverkkoa (engl. recurrent graph neural network), jonka toimintaperiaatetta on havainnollistettu kuvassa 4 [3, 15]. Alussa alkuperäisen verkon ominaisuusvektorit syötetään neuroverkkoon, jolloin solmujen ominaisuusvektoreista saadaan ensimmäiset vektoriesitykset h_i^0 (engl. embedding) [15]. Tämän jälkeen vektoriesitykset päivitetään neuroverkon jokaisella kerroksella, missä neuroverkko laskee solmujen uudet vektoriesitykset siirtymäfunktion f(engl. state transition function) avulla, joka on kaikille solmuille sama [15]. Seuraava vektoriesitys saadaan kaavasta

$$h_i^k = \sum_{j \in N_{v_i}} f(v_i^F, e_{ij}^F, v_j^F, h_j^{k-1}), \tag{11}$$

missä h_j^{k-1} on keskussolmun lähiympäristöön kuuluvan solmun edellinen vektoriesitys [15]. Tämä voidaan tulkita siten, että solmun seuraava vektoriesitys riippuu sen lähiympäristön ominaisuuksista sekä siihen kuuluvien solmujen vektoriesityksistä [15]. Tästä johtuen jokaisella iteraatiolla solmun vektoriesitykseen vaikuttaa aiempaa useamman kaaren päässä olevien solmujen ominaisuudet, jolloin siis iterointi välittää "viestejä"verkkoa pitkin [15]. Iterointia jatketaan, kunnes vektoriesitykset alkavat suppenemaan ja tämän jälkeen viimeisen kerroksen vektoriesityksistä voidaan laskea lopputulos tulostefunktion g avulla [15].

Siirtymäfunktion lisäksi myös tulostefunktio toteutetaan neuroverkon avulla. Graafineuroverkon kouluttaminen vastaa siirtymä- ja tulostefunktiota esittävien neuroverkkojen parametrien eli painokertoimien optimointia. Ohjatun oppimisen tapauksessa tämä onnistuu minimoimalla sakkofunktiota gradienttimenetelmällä vastavirta-algoritmia hyödyntäen. [15]



Kuva 4: Rekursiivisen graafineuroverkon toimintaperiaate. Kuva on lainattu Wardin artikkelista "A Practical Tutorial on Graph Neural Networks". [15]

3 Koneoppimisen hyödyntäminen molekyylidynamiikassa

Neuroverkkoihin perustuvilla koneoppimismenetelmillä voidaan nopeuttaa huomattavasti molekyylidynamiikan simulaatioita, jotka ovat perinteisesti edellyttäneet laskennallisesti vaativan tiheysfunktionaaliteorian käyttöä tarkkojen tuloksien saamiseksi. Erityisesti tukielmassa tarkasteltavaan GemNet-arkkitehtuuriin perustuvien graafineuroverkkojen koneoppimispotentiaalit ja -voimat ovat osoittautuneet robusteiksi ja tarkoiksi. Tämä mahdollistaa lähes alkuperäiseen simulaatioon verrattavissa olevan tarkkuuden saavuttamisen merkittävästi pienemmällä laskennallisella vaativuudella. [6]

3.1 Koneoppimispotentiaali

Potentiaalienergiapinnat (enlg. potential energy surfaces) ovat multidimensionaalisia funktioita $V(\boldsymbol{r}_1,\boldsymbol{r}_2,...)$, jotka määrittävät atomimittakaavan rakenteen ja potentiaalienergian välisen relaation [3, 4]. Ne tarjoavat vaihtoehtoisen approksimatiivisen lähestymistavan esimerkiksi laskennallisesti paljon vaativammalle tiheysfunktionaaliteorialle, joka vaatii elektronien Schrödingerin yhtälön likimääräisen ratkaisemisen [4]. Born-Oppenheimer-approksimaation perusteella potentiaalienergiapinnat sisältävät paljon informaatiota tarkasteltavan systeemin ominaisuuksista [3]. Niistä saadaan ratkaistua yhtälön (2) mukaisesti voimat, joita tarvitaan molekyylidynamiikan simulaatioissa jokaisella aika-askeleella [6].

Potentiaalienergiapintoja voidaan approksimoida koneoppimismenetelmillä saatavien koneoppimispotentiaalien avulla [4]. Koneoppimismallin kouluttaminen tapahtuu lasketun datan avulla [3]. Koulutusdata sisältää atomikonfiguraatiot ja näitä vastaavat potentiaalienergiat, mistä koneoppimismallin on opittava kuvaus konfiguraatiosta potentiaalienergiapinnoiksi [3]. Käytännössä oppiminen tarkoittaa sitä,

että mallin parametrit optimoidaan siten, että sen antamien ennustusten ja tunnettujen arvojen väliset erotukset minimoidaan [3]. Koulutettua koneoppimispotentiaalia voidaan käyttää interpolointiin koulutusdatassa käytetyn konfiguraatioavaruuden sisällä, mutta ekstrapolointi ei välttämättä onnistu [3].

3.2 Koneoppimispotentiaalien kouluttaminen

Koneoppimispotentiaalin kouluttaminen alkaa referenssidatan generoimisella [4]. Tämä on myös koneoppimispotentiaalin tarkkuuden kannalta kriittinen vaihe, sillä koneoppimispotentiaalin tarkkuus ei voi olla koulutuksessa käytettävää dataa parempi [3, 4]. Toisaalta referenssidatan generoiminen on usein koneoppimispotentiaalin kouluttamisen laskennallisesti vaativin vaihe, joten referenssimenetelmän valinta on kompromissi tarkkuuden ja laskennallisen vaativuuden välillä [4].

Jotta laskennallisesti vaativista referenssimentelmän laskuista saataisiin kaikki hyöty, lisätään koulutusdataan myös referenssimenetelmästä saatavat voimat, jotka antavat tietoa potentiaalienergiapinnan muodoista. Rajallisen ekstrapolointi-kyvyn takia datasetin tulisi sisältää monipuolisesti eri rakenteita siitä konfiguraatioavaruudesta, jossa koneoppimispotentiaalia aiotaan käyttää.[4]

Koneoppimismalliin laitettavan syötteen tulee olla sellaisessa muodossa, että saatava koneoppimispotentiaali toteuttaa invarianssit translaation, rotaation ja permutaation suhteen. Tämän varmistamiseksi referenssidata on yleensä muutettu kuvailijoita (engl. descriptors) käyttämällä muotoon, jossa vaaditut invarianssit toteutuvat. Esimerkiksi atomikeskeiset symmetriafunktiot (engl. atom-centered symmetry functions) ovat yksi tapa esittää atomiympäristö tällaisessa muodossa. Tällöin jokaiselle atomille lasketaan sen atomiympäristöä kuvaavat radiaali- ja kulmajakaumat, joihin vaikuttavat kaikki keskusatomin ympäristöön kuuluvat atomit. Atomin ympäristöön katsotaan kuuluvaksi kaikki ennalta määrätyn leikkaussäteen (engl. cutoff radius) sisällä olevat atomit. [3, 4]

3.3 Rekursiiviset graafineuroverkot emulaattorina

Graafineuroverkot ovat menestyneet hyvin atomien välisten vuorovaikutuksien ennustamisessa [17]. Graafineuroverkoihin perustuvissa menetelmissä atomin ympäristö esitetään graafina, jossa atomit ovat solmuja ja kaaret atomien välisiä vuorovaikutuksia [3, 6]. Jokaisesta atomista (solmusta) oletetaan vuorovaikutus kaikkiin leikkaussäteen sisällä oleviin atomeihin [3, 6]. Koneoppimispotentiaalin laskemista varten atomeista tehdään vektoriesitykset, joita päivitetään graafineuroverkossa jokaisella iteraatiolla [3].

Vektoriesityksen päivitys voidaan jakaa kahteen osaan, joista ensimmäisessä lasketaan viesti m_i^{t+1} päivitettävälle vektoriesitykselle h_i^t, jossa i indeksoi atomin ja t iteraation. Viestin funktio on muotoa

$$m_i^{t+1} = \sum_{j \in N(i)} M_t(h_i^t, h_j^t, e_{ij}). \tag{12}$$

Tässä M_t on viestifunktio ja h_j^t:t ovat lähiympäristön atomien vektoriesityksiä. Viestifunktio ottaa parametrikseen myös lähiympäristön atomien vuorovaikutukset e_{ij}. [3]

Viestifunktion m_i^{t+1} ja atomin edellisen vektoriesityksen h_i^t avulla voidaan edelleen laskea uusi vektoriesitys h_i^{t+1} yhtälöllä

$$h_i^{t+1} = U_t(h_i^t, m_i^{t+1}). \tag{13}$$

Tässä U_t on neuroverkon avulla toteutettu opittava funktio vektoriesityksien päivittämiseksi. Neuroverkon koulutus vastaa nyt sellaisen funktion U_t etsimistä, jolla iteraatioiden tuloksena saadaan atomien (approksimatiiviset) energiat. [3]

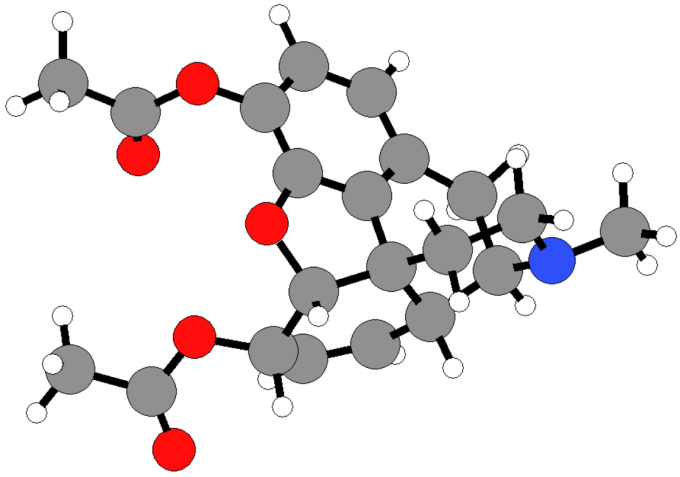
Yksi esimerkki koneoppimispotentiaalin ja voimien laskemiseen käytettävistä graafineuroverkoista on seuraavassa osiossa käytettävä GemNet [17]. Siinä atomien ja niiden välisten vuorovaikutusten kuvaamiseen käytetään moniulotteisia vektoriesityksiä [6]. Niissä käytetään atomien välisiä etäisyyksiä, kaarien välisiä kulmia ja

kaarien välisiä dihedraalisia kulmia kaiken geometrisen informaation sisällyttämiseksi [6, 17]. Päivitettyjen vektoriesityksien avulla GemNet ennustaa molekulaarisen energian ja atomeihin kohdistuvat voimat niiden paikkojen ja järjestyslukujen avulla [17].

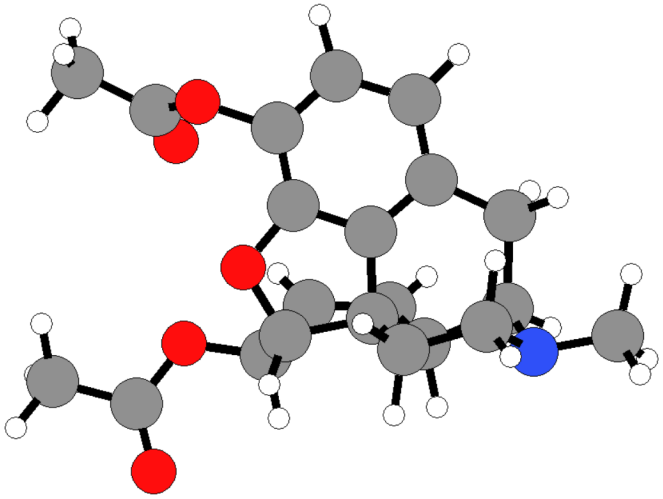
3.4 GemNet-arkkitehtuuriin robostiuden testaaminen

Tutkin GemNet-graafineuroverkon ennustaman koneoppimispotentiaalin robustiutta. Tätä varten implementoin GemNet-verkon ja sovelsin Velocity-Verlet algoritmia ajaakseni molekyylidynamiikkaa. Arvoin atomien alkunopeudet v_i Maxwell-Boltzmann-jakaumasta lämpötiloilla 300K ja 10000K. Simuloin heroiinimolekyyliä ($C_{21}H_{23}NO_5$) kahdessa lämpötilassa etsien voimakentän numeerisia epävakauksia. Yhdessä lämpötilassa simulaatiota ajettiin 1000 fs verran ja aika-askeleen pituus on 0.5 fs.

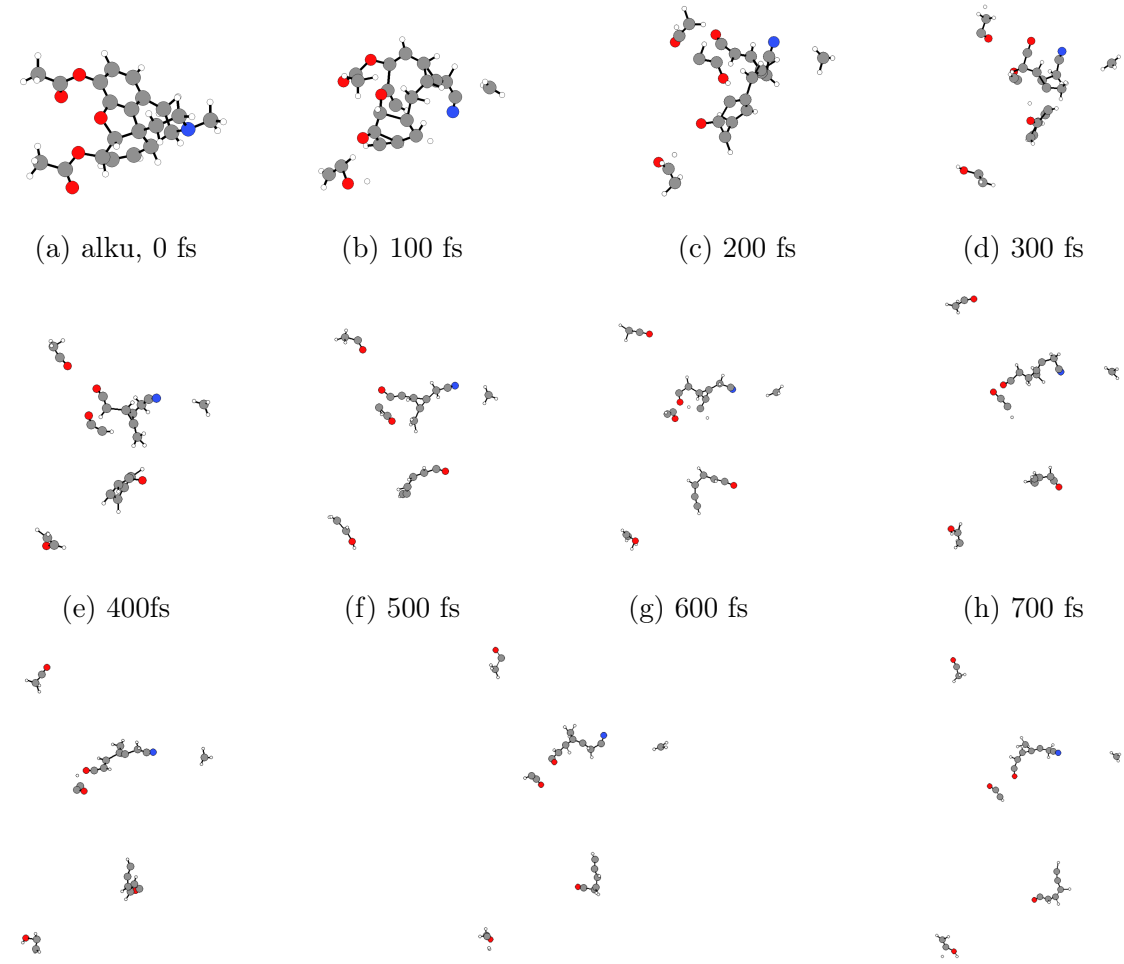
Kuvassa 6 on esitetty heroiinimolekyyli 300K lämpötilassa suoritetun simulaation lopussa. Vertaamalla sitä kuvassa 5 esiintyvään simulaation alkutilanteeseen huomataan, että molekyylin rakenne on pysynyt kasassa simulaation aikana. Kuvassa 7 on esitetty kuvasarja, joka kuvaa heroiinimolekyylin rakenteen kehitystä 10000 K lämpötilassa suoritetussa simulaatiossa. Molekyyli hajoaa termisen dissosiaation takia, mutta simulaatio etenee kuitenkin numeerisesti stabiilisti eikä äkillisiä epäjatkuvuuksia löytynyt. Vaikka havaittu terminen dissosiaatio ei luultavasti edusta reaalimaailman ilmiötä, osoittautuu GemNet poikkeuksellisen vakaaksi koneoppimisvoimakentäksi. Ennusteet käyttäytyvät molemmissa lämpötiloissa säännöllisesti, eikä merkkejä katastrofaalisesta ekstrapolaatiosta löydy.



Kuva 5: Heroiinimolekyyli simulaatioden alkutilassa.



Kuva 6: Heroiinimolekyylin lopputila 300K lämpötilassa tehdyssä simulaatiossa. Molekyyli on pysynyt ehjänä.



(a) alku, 0 fs (b) 100 fs (c) 200 fs (d) 300 fs

(e) 400fs (f) 500 fs (g) 600 fs (h) 700 fs

(i) 800 fs (j) 900 fs (k) 1000fs

Kuva 7: Kuvasarjassa on esitetty heroiinimolekyylin hajoaminen 10000K lämpötilassa tehdyssä simulaatiossa. Kuvat on otettu 200 iteraation välein, mikä vastaa 100 fs:n propagointia. Koko simulaation pituus on 1000 fs:a.

4 Yhteenveto

Tutkielmassa esiteltiin molekyylidynamiikan simulaatioissa yleisesti käytettävä velocity verlet -algoritmi, koneoppimisen menetelmiä yleisesti sekä näiden hyödyntämistä molekyylidynamiikassa. Koneoppimisessa keskityttiin erityisesti molekyylidynamiikan simulaatioissa käytettäviin neuroverkkoihin sekä niihin lukeutuviin graafineuroverkkoihin. Tarkastelun painopiste oli koneoppimismallien kouluttamisessa ja tutkielmassa esiteltiin siinä yleisesti käytettävä gradienttimenetelmä. Lisäksi esiteltiin neuroverkkojen sakkofunktion gradientin laskemiseen käytettävä vastavirta-algoritmi, joka mahdollistaa gradienttimenetelmän käyttämisen neuroverkkojen kouluttamisessa.

Koneoppimisen sovelluksista molekyylidynamiikassa esiteltiin koneopitut potentiaalienergiapinnat, jotka tarjoavat nopeamman vaihtoehdon voimien laskemiseksi simulaatioissa. Koneoppimispotentiaalin kouluttamisen todettiin vastaavan mallin parametrien optimointia sellaiseksi, että neuroverkon esittämä funktio vastaa (approksimatiivisesti) potentiaalienergiapintoja eli kuvausta atomikonfiguraatiosta potentiaalienergiaksi. Lisäksi käytiin läpi koneoppimispotentiaalin kouluttamiseen liittyviä erityispiirteitä. Koneoppimispotentiaalin käyttöä käytännössä demonstroitiin omalla tutkimuksella, jossa ajettiin molekyylidynamiikan simulaatioita kahdessa eri lämpötilassa.

Vaikka koneoppimispotentiaalit pystyvätkin jo approksimoimaan potentiaalienergiapintoja hyvällä tarkkuudella, on niiden robostiutta mahdollista kehittää uuden tutkimuksen myötä. Koska potentiaalienergiapinnat ovat suurissa systeemeissä hyvin kompleksisia funktioita, ei koulutusdatan rajallisella määrällä ole mahdollista oppia kaikkia potentiaalienergiapinnan yksityiskohtia tarkasti. Myös äärellinen leikkaussäde pienentää niiden tarkkuutta. Tällöin ylisovittaminen on todellinen haaste neuroverkkojen taipuisan funktionaalisen muodon takia. Koneoppimismalli voikin oppia äärellisestä koulutusdatasta ja leikkaussäteestä johtuvan varianssin piirteitä

pelkän potentiaalienergiapinnan muotojen sijaan. Tämä luo tarpeen mallien koulutuksessa käytettävien regularisaatiomenetelmien kehittämiselle ja tutkimiselle.

Viitteet

[1] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto ja L. Zdeborová, Reviews of Modern Physics **91**, 045002 (2019).

[2] M. Hutson, Science **367**, 728 (2020).

[3] F. L. Thiemann, N. O'Neill, V. Kapil, A. Michaelides ja C. Schran, Journal of Physics: Condensed Matter **37**, 073002 (2025).

[4] A. M. Tokita ja J. Behler, The Journal of Chemical Physics **159**, 121501 (2023).

[5] M. P. Allen, D. J. Tildesley ja D. J. Tildesley, *Computer simulation of liquids*, *Oxford science publications*, reprinted ed. (Clarendon PrOxford, 2009).

[6] S. Stocker, J. Gasteiger, F. Becker, S. Günnemann ja J. T. Margraf, Machine Learning: Science and Technology **3**, 045010 (2022).

[7] W. C. Swope, H. C. Andersen, P. H. Berens ja K. R. Wilson, The Journal of Chemical Physics **76**, 637 (1982).

[8] J. Behler, The Journal of Chemical Physics **145**, 170901 (2016).

[9] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*, first edition, fifth release ed. (O'ReillyBeijing Boston Farnham Sebastopol Tokyo, 2018).

[10] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher ja D. J. Schwab, Physics Reports **810**, 1 (2019).

[11] S. Badillo, B. Banfai, F. Birzele, I. I. Davydov, L. Hutchinson, T. Kam-Thong, J. Siebourg-Polster, B. Steiert ja J. D. Zhang, Clinical Pharmacology & Therapeutics **107**, 871 (2020).

[12] K. Hornik, M. Stinchcombe ja H. White, Neural Networks **2**, 359 (1989).

[13] G. E. a. W. R. J. a. o. Rumelhart, David E and Hinton, Learning internal representations by error propagation, 1985.

[14] M. A. Nielsen, *Neural Networks and Deep Learning* (Determination Press, 2015).

[15] I. R. Ward, J. Joyner, C. Lickfold, Y. Guo ja M. Bennamoun, ACM Computing Surveys **54**, 1 (2022).

[16] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li ja M. Sun, AI Open **1**, 57 (2020).

[17] J. Gasteiger, F. Becker ja S. Günnemann, GemNet: Universal Directional Graph Neural Networks for Molecules, 2024, arXiv:2106.08903 [physics].