Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Thesis

Deep Learning methods for Computer Aided Drug Design

Ákos Godó
Info-Bionics Engineering MSc

2017

Supervisor:
István Ladjánszki
Faculty Mentor:
György Cserey PhD

I/645/2017

PÁZMÁNY PÉTER CATHOLIC UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY AND BIONICS
REGISTRAR'S OFFICE
H-1083 Budapest, Práter u. 50/a.
Tel: (36-1) 886-4711 FI: 79633 E-mail: tanulmanyi.osztaly@itk.ppke.hu

Diploma Thesis Proposal

Student

Name: Ákos Godó	**Neptun code:** ABTP8A
Study program: Info-Bionics Engineering MSc IMNM-AIB	

Supervisor

Name: István Ladjánszki
Institution, Company: Streamnovation Kft.
Position / Academic degree: Senior Quantumchemist – Work Package Leader

Name of Faculty mentor: György Cserey, PhD

Thesis

Title: Deep Learning methods for Computer Aided Drug Design

Summary of the Thesis:
Recently, machine learning methods have received increasing attention in modern medicine. The work of doctors and researchers can be enhanced by systems processing data in magnitudes impossible to handle manually. Furthermore, these systems can extract features and learn from such datasets and augment or replace human decision making.

A highly relevant task in the pharmaceutical industry during the research of drug candidate molecules is the study of the connection between proteins and the candidate small molecules which would, with high certainty, bind and form a protein-ligand complex. This is of high complexity even for a single protein on the one hand due to the large amount of both possible ligands and binding sites. On the other hand, the calculation of even a single bond with necessary precision requires immense computational power. As such it is practical to augment the currently standard methods with an estimation before the actual calculations to hasten the discovery of drug candidates by eliminating some samples. One feasible approach to this estimation would be the use of machine learning methods.

The goal of this thesis is to understand in detail the inner workings of deep neural networks, the current state-of-the-art machine learning method and to review to the literature on methodological breakthroughs of recent years. Using this knowledge, a deep learning system is to be designed and implemented which, exploiting the widely available high-performance GPU hardware solutions, could speed up a drug discovery process.

Detailed task description:

- Study relevant literature in the fields of machine learning and deep neural networks, recent new results and breakthroughs in methodology.
- Select a relevant drug design task which would be explained in detail and solved
- Create a dataset for the chosen task which would be used to train the system and through the use of which the system's performance could be evaluated
- Introduce and explain the deep neural network architecture serving as the basis of the system
- Compare the performance of the system with earlier methods
- Detail the results, describe observations and present possible enhancement possibilities

Hereby I undertake to supervise the Diploma Thesis work of the student.

[illegible]

Signature of Supervisor

Hereby I undertake to supervise the Diploma Thesis work of the student as his/her institutional supervisor.

[illegible]

Signature of Faculty Mentor

Hereby I apply for the approval of the theme of my Diploma Thesis.

Budapest, 2017. 04. 24.

[illegible]

Signature of Student

The theme of the Diploma Thesis has been approved by the Faculty of Information Technology and Bionics.

Budapest, 2017. 05. 15

[illegible]

Dean

The student has regularly attended consultation sessions, and has met all the requirements of the task description.

Budapest, 2017. 12. 13

[illegible]

Signature of Supervisor

The diploma thesis work meets the formal and substantive requirements of Annex No. 3 of the Code of Studies and Exams.

Budapest, 2017. 12. 13

[illegible]

Signature of Faculty Mentor

I, the undersigned Ákos Godó, student of the Pázmány Péter Catholic University, Faculty of Information Technology and Bionics, declare that I have written this diploma thesis solely myself, without any unauthorized help, and I have only used the sources referenced. Every part quoted word by word or in a paraphrased manner is indicated clearly, with a reference made. I have not submitted this diploma thesis in any other training program.

Signature

Contents

1. Introduction 9

1.1. Detailed explanation of tasks 11

2. Problem definition 12

2.1. Acquiring Potential Energies 12

2.2. The UFF Forcefield 14

3. Methods 19

3.1. Machine Learning and Neural Networks 19

3.2. Establishing a Dataset 31

3.3. Neural Network Architecture and Training Strategies 35

4. Results 38

4.1. Reaction Energy Strategy 39

4.2. Separate Energy Strategy 40

5. Conclusion 46

Abstract

A highly relevant task in Computer Aided Drug Design during the research of drug candidate molecules is the study of the bonding affinity between proteins and various small molecules which exhibit biological or pharmacological activity. This task requires a large amount of computational resources because even for a single protein a high quantity of binding sites and feasible ligand molecules exist. Thus, it is customary to support the high-precision simulations by less resource intensive calculations beforehand to filter out less promising candidates. A possible angle of approach to this early approximation is the use of machine learning methods.

To be able to determine if a protein-ligand binding reaction would take place the reaction energy must be obtained. I presented various methods currently in use both in the industry and academic research which are used to obtain these energies: the basics of the Density Functional Theory (DFT) and wave function based methods while diving into further detail with the Molecular Mechanics based approach.

I constructed a dataset for training and evaluation using the 2016 edition of the PDB-Bind Database [1][2] as a source of samples and the energies according to the Universal Force Field (UFF) [3], a MM-based method, served as their labels. Due to the initially low sample size I augmented the dataset using a line search algorithm adapted to position the ligands in new positions around the protein and cleft the pocket regions serving as the loci of the reaction to reduce computational complexity.

My choice of machine learning approach was the use of deep neural networks and the novel Atomic Convolutional Neural Network (ACNN) [4] architecture. I pre-processed the established dataset by implementing a parallel neighbour listing algorithm and prepared the data for use with the ACNN. The training phase has been executed according to two strategies, one focusing on the reaction and the other on the reactants' contribution. Although held back by a low sample size, the architecture proved highly capable of handling chemical information offering errors within an order of magnitude between the energy labels and the energies estimated by the network during the strategy focusing on the reactants.

Kivonat

A számítógépes hatóanyagtervezésben a gyógyszerkandidát-molekulák kutatása során releváns feladat a kötési affinitás vizsgálata fehérjék és biológiai vagy gyógyszertani aktivitást mutató kismolekulák között. Ez a feladat nagy mennyiségű számítási erőforrást igényel, mivel már egyetlen fehérje esetén is nagy számú lehetséges ligandmolekula és kötési hely létezik. Ezért megszokott, hogy a nagy pontosságú szimulációkat, a kevésbé ígéretes jelölteket kiszűrendő, előzetesen támogatják kevésbé erőforrásigényes számítások. Egy lehetséges megközelítése ennek a korai becslésnek a gépi tanulási módszerek használata.

Egy fehérje-ligand reakció létrejöttének meghatározásához hozzá kell jutni a reakcióenergiához. Bemutattam különféle, mind az iparban, mind pedig kutatásokban jelenleg használt módszereket, melyekkel megszerezhetőek ezek az energiák: a sűrűségfunkcionál elmélet (density functional theory, DFT) és a hullámfügvény alapú módszerek alapjait, a molekulamechanikai megközelítést pedig részletesebben mutattam be.

Létrehoztam egy adathalmazt tanításra illetve tesztelésre a PDBBind adatbázis [1][2] 2016. évi kiadását használva a minták forrásaként, a hozzájuk tartozó címkékként pedig a Universal Force Field (UFF) [3] molekulamechanikai módszer által kiszámolt energiák szolgáltak. Az alacsony kiindulási mintaszám miatt kibővítettem az adathalmazt egy lineáris keresési algoritmust használva, mellyel új pozíciókba helyeztem a ligandmolekulákat a fehérje körül. Kivágtam továbbá a reakció helyeként szolgáló úgynevezett zsebrégiókat, a számítási komplexitást csökkentendő.

A választott gépi tanulási módszerem a mély neurális hálózatok illetve az újszerű atomi konvolúciós neurális hálózati architektúra (Atomic Convolutional Neural Network, ACNN) [4] használata volt. A létrehozott adathalmazt előfeldolgoztam és előkészítettem a rendszer általi használatra egy általam implementált párhuzamosított szomszédlistázó algoritmussal. A tanítási fázist két stratégia szerint hajtottam végre, elsőként a reakcióra, majd pedig az egyes összetevők hozzájárulására koncentrálva. Bár a rendszer teljesítményét visszafogta az alacsony mintaszám, az architektúra igazán hatékonynak bizonyult a kémiai információ kezelésében, egy nagyságrenden belüli hibát nyújtva az energiacímkék és a becsült értékek között az összetevőkre fókuszáló stratégia során.

1. Introduction

Computer Aided Drug Design (CADD) is a field dealing with the storage, management and analysis of the vast amount of chemical data available to be used for accelerating the development of pharmaceuticals and therapeutic solutions. Its current focus includes improving the design and management of data sources as the amount of data at its disposal keeps growing at an accelerating pace due to the automated generation of pharmacologically interesting compounds [5].

One of the main tasks of CADD is lead detection: the search for chemical compounds that exhibit biological or pharmacological activity (although with suboptimal structure that requires further optimizations) for use in the discovery of drug molecules. The usual method of action of such compounds (an inhibitor or a ligand) is through binding with a target molecule, for example a protein, at a binding site or receptor. Various strategies are available for lead detection depending on the amount and properties of data available describing either the target or the lead compound. The so-called direct approach employs the three-dimensional structural information (the position of constituting atoms) of the molecules, with the indirect approach working with comparative analysis based on structural features. In the case of a structure-based drug design effort the three-dimensional structures of the target and compound are both known [6].

CADD is aided by the availability samples in publicly accessible databases such as the Protein Data Bank (PDB) [7] for protein structures which at the time of writing had 125,312 protein entries available out of which 35,901 were proteins present in human biology. The three-dimensional molecular structure of these samples is usually obtained through the use of X-ray diffraction or NMR [8]. These databases enable structure based drug design tasks with a low preliminary data collection burden.

Another factor which helps with such tasks is the increasing performance of consumer-grade computer hardware and computing services. The availability of relatively cheap cloud based solutions with scalable performance such as the Elastic Compute Cloud (EC2)[9] of Amazon Web Services enables projects where computations are to be centralized or the necessary computing power is not available locally. With the advent of high-performance GPUs parallel computing got a significant boost in the last decade largely aiding CADD with its advances. Current cutting edge consumer-grade GPUs can reach computing performance in the teraflops. These two factors allow for fast and efficient processing of chemical information, despite these, a new caveat has surfaced due to the very same advancements.

Although there is a wealth of data available, and their processing has become both easier and faster, all the algorithms in use have trade-offs in speed and accuracy. The current estimates for the number of natural proteins is approximately around 25,000, along with the number of possible compounds with interesting pharmaceutical properties are estimated to be in the magnitude of 10^{60} [5]. One might be tempted to attempt performing a structure-based CADD task using brute force approaches only to waste time, money and various other resources.

A typical cycle of discovering a new drug, from the identification of the possible compounds to the end of clinical trials, was estimated to take up to 14 years and 800 million USD in 2009, put to be in the billions as of 2017 [10]. Of importance is that these costs not only keep growing, but keep accelerating in their growth too: from costing about 4 million US dollars in the 1960's, through taking over 350 million USD in 1996, to the billion-dollar estimate in the current year. Any tool that could increase the efficiency or decrease the time necessary in the drug discovery is highly sought after and would bring with itself immeasurable advantages. Various projects have been started to hasten the process and reduce costs, proving that this indeed is an important task not only for research purposes, but for the industry as a whole. One such project is the Horizon 2020 research and innovation program, funded by the European Union and offering over 80 billion EUR of total funding [11].

A field that has also immensely benefited from the influx of data and growing computational capabilities is machine learning. Such methods use a relatively low amount of already processed data (in comparison to all data that would be available, ever) to extract features from them and become able to apply the extracted knowledge to new, never before seen samples. These algorithms have already proven themselves in various fields: offering near-perfect handwriting recognition [12], medical diagnoses [13] by assisting doctors in medical image processing and effortless separation of malicious e-mail messages from legitimate ones [14]. The goal of this work is to find a machine learning algorithm which, once trained, could feasibly approximate the results of chemical simulations at a much-lower computational cost, filtering and leaving only those compounds which are worthwhile running more expensive simulations on.

In this work I will outline my attempt to create such a system. I will present the various steps taken and considerations that led to them being taken: from the definition of the problem to be solved, the data collection and warehousing necessary for a training set to be established and augmented, the machine learning method chosen to tackle the problem, its inner workings in detail, to the results achieved. Finally, I would like to provide possible enhancement proposals and outlook based on my observations and the current state of the science of deep learning.

1.1. Detailed explanation of tasks

- **Study relevant literature in the fields of machine learning and deep neural networks, recent new results and breakthroughs in methodology:** Literature review tied into the selection of a relevant task, choice of approach and strategies. The relevant points have been described all throughout this work.

- **Select a relevant drug design task which will be explained in detail and solved:** I chose the task of estimating whether or not a ligand would bind to a protein in a given conformation using machine learning approaches. Determining the reaction energy and using it to decide if a conformation is binding is explained in chapter 2, Problem definition.

- **Create a dataset for the chosen task which will be used to train the system and through its use the system's performance can be evaluated:** The dataset used for training and evaluation was created based on the PDBBind Database [15] with reference labels provided by the Universal Force Field [3] implementation of the openbabel [16] Python package. Further augmentation was also performed as it was deemed necessary. These steps are explained in detail in the Establishing a Dataset section of chapter 3, Methods.

- **Introduce and explain the deep neural network architecture serving as the basis of the system:** The novel (introduced in 2017) Atomic Convolutional Network architecture [4] is used to tackle the task at hand. Its inner workings are explained in detail in the Machine Learning and Neural Networks section of chapter 3, Methods.

- **Compare the performance of the architecture with earlier methods:** The network's performance is tested against the Universal Force Field [3] reference energy labels of the dataset.

- **Detail the results, describe observations and present possible enhancement possibilities:** The findings and various enhancement possibilities based on these are outlined whilst providing outlook in chapter 5, Conclusion.

2. Problem definition

When considering the viability of chemical reactions one of the most important factors is its spontaneity. While this holds little information about the speed of the reaction, it does describe if the reaction in question will occur at all. A way to examine this is through the reaction energy. Chemical processes always happen in a way that the resulting compound is in an energetically more favourable state, meaning that it has less energy than the reactants before the reaction took place. Hess's Law states we can obtain the reaction energy by subtracting the energies of the reactants from the resulting compound. For a protein-ligand bonding reaction this is described below in equation 2.1.

$$E_{reaction} = E_{complex} - E_{protein} - E_{ligand} \tag{2.1}$$

The energy of a protein-ligand binding reaction. If the resulting complex is energetically favourable ($E_{reaction} < 0$) the reaction occurs spontaneously.

2.1. Acquiring Potential Energies

The exact potential energy of each molecule can be obtained by using quantum-chemical simulations. Quantum chemistry applies quantum mechanical models to ascertain chemical properties of chemical systems. Ideally, the chemical properties of any system could be determined by solving the Schrödinger equation with the corresponding molecular Hamiltonian which describes the energy of the electrons and nuclei in a molecule (essentially quantifying the various energy contributions of the components of the system). But an exact solution for the Schrödinger equation is only obtainable up to the reduced three-body problem (the case of Helium) and these molecules can have larger atoms possibly numbering in the ten-thousands.

As such, approximations are necessitated thus creating the trade-off between accuracy and speed. These approximative methods run the gamut from using classical mechanics in their models representing atoms and bonds with weights on springs or more advanced ones employing wave function approximations and the Density Functional Theory (DFT). Wave function-based methods, such as the Hartree-Fock method (HF) approximate the Schrödinger equation's solution and provide an estimate for the resulting energies. The main issue with the HF approach is that it treats electrons as uncorrelated entities meaning they can move independently of one another, while upgraded wave function-based methods solving this issue have difficulties with scalability which undermines their usefulness in biomedical applications [5].

DFT-based calculations swap the many-body electronic wave function for electron density as their base quantity. This is a simpler to deal with and the practicality of this method relies on the theorems by Hohenberg and Kohn [17], stating:

1. The ground state electron density determines the wave function (thus all ground state properties)
2. An electron distribution's energy is a functional of the electron density and is minimal for the ground state

This way the solution is obtained by minimizing an energy functional instead of solving or approximating the many-body Schrödinger equation. This effectively sidesteps the issues and limitations of the wave function methods.

Molecular mechanics (MM) based methods being less accurate by no means excludes them from being useful: using them to give quick (relatively speaking, getting results can take longer depending on the system's size) energy approximations can save time by not running higher complexity methods on samples that were proven to be not promising early on. The initial approximations can spare us from spending any further computational resources should it be deemed unnecessary but they still take considerable time to calculate.

Despite being used for early assay purposes, MM-based approaches can still take significant time. A relatively small single molecule can take minutes or hours to process and considering the aforementioned figures on the amount of compounds available it becomes clear to see that running even MM-based simulations on every possibility without any filtering is simply unacceptable. This motivated me to attempt to create a machine learning based estimator of an MM approach.

To accomplish this, a machine learning method must be found which is capable of handling chemical information and is robust enough to work with the wildly varying sizes of molecules and able to use the recent advances in computing (GPUs) to the fullest. A dataset for training and evaluating the system has to be established which includes exploring various sources for relevant data, processing it to be in a format which is compatible with the aforementioned machine learning method and augment the dataset it if necessary.

One of the most important parts of seeing a machine learning project to fruition is the acquisition of a well-defined and reliable dataset. So much so that a sufficiently large one can enable non-specialized architectures to outperform highly specialized models employing an inferior dataset. The chase to reach ever better performance is a concerted effort between the available data and the model designed to fit it. It is paramount to know what the nuances of the task are and what features we can expect to be extracted to prevent the neural network from becoming a black box, preventing us from understanding its inner workings and enhancing its performance.

To create a Neural Network based reaction energy estimator, a reference for these energies must be found. I chose to work with Molecular Mechanics (MM) based energies as their literature is well defined and being an entirely in silico method it is simple to augment the dataset with new samples should the need arise. MM-forcefields calculate several bonded (bond lengths and angles, dihedral angle torsion, etc.) and non-bonded (Van der Waals and electrostatic) terms and sum them up to obtain the energy of the whole molecule.

The information required to calculate these partial energies is looked up from a table where the approximate energy of the atom type in question is listed. These approximate energies are obtained by fitting the parameters of the forcefield to experimental results. The atom types of a forcefield contain information on the element of an atom and their place and connections within a molecule. They also hold data on the atom's energy contribution to each partial energy term. For instance, the type of a carbon atom depends on what kind bonds it forms and with what atoms: the *CA* atom type corresponds to an aromatic carbon atom, whilst the *CZ* type marks a carbon atom within a cyano group. The atom types all have their own characteristic number of bonds, bond lengths, bond angles and so on. These parameters are obtained experimentally and can be applied to a wide variety of molecules.

2.2. The UFF Forcefield

My choice of forcefield was the Universal Force Field (UFF) [3] which consists of the following terms:

- Bond stretching (E_R)
- Bond angle bending (E_θ)
- Dihedral angle torsion (E_ϕ)
- Inversion terms (E_ω)
- Van der Waals terms (E_{vdw})
- Electrostatic terms (E_{el})

The potential energy of a molecule, according to the UFF, arises as follows: $E_{UFF} = E_R + E_\theta + E_\phi + E_\omega + E_{vdw} + E_{el}$. I chose this forcefield due to its applicability to a large range of molecules due to it employing rather general terms. I will provide an explanation of each term and their contributions to the total energy in detail below.

Covalent bonds are the backbone of a molecule. They are formed when two atoms establish a shared electron pair. The energy of a bond is smallest when it is at its natural length. Compressing the bond makes the electron clouds of the bonded atoms overlap, leading to a gradual increase in energy (figure 2.1). Stretching the bond also increases its energy until a certain length where the bond will dissociate [18]. The bond stretching term is described either as a harmonic oscillator (equation 2.2) or as the Morse function as shown in equation 2.3.

$$E_R = \frac{1}{2} k_{ij} (r - r_{ij})^2 \tag{2.2}$$

$$E_R = D_{ij}[e^{-\alpha(r-r_{ij})} - 1], \text{ where } \alpha = (\frac{k_{ij}}{2D_{ij}})^{\frac{1}{2}} \tag{2.3}$$

The bond stretching energy term as a harmonic oscillator (eq. 2.2) and as the Morse function (eq. 2.3). The force constant (k_{ij}) in kcal/mol and the equilibrium bond length (r_{ij}) between atoms i and j are based on their typings. D_{ij} is the bond dissociation energy.

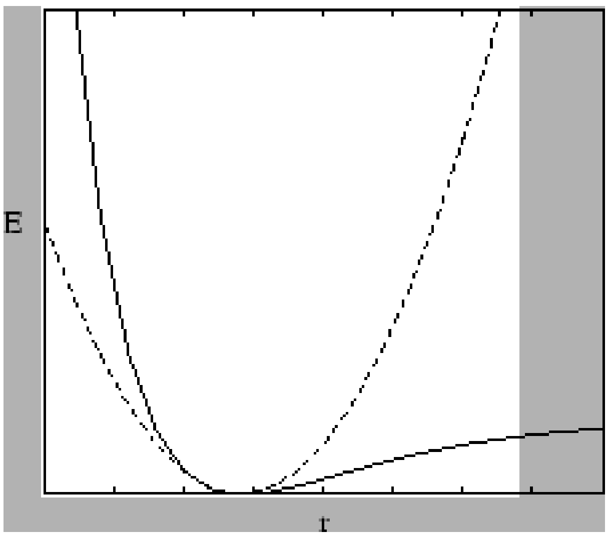

Figure 2.1.: Bond energy (E) as a function of the bond length (r). If overstretched, the bond dissociates (greyed out region on the right). Source: [18]

Angular distortions can occur between two bonds (Angle Bending) or amongst multiple ones (Dihedral Angle Torsion). The energy required to bend a bond out of equilibrium (the angle between the bonds of an atom are considered, as shown in figure 2.2) is much lower than the energy needed to stretch it [19]. UFF describes the angle with a Fourier expansion in θ (equation 2.4), its accuracy can be increased by including higher order terms.

$$E_\theta = K_{ijk} \sum_{n=0} mC_n cos(n\theta_{ij}) \tag{2.4}$$

The angle bend term of the UFF. C_n constants are chosen to satisfy a boundary condition where the energy is minimal at the natural bond angle (θ_{ij_0}).

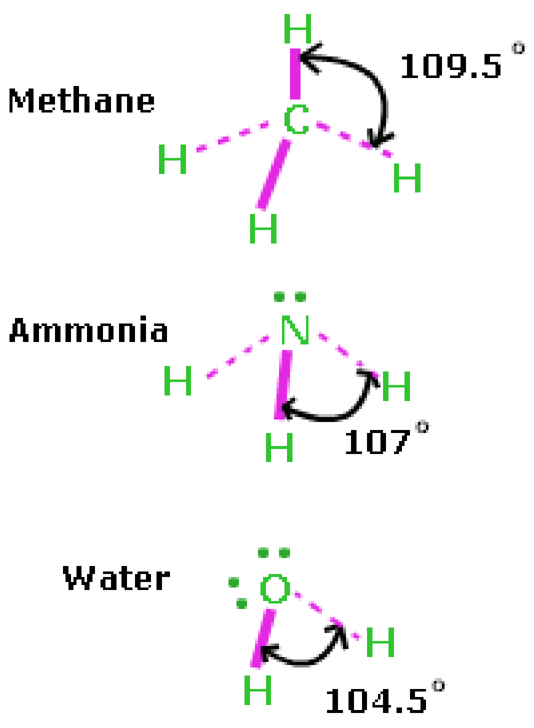

Figure 2.2.: Bond angle bending terms account for the energy necessary for opening or closing the angle between two bonds, a few of which are shown here. Source: [20]

A dihedral angle is the angle between two planes which intersect each other. In a molecule these planes have 4 atoms in total, with two in common (they are altogether two sets of three atoms) [21]. Another way to describe it is the angle between the two bonds ij (connecting atoms i and j) and kl (between atoms k and l) with atoms j and k being connected by bond jk as the bond between two atoms is always in their plane: ϕ_{ijkl} (figure 2.3). Similarly to E_θ, the torsion term is also described as a Fourier expansion in equation 2.5.

$$E_\phi = k_{ijkl} \sum_{n=0}^{m} C_n cos(n\phi_{ijkl}) \tag{2.5}$$

Dihedral angle torsion term of the UFF. As in the case of E_θ (eq. 2.4) the C_n terms minimize the energy at equilibrium.

Inversion terms are present when an atom i is connected to exactly three other atoms j, k and l. In this case there are three angles between the plane ijk and the unique axes formed by the bonds ij, ik and il if i is the central atom [21]. Inversion refers to the act of inverting this structure around one of these axes. This term is expressed as a second degree Fourier expansion of the angle between the il bond and the ijk plane, ω_{ijkl}, as shown in equation 2.6.

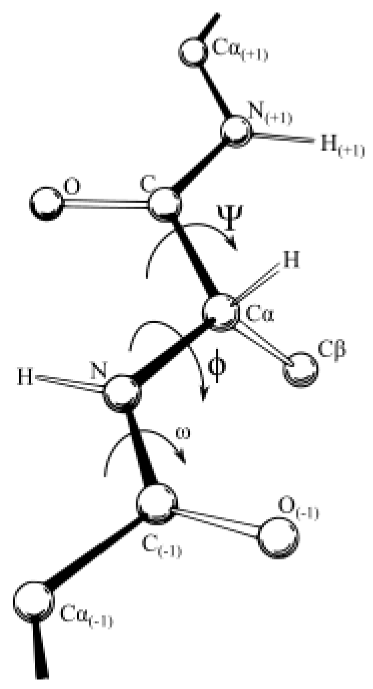

Figure 2.3.: Dihedral angles within a protein. Source: [22]

$$E_\omega = k_{ijkl}(C_0 + C_1 cos(\omega_{ijkl}) + C_2 cos(2\omega_{ijkl})) \tag{2.6}$$

The inversion term with regards to the angle between ijk plane and the il bond, ω_{ijkl}.

Non-bonded interactions, as their name would suggest, do not occur along bonds and as such are even capable of affecting other molecules in a system other than the one they originate from. The electrostatic interaction (equation 2.7) is due the inhomogeneous distribution of charge within a molecule [23]. This is modelled by point charges at atoms and the net electrostatic interaction must be zero for a non-charged molecule. Although this effect is much weaker than a bonded interaction, it is capable of acting at a distance and is not negligible.

$$E_{el} = 332.0637(\frac{Q_i Q_j}{\epsilon R_{ij}}) \tag{2.7}$$

The electrostatic interaction depends on the charges of the interacting atoms (Q_i, Q_j) and their distance (R_{ij}). ϵ is the dielectric constant.

The Van der Waals term incorporates all non-bonded interactions not covered by the electrostatic term including dispersion, repulsion and induction [23]. The Van der Waals energies (D_i) for each atom are obtained according to their typing. The contribution of the inter-atomic distance in the Van der Waals interaction is shown below in equation 2.8.

$$E_{vdw} = D_{ij}[-2(\frac{x_{ij}}{x})^6 + (\frac{x_{ij}}{x})^{12}] \tag{2.8}$$

The Van der Waals interaction depends on the distance of atom i and j, x_{ij} and D_{ij}, the so-called well depth, obtained as $(D_i D_j)^{\frac{1}{2}}$.

As shown above, the total UFF energy (in units of kcal/mol) is obtained as the sum of these terms ($E_{UFF} = E_R + E_\theta + E_\phi + E_\omega + E_{vdw} + E_{el}$) which contain a variety of both bonded and non-bonded terms. The estimator is expected to find similar features within the molecule in order to become able to approximate their energies (and any other input sample presented to it). To assist with this task a capable machine learning method must be found and a high-quality dataset must be established for training and evaluation.

3. Methods

3.1. Machine Learning and Neural Networks

The goal of Machine Learning, in general, is to endow computers with the ability of making reasonable decisions when faced with unexpected data or when storing and/or preparing algorithmically for all possible input-output pairs is impossible. This can either stem from the sheer amount of possibilities available or the fact that we, as the architects of a system, have not seen enough information to create algorithms to deal with the data reliably.

The current task is an illustrative example: it is both unreasonable and infeasible to store a lookup table with every known molecule's structure and their potential energy. Even if it were achievable, in its current state our system only accounts for known molecules and disregards chemically important factors such as their environment, neighbours and position which can not only greatly influence the energy associated with it, but it will also have the table explode in size. Instead, finding the connection between the two, pinpointing the characteristics which contribute the most (the so-called features embedded implicitly into the data) and storing these would result in a much more robust system: which is indeed what the rules are as explained by Chemistry. Machine Learning deals with designing systems that can extract these connections by themselves when a (sometimes extremely) limited amount data is available and use the learnt insights to render judgement when new, never-before-seen samples are presented to the system.

Machine Learning methods can be grouped into two main classes: unsupervised and supervised learning approaches. The difference between the two is the availability of labelled training samples. Supervision is possible through comparing the system's outputs against the desired outputs defined in the labels, tuning its parameters to achieve better results: supervised learning requires labelled samples. Unsupervised learning is able to group samples based on similarity. Clustering methods such as k-Nearest Neighbour (k-NN) or Naive Bayes Classifiers are unsupervised learning techniques. Supervised learning, as mentioned above, needs a training set with labels containing the desired output corresponding to each sample. Regardless of the task, there exists a function of the inputs which maps the required values to them: the desired class in a classification task or an arbitrary value in a regression task. The goal is to approximate this mapping to the best of our and the system's abilities.

The Artificial Neural Network is a supervised Machine Learning method, a system comprised of interconnected, identical processing units, the so-called Artificial Neurons. It is capable of extracting features from samples and approximating the desired mapping to the desired outputs through its learning algorithm and generalize this information in a wide variety of tasks when faced with never-before-seen samples. Their tolerance to noise and scalability has earned them a plethora of accolades, achieving state-of-the-art performance in several fields such as Computer Vision [24] and Medical Image Processing [25].

The building block of a Neural Network is the Artificial Neuron, a non-linear processing unit with a single output and possibly multiple inputs. The output, also called the response, is determined by the weighted sum of its inputs passing through a non-linear function ϕ, the activation function. Thus the output y of an artificial neuron can be obtained as described in equation 3.1.

$$y = \Phi(\overline{w}^T \overline{x}) \tag{3.1}$$

The output of an artificial neuron, y, with activation function Φ, input vector $\overline{x} = (x_0, x_1, \ldots, x_n)$ and the corresponding weight vector $\overline{w} = (w_0, w_1, \ldots, w_n)$.

This model is based on the mechanism of real neurons (figure 3.1). In nerve cells, the stimuli are collected through the dendrites and funnelled towards the cell body, the soma. Should this accumulated stimulus exceed a threshold (the action potential of the neuron) the neuron will fire. The outgoing potential travels along the axon of the cell, reaching the synapses where it will transfer to the dendrites of other neurons. Furthermore, the effect that the synaptic potential can be modulated, both attenuated or amplified. The so-called post-synaptic potentials can be either inhibitory (IPSP for Inhibitory Post-Synaptic Potential) or excitatory (EPSP for Excitatory Post-Synaptic Potential), respectively increasing or decreasing the intensity of the pre-synaptic potential's effects on the post-synaptic neuron. In the artificial neuron model, the inputs correspond to the dendrites, the weights to the post-synaptic potentials and finally the action potential to the activation function.

Of note is that the artificial neuron has a so-called bias term. The bias largely influences the threshold of the activation and determines an offset like quantity. In a nerve cell this could correspond to the neuron being primed: a high bias could have it fire even though its inputs or weights are small and conversely a low bias could dull a neuron even with strong inputs or weights. Due this special role it is usually treated as a separate input (b) or the first of the inputs (x_0) with the first of the weights being its corresponding weight (w_0). Separating the bias (treating the inputs as $\overline{x} = (x_1, x_2, \ldots, x_n)$ and the weights as $\overline{w} = (w_1, w_2, \ldots, w_n)$ from now on) we can determine the output of an artificial neuron as shown in equation 3.2.

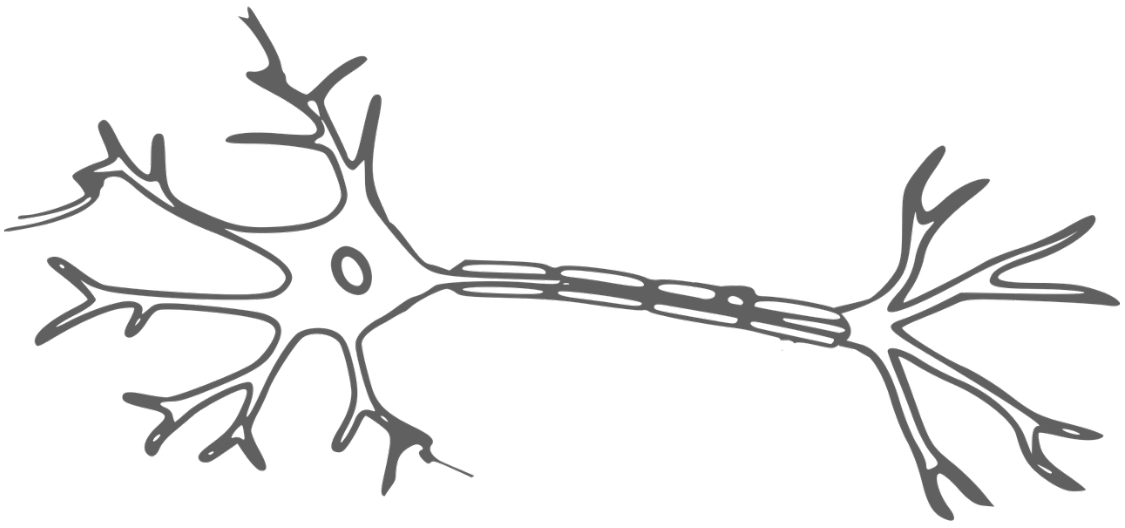

Figure 3.1.: A myelinated nerve cell. The dendrites (on the left) collect inputs and feed the to the cell body. The neuron fires if a threshold (the action potential) is reached and a depolarization spreads towards the end of the axon (on the right).

$$y = \Phi(\overline{w}^T\overline{x} + b) = \Phi(\overline{w}^T\overline{x} + w_0x_0) \tag{3.2}$$

An artificial neuron's output, y, with the bias term, b, separated from the inputs $\overline{x} = (x_1, x_2, \ldots, x_n)$ and $\overline{w} = (w_1, w_2, \ldots, w_n)$, first as a separate term, then expressed as $b = w_0x_0$.

One such artificial neuron model is Rosenblatt's Perceptron [26]. Even a single perceptron is capable of learning from its inputs. Its activation function is sign function, $sgn(x)$, thus its outputs are binary: -1 or $+1$. In classification tasks it separates input samples into two classes with a hyperplane. A single perceptron can only perform linear classification tasks along the w^Tx hyperplane. Positives samples (ones that are deemed to be positive by the perceptron) are grouped towards the direction the normal vector of w^Tx points in. Thus samples deemed to be negative are grouped on the other side of the separating hyperplane.

Artificial neurons (as shown in figure 3.2) and consequently neural networks are supervised machine learning approaches. The supervision they require is through comparing their outputs with predetermined labels of the inputs and using any error created through a possible mismatch between the two to update their parameters to increase future accuracy. The training of a perceptron is done using a training set T containing input samples and desired outputs corresponding to each sample within. As seen in equation 3.3 the training set can be defined as pairs of samples and outputs in the range the perceptron (or a neural network) can output.

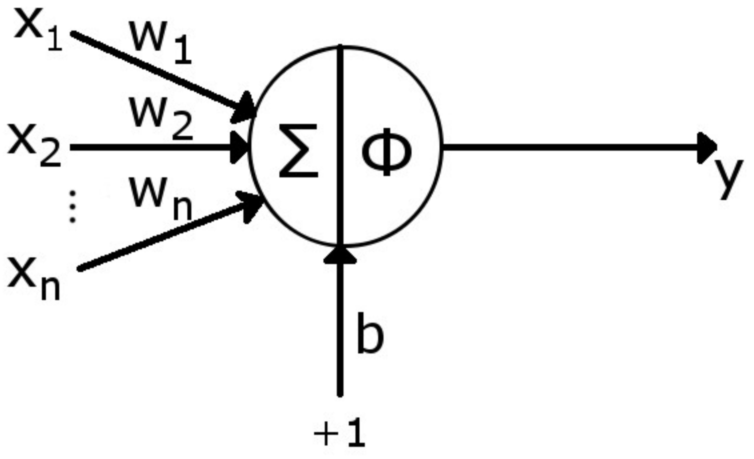

Figure 3.2.: An artificial neuron. The inputs (x_i) and weights (w_i) are on the left with the bias term b separated (bottom). The weighted inputs are summed, then the non-linearity (ϕ) is applied giving us the output, y.

$$T = \{(\overline{x}, d) | d \in \{-1, 1\}\} \tag{3.3}$$

The training set T contains pairs of input samples ($\overline{x}$) and their labels (d). It is important to have the expected outputs align with the capabilities of the system - in this case due to using $sgn(x)$ they must either be -1 or 1.

The trainable parameters of the perceptron are its weights (including the bias). As described above, should the output of the unit and the label not be the same, some error is generated which is then used to update the weights of the perceptron. The extent of the change in weights is regulated by the learning rate η (where the bias may or may not have a separate learning rate). The weight update rule in this case is the so-called perceptron learning algorithm described in equation 3.4. Practical use of the learning algorithm is supported by the fact that for a fixed training set the weights converge under a finite number of iterations [27].

$$\overline{w}_{i+1} = \overline{w}_i + \eta(d(k) - y(k))\overline{x}(k) \ , where \ (\overline{x}(k), d(k)) \in L \tag{3.4}$$

The update rule for a perceptron's weights ($\overline{w}_i$): the perceptron training algorithm. $(d(k) - y(k))$ is the difference between the expected and actual outputs for the k-th sample of the training set.

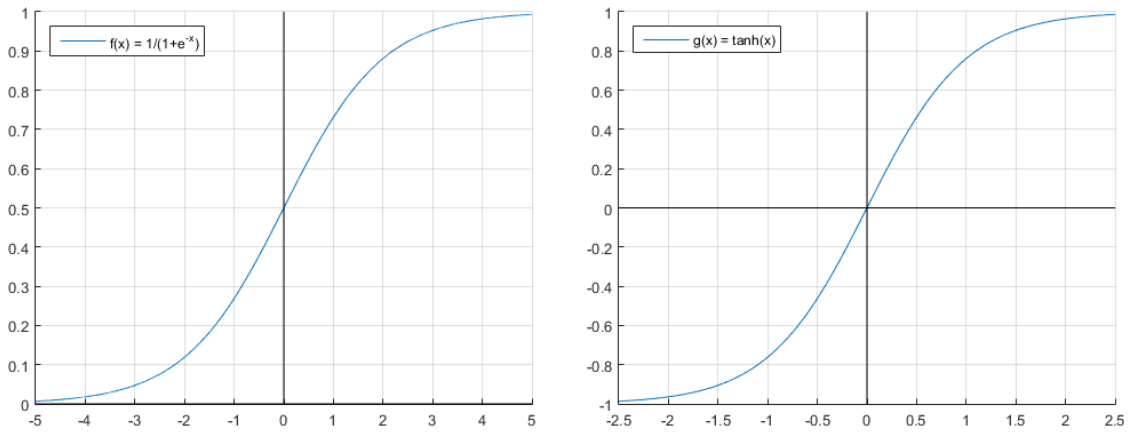

Figure 3.3.: The sigmoid (left) and $tanh(x)$ activation functions. Their advantages over the $sgn(x)$ non-linearity include the ability of taking up arbitrary values within the $[-1, 1]$ range and being differentiable.

With a minor alteration a perceptron can be used for regression purposes too. Their binary output is rather useless in regression tasks. For the unit to be able to have real valued outputs the activation function is swapped from the sign function $sgn(x)$ to the sigmoid or logistic function (equation 3.5) The new non-linearity saturates at -1 and $+1$ similarly to $sgn(x)$ but is also capable of producing values in the range between the two values. By normalizing the input values to be within $[-1, 1]$, the perceptron and its learning algorithm can be re-purposed for regression tasks. Other solutions also exist with similar properties, one such alternative is the hyperbolic tangent function $tanh(x)$.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$

The form of the sigmoid activation function. Its advantages over the $sgn(x)$ non-linearity include the ability of taking up arbitrary values within the $[-1, 1]$ range and being differentiable.

The capabilities of a single artificial neuron are severely limited once classification or regression becomes hard using the raw input data. Connecting several of these units both in parallel and sequentially creates a Neural Network. The artificial neurons connected in parallel form layers and the output of each one of such layer's neurons is passed forward to all of the units of the subsequent layer. The resulting structure is referred to as a Feed Forward Neural Network (FFNN). The first layer of an FFNN is the input layer, the final layer is the output layer with the layers in-between collectively addressed as hidden layers. An example for such an architecture can be found in figure 3.4.

Such systems can process data more efficiently as a decision (the output, be the objective regression or classification) is only rendered at the final layer. As data is passed along the layers various features get extracted from the input data, and this feature representation is passed along to subsequent layers leading to more complex representations being born as the outputs of each stacked layer. As the depth of the network increases, that is more layers are stacked one after the other, the representation passed to the out-

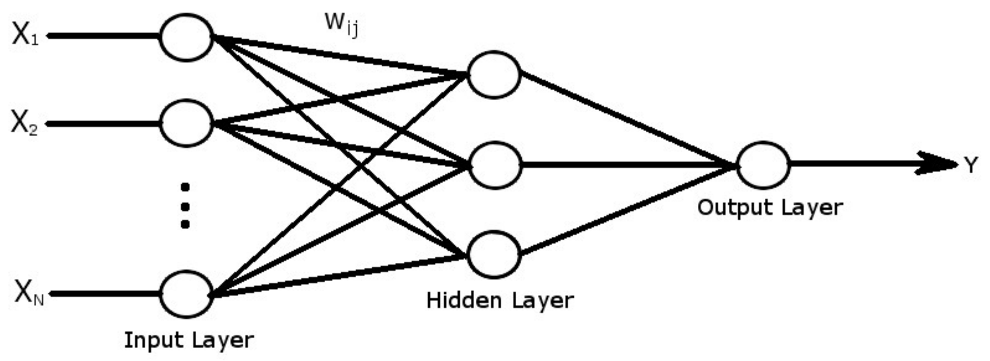

Figure 3.4.: A Feed Forward Neural Network (FFNN) consists of interconnected artificial neurons with their outputs serving as inputs for the subsequent layer. The layers between the input and output layers are referred to as hidden layers

put layer is also deepened which is why the use neural networks with a relatively large amount of layers has been coined as Deep Learning.

Given a feed-forward neural network consisting of L layers with the l-th layer ($l \in \{1, 2, \ldots, L\}$) having N_l neurons, the output of the i-th neuron (($i \in \{1, 2, \ldots, N_l\}$) will be $y_i^{(l)}$. The weights of layer l will be stored in the weight matrix $\overline{W}^{(l)}$. In such a network the weight connecting the i-th neuron of layer $(l-1)$ with the j-th neuron of layer l is $w_{ij}^{(l)}$. The outputs of the l-th layer are in the vector $\overline{y}^{(l)}$. The 0-th layer is considered to be the input, with the neurons in layer 1 receiving it. This dictates that the size of the first layer must be the exact size of the input. Each weight matrix has a pre-set size as all neurons of a layer establish connections with all of the next layer's neurons. Thus the size of a weight matrix connecting layer $(l-1)$ made up of $N_(l-1)$ neurons and layer l of N_l neurons is of dimensions $N_(l-1)$ by N_l. This holds for all layers except for the first layer's inputs where the weights connecting the neurons to the input sample form an identity matrix.

Using the above notation we can obtain the output of layer l, $\overline{y}^{(l)}$, as described in equation 3.6.

$$\overline{y}^{(l)} = \Phi(\overline{W}^{(l)}\overline{y}^{(l-1)} + \overline{b}^{(l)}) \tag{3.6}$$

The output of layer l. Note the existence of separate bias terms for each of the neurons in the FFNN.

The increase in learning capabilities compared to a single artificial neuron is so powerful that a FFNN with only a single hidden layer is capable of performing universal function approximation. It is equipped to realize any mapping between its inputs and outputs should it have enough parameters represented in its weights. This holds true for arbitrary non-linearities as long as the one in use is bounded and non-constant [28].

As data is passed forward in an FFNN the errors created by each neuron are also propagated along and are accumulated by the time the network produces its output. In the case of the perceptron this error was the $(d(k) - y(k))$ term which specified how well (or how badly) the artificial neuron was classifying its inputs. Practically this quantity should be a metric specifying the difference between the expected and actual outputs. In order to be able to efficiently update the weights of a large network we have to change our perspective on the error: as the network is trained, the knowledge necessary to make better conclusions based on the inputs are stored in its parameters. Thus the source of the error are the weights and biases. We can use a so-called cost function, C to express the error in terms of the weights and biases as seen in equation 3.7.

$$C(w) = \sqrt{\sum_i (y(\overline{x})_i - d(\overline{x})_i)^2} \tag{3.7}$$

The mean squared error as a function of the parameters of the network. Note that the biases are incorporated into w in this case.

This approach means that if we change the parameters of the network the error should change as well. If the weights get nudged in the right direction the value of the cost function should decrease, conversely if change is made in the wrong direction the cost would increase [29]. Formalizing this mathematically the net change in the cost, ΔC, depends on the change in a weight, Δw_i, and the effect this would have on C: $\Delta C = \frac{\partial C}{\partial w_i}\Delta w_i$. Taking all the weights into account we get a gradient, ∇C, along which we aim to minimize the cost [30]. In this case the total ΔC depends on all the changes we introduced to the weights, Δw, expanded in equation 3.8.

$$\Delta C = \nabla C \Delta w \ , \ where \ \nabla C = (\frac{\partial C}{\partial w_0}, \frac{\partial C}{\partial w_1}, \dots \frac{\partial C}{\partial w_n}) \tag{3.8}$$

The main goal is to descend along this gradient, hopefully landing in a global minimum as the gradient descent method is capable of getting stuck in local minima. In practice the weights are updated using the back-propagation algorithm [30]: the error created by layers is propagated backward to generate the updates necessary which change the weights of the network. One of the constraints of the back-propagation algorithm is that the activation function must be differentiable as the method employs gradient descent to determine the updates. Fortunately, the widely used sigmoid non-linearity is one such function. To describe the inner workings of the algorithm the fundamental quantity it uses must be defined: the error of a neuron, δ_i^l, as described in equation 3.9.

$$\delta_i^l = \frac{\partial C}{\partial z_i^l}. \tag{3.9}$$

The error (the amount by which it contributes to the cost) of the i-th in layer l depends on its weighted inputs z_i^l.

To be able to propagate the error backwards we must define the error of each layer (δ^l) in terms of the error of the next layer (δ^{l+1}, towards the output layer), as shown in equation 3.10.

$$\delta^l = ((w^{l+1})^T \delta^{l+1})\phi'(z^l)^T \tag{3.10}$$

The error of layer $l+1$ is propagated back along its weights, w^{l+1}, to layer l.

The error of each neuron in the final layer is simply their contribution to the output error. Once we have this quantity we can send it backwards in the network to calculate the layer errors using equation 3.10. Gradient descent is performed by applying updates based on these errors to the corresponding layers. The update rule for weights is described in equation 3.11, whereas the rule for biases is found in equation 3.12. For increased speed it is possible to apply these updates after a batch of input samples, in this case the average loss for the batch counts as the output error [29].

$$w^l_{k+1} = w^l_k - \eta(\delta^l \phi(z^{l-1})^T) \tag{3.11}$$

$$b^l_{k+1} = b^l_k - \eta\delta^l \tag{3.12}$$

Weights and biases of layer l in the update iteration $k+1$. η is the learning rate.

As mentioned above, one of the caveats of gradient descent based methods is the ability of being able to get stuck in local minima. As different weight initializations (specifying the values of w^l_0 and b^l_0 for all $l \in \{1,\dots,L\}$) start the network out in different spots within the high-dimensional parameter space when training, the same network architecture with varying initial weight values can converge to a different point. Choice of the learning rate can help with breaking out of or avoiding getting stuck in local minima as it specifies the step size along the gradient. A gradually decreasing η can help with honing in on the final convergence point.

Choosing the amount of parameters and their distribution amongst the layers greatly influences the performance of the network. Opting for a deeper architecture can increase the complexity of the representation, at the cost of some useful simpler features possibly being lost. Should the network have too few parameters, it will become unable to extract useful information from the input samples and incapable to render reliable decisions at the output. On the flip side, if too many parameters are available, the system will be able to distinguish features characteristic to each of the training samples instead of learning to generalize. The former phenomenon is referred to as underfitting, which can be overcome by either reducing input complexity or increasing the size of the neural network. The latter case is overfitting: it can be reduced by increasing input complexity, increasing the training set size, or hindering the learning of the network during the training process in ways that ensures the emergence of the generalization ability instead of it fitting the noise in the training set.

It is important to consider the task when creating both the network architecture and datasets. The proposed neural network must be able to work well with not just the samples available as the amount of samples available may not only be obscenely small compared to all the possibilities the network may see, but also be non-representative, covering merely a single aspect of the whole input space. Thus it is important not to overfit ourselves to the dataset available per se, misleading ourselves that the dataset does indeed cover all possibilities and should the network perform well on the test set it will be able to handle any real-life situation it is put in.

A further fallacy is the assumption that if the network performs well on the given dataset it should, by all means, achieve a better performance on a more complete one. In this case the network can overfit to the aspect of the whole input space the less complete dataset covered and struggle when presented with more variation in the input samples.

A hindrance applied to reduce overfitting is the introduction of dropout [31] to layers. Applying dropout to a layer can set some of its neurons' activations to zero. The chance of a neuron having its activation 'dropped out' is the same across the layer, $p \in [0, 1]$. This prevents co-adaptation between the units of a layer and the stochastic behaviour it adds to the network is beneficial to its learning ability, although it slows convergence. When the network is not being trained, dropout multiplies the weights of the layer by $1 - p$ to deterministically simulate the stochastic effect dropout has on the layer during training and keep the average weight magnitude of the layer in line with those during training. This ensures the post-training behaviour of the network aligns with the behaviour shown at training time and keeps its predictions consistent.

Another approach is to introduce a regularization term into the cost function. One common regularization strategy is L2-regularization: the L2-norm of the weights is introduced into the loss [29]. With this the objective is not only to minimize the difference between the expected labels and the network outputs but also to keep the weights low. This denies the network from being able to develop neurons which have strong weights for some features only present in the training samples. A small change in weights introduced by getting features inherent for all the possible samples (and consequently describing the task at hand well) wrong can have a relatively high impact on these neurons' activations. As such, regularization also helps with the emergence of generalization ability in the network by keeping the weights low.

Pushing the boundaries of the back-propagation algorithm above is the introduction of the Rectified Linear Unit (ReLU) [32][33]. The ReLU is a non-saturating non-linearity, which takes the form of $\phi(x) = max(0, x)$, as seen in figure 3.5. The violation of the above is made by the function being non-differentiable at $x = 0$. This is disregarded as the computers these networks are run on are digital (a differentiable approximation does exist for analytical purposes), and as such we can specify the slope of the function as zero if $x \leq 0$ and x when $x > 0$ instead of taking the actual derivative of the function. The usefulness of the ReLU lies in that if an input truly is strong enough to elicit a strong response, the activation is not capped at 1. An upgrade to the ReLU is the leaky ReLU where some of the activation is allowed through even if it is below 0, scaled by an $\alpha \in [0, 1]$ constant. In this case the function takes the form $\phi(x) = max(\alpha x, x)$. Some approaches exchange this constant for a learnable parameter of the network where it is adjusted by gradient descent to minimize the loss.

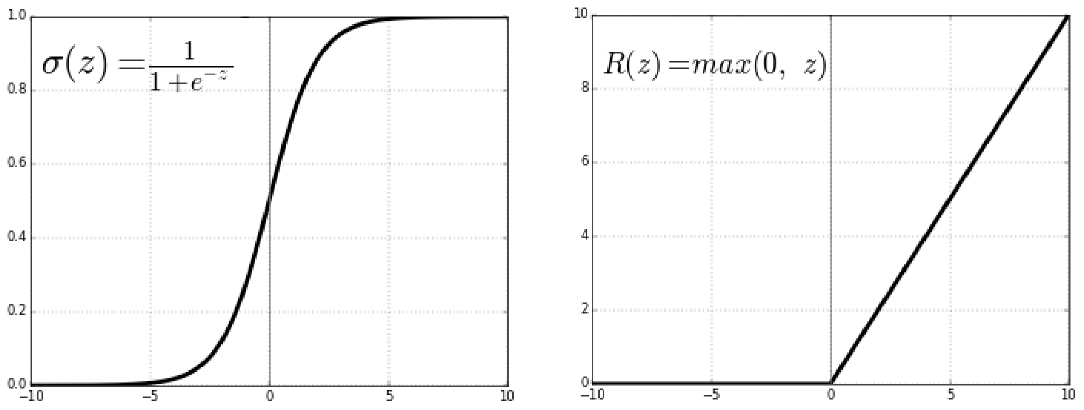

Figure 3.5.: The sigmoid function (left) compared to the Rectified Linear Unit's activation (right). Source: [34]

The network's performance can also be increased by providing the inputs to the network in a more pre-processed format, instead of using raw data, which can help with processing it. A widely adopted approach for image-based data is the use of Convolutional Neural Networks which can learn convolutional filters to apply to the raw image input and feed the filtered image to a FFNN to achieve state-of-the-art performance in various fields [35] [24]. My experience with such networks [36] motivated me to find a trainable pre-processing method instead of using the complex and high-dimensional raw structural data.

The approach I opted for is the so-called Atomic Convolutional Neural Network (ACNN) [4]. One of the biggest limitations when constructing the training dataset is that all inputs to the FFNN must be the same size. This is a heavy hit to the robustness of the system as, purely by itself, an FFNN is unable to deal with molecules of varying sizes. The ACNN architecture alleviates this issue by being able to extract features from varying sizes of molecules, even orders of magnitude apart making it ideal for working with complexes made up of large proteins and comparatively small ligands. This is achieved by focusing on all atoms of each element as a group and considering their total contribution to the potential energy of the molecule instead of account for each and every atom within.

The input to the system is a pre-processed form of the list of all the atoms within a molecule and their respective atomic numbers. Given a molecule consisting of N atoms, described by a vector $\overline{z}$ of dimensions $(N, 1)$ detailing the atomic numbers of each atom along with a matrix C of dimensions $(N, 3)$ detailing the three-dimensional Cartesian co-ordinates of the atoms at the appropriate indices in $\overline{z}$. The main goal of this pre-processing is to reduce the input data complexity. As a first step, the inter-atomic distances are calculated and neighbour listed into a matrix, R, with only the $M << N$ spatially (disregarding the bonds of the molecule) closest neighbours being kept. An element of R ($R_{i,j}$) is the Euclidean distance of the i-th atom from the j-th atom in C: $R_{i,j} = ||C_i - C_j||_2$. Instead of keeping all neighbours for each atom and having a (N, N)

size neighbour listed matrix, this results in R being a matrix being of dimensions (N, M), greatly reducing the computational cost of further processing of the data. Along with this, a matrix Z of the atomic numbers corresponding to the atoms listed is created. Z is of dimensions (N, M) where $Z_{i,j}$ is the atomic number of the atom listed at $R_{i,j}$.

For further ease of processing the matrix R is expanded into a tensor of dimensions (N, M, N_{at}) where N_{at} is the number of distinct atomic numbers within Z, the number of elements found within the molecule. This done through a procedure called Atom Type Convolution (ATC) [4]. The output of the ATC is constructed from the matrices R and Z obtained during the previous step. A convolutional filter (K) with a window size of $(1, 1)$, a stride of $(1, 1)$ and depth of N_{at} is applied (see equation 3.13) to R, with the result being a tensor E, whose layers correspond to the elements of the molecule and one-hot encode their positions $E_{i,j,a} = (K * R)_{i,j}$. The first of N_{at} layers tells us whether the atom in position $E_{i,j}$ is of type 1: if it is, the value at the position corresponding to the atom at $R_{i,j}$ will be 1, otherwise 0. This is repeated for as many layers as the number of available atom types (N_{at}). Layer 2 corresponds to atom type 2, the third layer to type 3, and so on, as shown in figure 3.6.

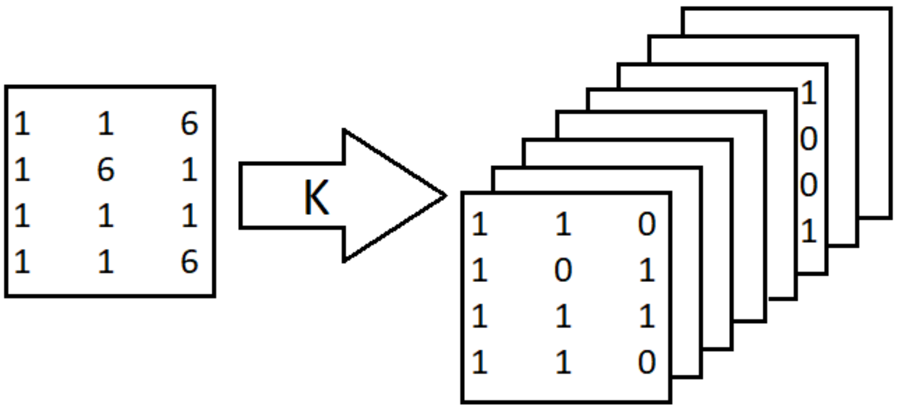

Figure 3.6.: Atom Type Convolution expands the matrix R (left) to a one-hot encoded tensor, E (right).

$$(K * R)_{i,j} = R_{i,j} K^a_{i,j},\ where\ K^a_{i,j} = 1\ if\ Z_{i,j} = a \tag{3.13}$$

The output of the Atom Type Convolution one-hot encodes the atomic numbers of the atoms constituting the molecule described by R and Z.

In order to deal with the variable number of atoms (N) still within the representation sub-sampling is introduced through radial pooling. This operation takes E as its input and applies non-overlapping radial filters over it with receptive fields of $(1, M, 1)$ and depth N_r, where N_r is the desired number of radial filters to apply. These radial filters take the form described in equations 3.14 and 3.15 below.

$$f_s(r_{i,j}) = e^{-\frac{(r_{i,j}-r_s)^2}{\sigma_s^2} f_c(r_{i,j})} \tag{3.14}$$

$$f_c(r_{i,j}) = \begin{cases} \frac{1}{2}cos(\frac{\pi r_{i,j}}{R_c}) & \text{if } 0 < r_{i,j} < R_c \\ 0 & \text{if } r_{i,j} \geq R_c \end{cases} \tag{3.15}$$

R_c is the radial interaction cut-off: atoms outside this radius are considered not to act on the current atom ($r_{i,j}$). Each filter has two tunable parameters: the mean of the Gaussian filter described in equation 3.14 (r_s), and the standard deviation (σ_s). If these parameters are trained during gradient descent, the radial pooling layer will become able to perform feature extraction and binning suited for the dataset currently in use. The output of the radial pooling layer, P, has a shape of (N, N_{at}, N_r). The way to obtain each element using the filters described above is shown in equation 3.16. Finally, the output is flattened into a shape $(N, N_{at}N_r)$.

$$P(i, n_a, n_r) = \beta_{n_r} \sum_{m=1}^{M} f_{s_{n_r}}(E_{i,m,n_a} + b_{n_r}) \tag{3.16}$$

The radial pooling layer applies Gaussian filters to E, with a set scaling parameter (β_{n_r}) and bias b_{n_r}.

The flattened output of a radial pooling layer is similar to the neighbour listed representation of the molecule: each atom is listed with some of its features. In this case these first level (obtained after one radial pooling feature extraction) features are the one extracted by the architecture as opposed to the 0-th level features being the neighbouring atoms and their distances. As such, the ATC and radial pooling can be applied again to obtain a feature set one level deeper than the current one. This combination of these two operations form the basic unit of the ACNN: the Atomic Convolution Layer (ACL), shown in figure 3.7. Once the desired feature depth is reached the flattened output of the final ACL is fed into a FFNN to obtain the final output of the network.

The FFNN receives the features in a row-wise manner as each row in the output of an ACL corresponds to the features of each atom. The task of this tail-end FFNN is to calculate a partial energy of the atom whose features were given as an input. To speed this task up, several weight-sharing FFNNs are instantiated and their partial energy outputs are summed to obtain the total energy: $\sum_i E_i$, where E_i is the output produced for each row of the final ACL output or the partial energy of atom i (figure 3.8. The difference between the total energy according to the ACNN and the actual energy labels is used to tune all trainable parameters to reduce the network's error as much as possible. To achieve this, a dataset containing high quality structural information and reference energy labels must be established.

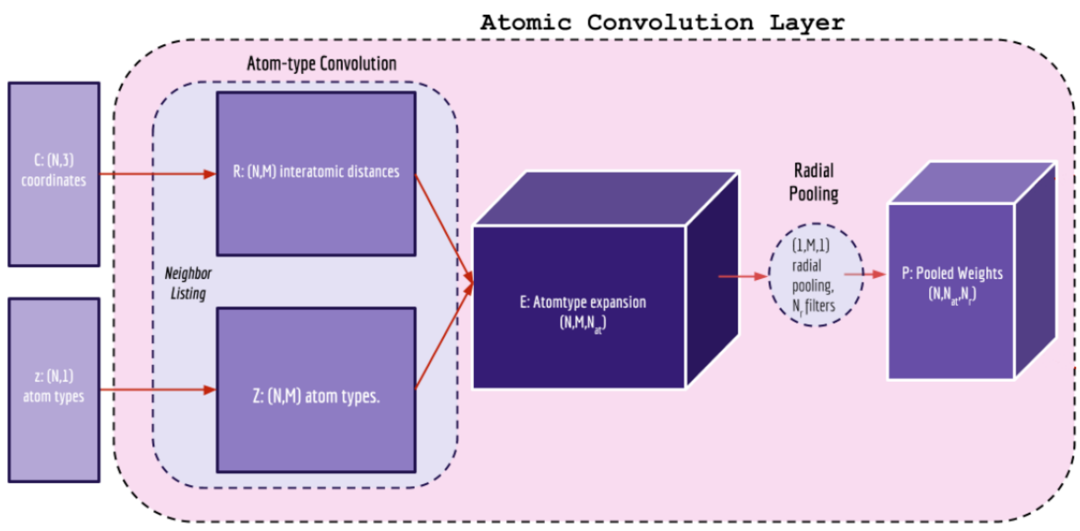
Atomic Convolution Layer
Atom-type Convolution
C: (N,3) coordinates
z: (N,1) atom types
R: (N,M) interatomic distances
Neighbor Listing
Z: (N,M) atom types.
E: Atomtype expansion (N,M,N_{at})
Radial Pooling
(1,M,1) radial pooling, N_r filters
P: Pooled Weights (N,N_{at},N_r)

Figure 3.7.: The Atomic Convolution Layer (ACL). Source: [4]

3.2. Establishing a Dataset

The dataset that would be used with the neural network had to adhere to a set of criteria to prove useful for a machine learning task:

- Contain samples in which small molecules bind to proteins.
- The samples must have information on the types of atoms and their respective three-dimensional positions with relatively large resolution.
- Have a reasonably large amount of these samples without being redundant (i.e. too many of a single protein with several bound small molecules).
- Contain the data in an easily separable manner for the protein, the ligand and the complex.

The first condition is a direct consequence of the task at hand: in order to calculate reaction energies (equation 2.1) we need to obtain the energies of the results of protein-ligand binding reactions, $E_{complex}$. The fourth condition allows us to get the energies of the reactants ($E_{protein}, E_{ligand}$) separately and, if needed, augment the dataset with relative ease by having access to a list of proteins and ligands to work with.

The second constraint ensures the data can be used with the Atomic Convolutional Network described above and that their energies can be calculated with the UFF force-field. Although it seems self-explanatory that this data should exist, there are other ways of describing the protein's structure such as the listing of its amino acid sequence.

In order to ensure that the network performs reasonably well outside the constructed dataset, redundancy should be kept to a minimum. Otherwise the network would only learn how to deal with proteins present instead of being shown a wide variety of them. Although this does not guarantee that the estimator would be able to deal with any input after being trained, it does greatly increase its robustness though.

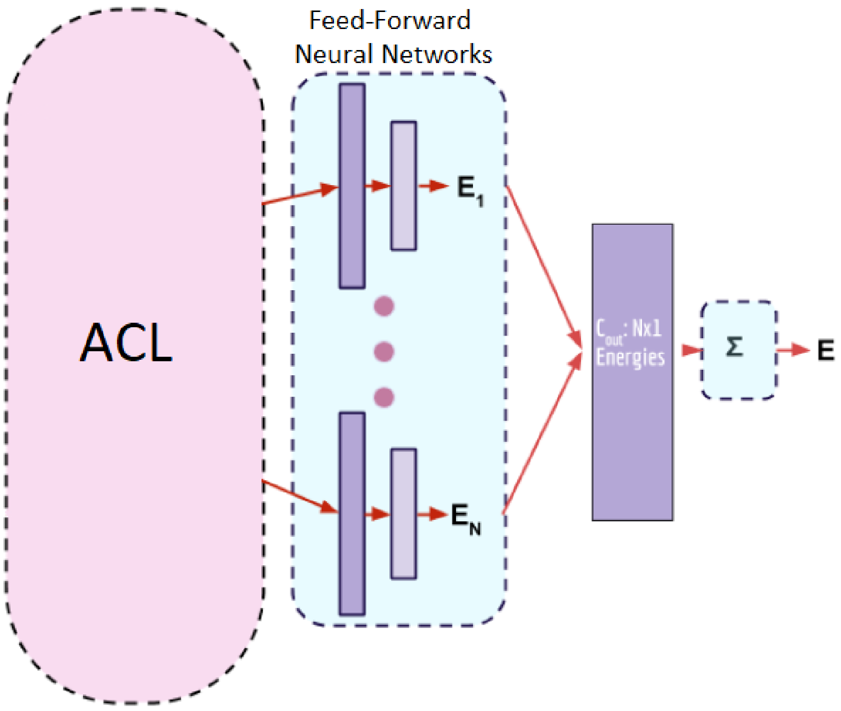

Figure 3.8.: The Atomic Convolutional Network consists of ACLs and tail-end FFNNs. The outputs of the final ACL are fed row-wise into the weights sharing FFNNs, whose outputs correspond to the atom-type energy. The output of the ACNN is the sum of these partial energy terms. Source: [4]

My initial search efforts pointed to the Protein Data Bank (PDB) [7] as the source of the dataset. Its wealth of protein related information seemed like an ideal fit for the task, but to mine it for the samples needed would have been a tremendous overhead as the PDB does not specifically list protein-ligand complexes. Fortunately, quite possibly due the relevance of the task, this effort has already been undertaken and its results are available in the PDBBind Database [15][1].

The PDBBind Database is a subset of the PDB, an annually curated database containing protein-ligand binding information with similar criteria to those set for the dataset to be used with the reaction energy estimator. Of special interest was the PDBBind Refined Set whose entries had to satisfy the following conditions:

- Samples had to be acquired using X-ray crystallography and the structure must have a resolution better than 2.5 Angstroms.
- Only non-covalently bound complexes are accepted.
- The complexes had to be binary, meaning that a single ligand binds to a single protein.
- The equilibrium constants K_d or K_i must be known.
- The ligand must consist of common organic elements C, N, O, P, S, F, Cl, Br, I, and H.

In the initial PDBBind Database (2003) [2] 900 samples made it into the refined set. This work uses the PDBBind 2016 Database in which the refined set contains 4,057 out of the total 13,308 protein-ligand complexes found in the database. The samples are separated into separate protein and ligand structures with both being in the binding position and conformation. The data is stored in the PDB file format [37] for the proteins and in Mol2 [38] and SD [39] for the ligand. Additionally, information regarding the binding site, the so-called pocket region, is also available.

Protein surfaces have several troughs and indentations on them where a ligand could bind (although less common, but possible are cavities as binding sites). These sites can vary in size from 10^2 to 10^3 Å. Larger proteins have more possible pockets but there is no correlation between the size of the protein and the volume of its pockets. Most often the largest binding site is where the ligand binds to the protein [40]

The initial step of creating the dataset was to extract the necessary information from the PDBBind Database. Since the majority of the work would be done in Python, I stored the data in a MongoDB [41] using Python built-in types for ease of conversion into the necessary formats in the future. Using PyMongo, the Python interface of MongoDB, I extracted and stored the following information from the samples:

- The four character, unique PDB ID of the protein-ligand complex.
- Three-dimensional Cartesian co-ordinates in Angstroms of the whole protein, the pocket region and the ligand.
- List of atom types of the whole protein, the pocket region and the ligand.

Following this I used the openbabel [16] python package to calculate the UFF energies of these samples, both for the protein-ligand complex along with the pocket-ligand complex as well. The data I extracted did not explicitly contain the Hydrogen atoms within the molecule, as their position is implied by free valences of atoms they could bind to. In order to get proper energy values for the calculation, I used a tool I coded in C++ to add these Hydrogen atoms to the molecule and optimize its geometry.

The issue with the current state of the dataset as of this step is that it only contains complexes which occur naturally and as a consequence a reaction must have taken place during which the ligand bonded with the protein. All the samples so far are positive, binding ones. The dataset must be augmented to contain negative samples that either do not bind or do so in a non-ideal spot or conformation. This step is to ensure that the neural network is presented with samples whose reaction energies are positive to improve its performance.

When augmenting a dataset it is important to consider the model it is going to be used with. As elucidated above, the data fed to the ACNN will be neighbour listed to reduce the computational burden. This step makes the input data invariant to rotation and translation, thus any augmentation performed in this way creates redundant inputs to the neural network, making it highly prone to overfitting. (Despite this, the neighbour listing is useful as it ensures the network is insensitive to rotational and translational perturbations.) My choice for augmenting the data was to move the ligand to another position close to the protein (within a distance of 1-2 Angstroms) and calculate the relevant energies for that conformation, effectively creating a plausible situation where a ligand would not find the ideal pocket.

The negative samples are generated exploiting the fact that both the protein and ligand co-ordinates are at our disposal independently. The process of this augmentation was done through a Python script written by me. As a first step, the ligand is placed three times the distance of the centre of mass of the protein and the ligand away from the protein on a random axis. Then, utilizing a simple line search algorithm, the conformation corresponding to the minimal energy is found. At every step of the algorithm the ligand is moved both towards and away from its original binding position. The initial step size by which the ligand is shifted is the distance of the centre of mass of the protein and the ligand. In each step the UFF energies are calculated for the forward (towards the original position), current and backward (away from the original location) positions moving the ligand to the position where the calculated energy is minimal for the three positions. Should the current position yield that minimum (signifying that the ligand would enter the protein should we move it forward), the step size is halved and the three energies are calculated again. This is repeated until the step size is 1 Angstrom. Executing this should leave us with ligands in invalid yet feasible locations (figure 3.9) which are useful for creating negative examples for the dataset.

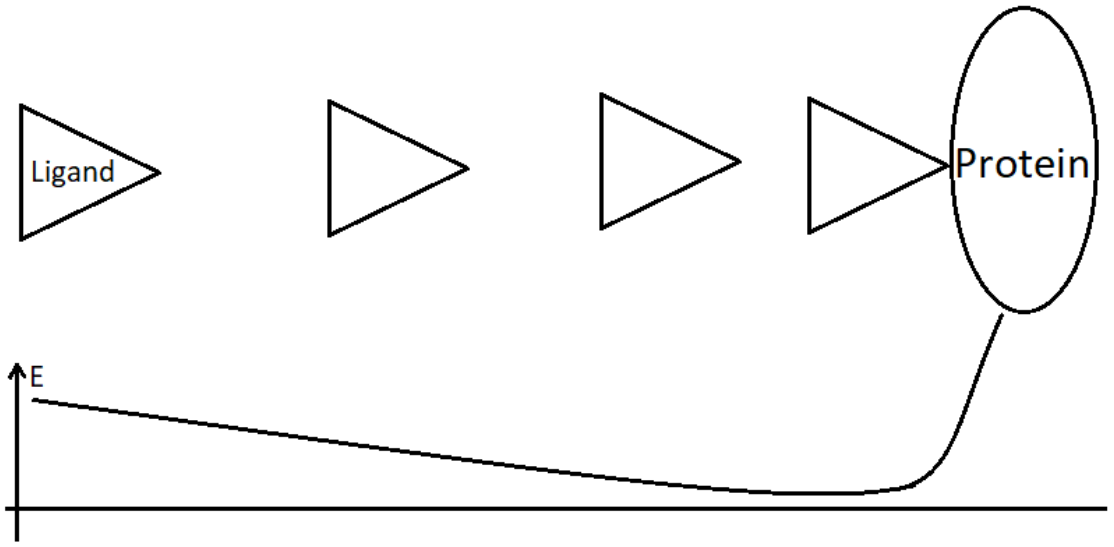

Figure 3.9.: A mock up of the UFF energy versus the position of the ligand relative to the protein. The adaptive step size helps finding the minima before moving the ligand inside the protein at which point the energy explodes.

Pocket regions similar to those of the positive samples should be generated. For this, the radius (the distance of the farthest pocket atom from the ligand's centre of mass) of the original pocket region is considered, following which a pocket of the same radius is cleft from the protein surrounding the centre of mass of the new, negative ligand position. Along with the UFF energy of the negative protein-ligand complex, the pocket-ligand complex also sees its energy calculated and added to the dataset.

Running this algorithm once should leave use with double the samples of the original dataset. I opted not to run it more than once to avoid violating the constraint I set for sample redundancy above. Unfortunately, the Van der Waals typing for some of the molecules failed when calculating the UFF energies, leaving us with less than twice the samples: the final extended dataset contains 6,029 protein-ligand complexes. Splitting this dataset in a 90% to 10% (this deviates from the usual 80% to 20% split to have more training samples since the available amount is quite meagre to begin with) sample ratio between training and test sets, leaving 5,426 complexes for training and 603 samples for testing.

For faster training I chose to run the preprocessing step separately from training the neural network. Using a parallel neighbour listing algorithm the samples were converted to a format the neural network can directly work with and saved to disk for later use. As the samples and their labels are stored separately, it is sufficient to calculate a new set of labels if the task is to be changed instead of having them attached to the samples and requiring the whole pre-processing procedure to be run again. This step was surprisingly time and resource consuming: using a cluster equipped with a 16-core Intel Xeon CPU, the pre-training neighbour listing of the protein-ligand complex dataset took approximately 150 hours even when run using my parallel neighbour listing script with 32 jobs (exploiting the availability of 2 threads per core). Had such hardware not been at hand, running this on a high-end i7 CPU would have taken up to 4 times as much time, about a month in total. An additional step before feeding the inputs to the neural network is the establishment of a logarithmic scale in hopes of increasing the performance of the estimator.

3.3. Neural Network Architecture and Training Strategies

As outlined above, due to the rather low sample size both a specialized architecture and a fitting training strategy is necessary to overcome the inherent potential of overfitting due to a somewhat redundant dataset. Despite the Atomic Convolutional Neural Network being a strong candidate for dealing with molecular data, its main use has only been the prediction of Gibbs free energies of reactions [4]. The architecture used is a three-way predictor with ACNNs in each fork tasked with predicting the energies of their respective inputs: one for each of the protein, the ligand the complex as illustrated in figure 3.10.

Initially the network is trained with the reaction energy in mind (equation 2.1). With each ACNN outputting its energy estimate for its respective compound the whole architecture is trained for the difference between the predicted and reference (as calculated by the UFF module of openbabel). The estimator is trained in batches, with the loss being used is the root mean square error between the two, as described in equations 3.17 and 3.18.

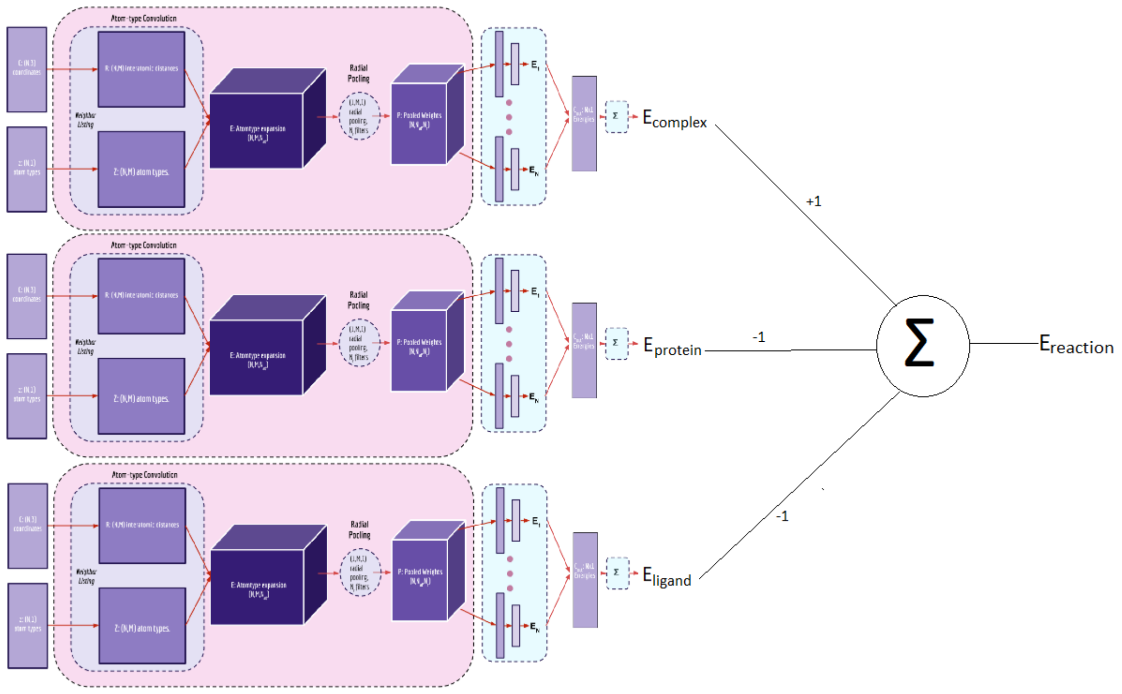

Figure 3.10.: The three-way ACNN architecture for estimating the reaction energy. Each fork deals with its own input, outputting the respective partial energy term which are then summed according to eq. 2.1 to produce the final reaction energy. Source: [4]

$$L(\overline{y}, \overline{d}) = \sqrt{\frac{1}{n}\sum_{i} = 1^{n}(y_i - d_i)^2}, \text{ where} \tag{3.17}$$

$$y_i = E_{complex_i} - E_{ligand_i} - E_{protein_i} \tag{3.18}$$

The root mean squared error cost the estimator is trained with. The outputs of the three-way ACNN forks ($E_{complex_i}, E_{ligand_i}, E_{protein_i}$) are the estimated energies for each of the compounds in a complex. From these the estimated reaction energy (y_i) is calculated according to eq. 2.1 and is compared to the label (d_i).

To help with the complexity of the input data the network is trained with the pocket-ligand complex. However, with such a low amount of samples severe overfitting is expected, especially when the size of a single input is also and order of magnitude lower (the average protein size in the dataset is approximately 8,000 atoms, while the cleft pockets are comprised of hundreds of atoms at most).

In another strategy, based on the availability of the UFF energies for each of the compounds present in the reactions, three ACNNs were trained for the estimation of merely the potential energies of their inputs. This has been done to test and ensure that the architecture is truly capable of handling the approximation of UFF energies.

In all above cases, to help deal with overfitting, L2 regularization has been applied with a penalty multiplier of 0.001 to bring the additional loss generated by this term in line with root mean square error magnitudes of the system. Furthermore, dropout has been applied to all layers (excluding the connection between the output and the final FFNN layer) to the tail end fully-connected layers of the ACNNs with a dropout probability of 0.5.

Training has been done using the Tensorflow (version 1.3) [42] and Deepchem [43] libraries within Python 2.7 using an NVidia GeForce GTX 1080Ti GPU on the Kubuntu 16.04 operating system. The parameters and corresponding training results are detailed below in the Results section. All source code files necessary for reproducing these results have been attached (either on the disk in the back sleeve or in the on-line repository this work has been uploaded to) and listed in the appendix, although some parameters might need adjustments due to differences in the hardware and software environment.

4. Results

All the training and testing phases for each of the aforementioned strategies have been executed with a standardized architecture. This was done to ensure that the results would be comparable and that valid observations could be deduced from them. The parameters and topology of this Atomic Convolutional Network is described below.

The architecture employs a single Atomic Convolutional Layer. This is due to concerns of losing viable features should multiple layers be stacked as each new layer takes the outputs of the previous one as raw inputs, instead of considering them as already processed features. Also of note is that stacking ACLs greatly increased the memory necessary and computational complexity of the neural network architecture because the extracted representation would also have to be expanded by the Atom Type Convolution. The layer has 12 radial pooling features, whose initial parameters can be seen in table 4.1. The radial cut-off parameters were chosen to span the 1 to 12 Angstroms range linearly, except for the first 1.5 Angstroms cut-off.

Radial Cut-off (Angstroms)	Initial Mean (Trainable)	Initial Variance (Trainable)
1.5	0.0	0.4
2.0		
3.0		
4.0		
5.0		
6.0		
7.0		
8.0		
9.0		
10.0		
11.0		
12.0		

Table 4.1.: The initial parameters of the twelve radial pooling filters employed by the Atomic Convolutional Layer.

Following this ACL, the tail end FFNN had three layers of sizes 128, 64 and 32 units each. Each layer employed the ReLU non-linearity described above and had dropout applied with a dropout probability of 0.4 to help with overfitting. In addition to this, their weights had L2 regularization in place with a penalty multiplier of 0.001. The networks were trained with a learning constant (η) of 0.001 in batches of 10 samples per input batch.

The loss used was the root mean square error (equation 3.17). The training labels were modified from the original UFF energy outputs (in units of kcal/mol) by establishing a logarithmic scale. This assists the system in approximating the energies as only the order of magnitude would have to be estimated. This has a negligible impact on using the reaction energy for deciding if two compounds bond as the signs of the energies of all three reactants are left intact. The loss of the system is to be interpreted as the difference in orders of magnitude between the actual (according to UFF) and predicted energies.

4.1. Reaction Energy Strategy

As described above in chapter 3, Methods, this strategy involved the use of a three-way architecture of ACNNs (figure 3.10) with the hopes of reproducing the architecture and results detailed within the article by Pande et al. [4]. Within the loss function, the prediction was the reaction energy comprised of the three networks' outputs as described in equation 2.1. The architecture was trained against the reference UFF reaction energy for 40 epochs. The initial training loss was 3.4184 and the best performance was produced in the 35th epoch with 2.54379 average root mean square error for the batches within as shown in figure 4.1 and table 4.2.

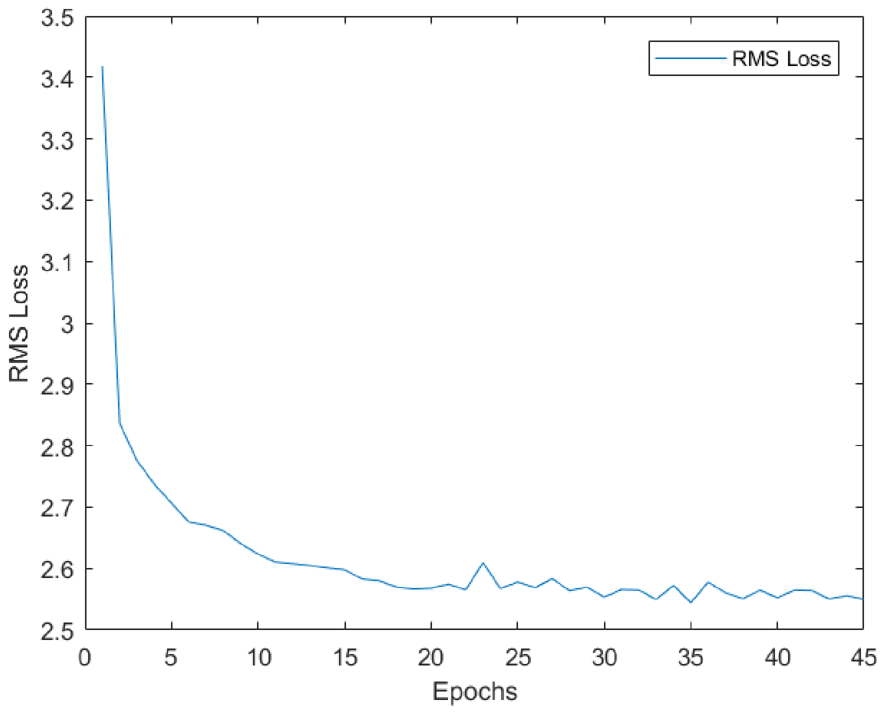

Figure 4.1.: The per epoch average RMS loss graph of the reaction energy strategy. The per epoch values can found in table 4.2

Well observably on figure 4.2, the architecture had difficulties with this strategy. The oscillatory nature of the losses and the minor drop in loss between the first epoch (3.4184) and the both of the final epoch (2.55232) and that of the best performing epoch (2.54379) are indications of this. To further compound this, the test loss was 5.18163 suggesting that the network was either unequipped to deal with the task of estimating reaction energies directly, or that the dataset was unsuitable (possibly due to the low amount of samples available).

To test the source of this, the three-way architecture was separated into three separate ACNNs and each of them was trained separately for the ligand, pocket and complex energies respectively.

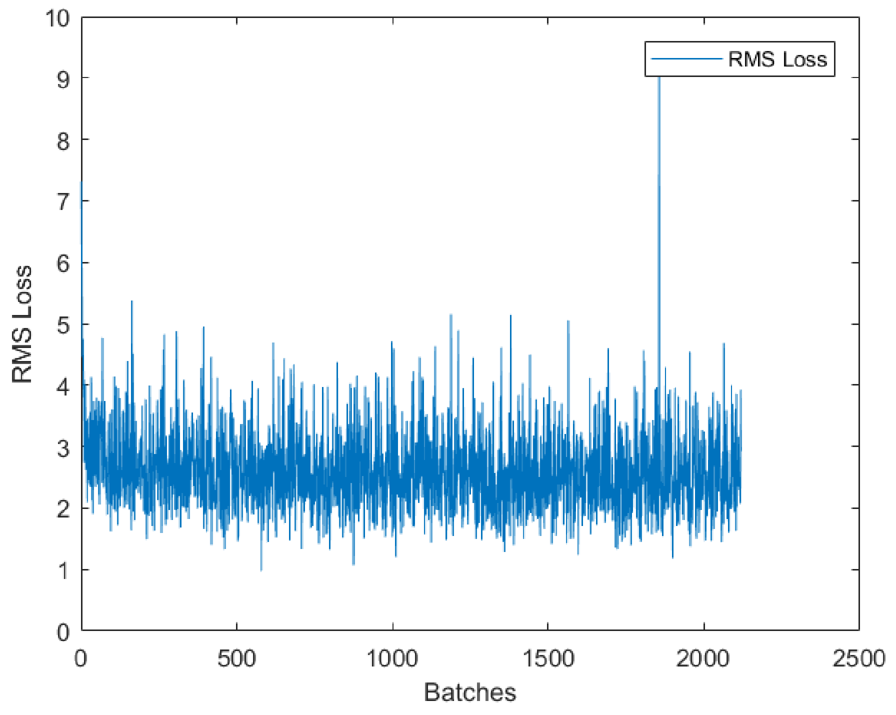

Figure 4.2.: The per batch average RMS loss graph of the reaction energy strategy. Of note is the oscillatory nature and high variance shown for samples even in later epochs.

4.2. Separate Energy Strategy

With the availability of the UFF energy labels for each compound, the architecture in the previous strategy could be separated into separate ACNNs and trained for the difference of their outputs and the reference UFF energy of the input compounds. The goal of this strategy was to decide if the ACNN approach is suitable at all for use with the current dataset. With full knowledge that the dataset is severely limited in size, heavy overfitting is to be observed if the network is capable of dealing with the inputs. Same as before, the loss used was the RMS loss with the established logarithmic scale to be able to compare the results of the two strategies and also of the separate networks within this strategy. All the ACNNs were trained for 20 epochs.

For the ligand, the loss of the first epoch has been 1.58574 decreasing until the final epoch with an average RMS loss of 0.827134 as seen in figure 4.3 and table 4.3. These per epoch loss scores do not properly describe the performance of the network, however.

Epoch	Average RMS Loss	Epoch	Average RMS Loss
1	3.4184	21	2.57412
2	2.83657	22	2.56516
3	2.77577	23	2.60943
4	2.73739	24	2.56718
5	2.70627	25	2.57784
6	2.67538	26	2.56863
7	2.67023	27	2.58318
8	2.66137	28	2.56361
9	2.64029	29	2.56936
10	2.62317	30	2.55331
11	2.61037	31	2.56553
12	2.60775	32	2.56466
13	2.60461	33	2.54958
14	2.6009	34	2.57194
15	2.59779	**35**	**2.54379**
16	2.58283	36	2.57764
17	2.57974	37	2.56043
18	2.56961	38	2.55077
19	2.56636	39	2.56446
20	2.56794	40	2.55232

Table 4.2.: The per epoch average RMS losses for the reaction energy strategy. The epoch with the least loss is marked with boldface.

The per batch losses (figure 4.4) are significantly more promising than in the reaction energy strategy, but the spikes are still present, raising the per epoch average. The test loss of the network was 2.8524, suggesting severe overfitting.

The pocket region network shows similar performance to that of the ligand network. The loss decreases until the final epoch starting from 1.95574 and ending at 0.961833, with the final epoch showing the best performance (figure 4.5 and table 4.3). The test loss was 3.365778, also pointing at overfitting. Examining the per batch losses in figure 4.6, the same spikes are present, but the network has managed to decrease their amplitude over time, suggesting it is capable of effectively tackling the outliers within the samples.

As for the complex, the performance is similar to the above cases. The loss sees a steady decrease starting from 2.75003 with a minimum in the 18th epoch with a value of 1.66022. Overfitting is also present, shown by an average test error of 4.150045. The performance of this network can be observed in figures 4.7 and 4.8.

The overfitting each of these networks produce is detrimental to the test performance but is an outcome that comes as no surprise due to the severely limited amount of samples available. The reaction energy strategy showed that expecting the networks to deduce that they should approximate their respective inputs' contribution to the reaction energy might have been too much. On the other hand, despite being undesired, the overfit of these networks when trained separately proves that using machine learning approaches

Epoch	Ligand Avg. RMS Loss	Pocket Avg. RMS Loss	Complex Avg. RMS Loss
1	1.58574	1.95574	2.75003
2	1.00114	1.2992	2.05567
3	0.954824	1.23426	1.97165
4	0.916143	1.17544	1.93014
5	0.882049	1.12592	1.88448
6	0.872508	1.12146	1.87869
7	0.866061	1.06058	1.84918
8	0.863618	1.0505	1.82897
9	0.850289	1.00463	1.81132
10	0.84927	1.02593	1.81527
11	0.84362	1.01192	1.76157
12	0.836237	1.00517	1.74612
13	0.840697	1.00612	1.70823
14	0.834516	0.986699	1.69793
15	0.837238	1.01403	1.69523
16	0.832967	0.975913	1.70882
17	0.836428	0.988384	1.68719
18	0.829633	0.99173	**1.66022**
19	0.830343	0.984088	1.67531
20	**0.827134**	**0.961833**	1.67173

Table 4.3.: The per epoch average RMS losses for the separate energy strategy. The best epochs with the best performance have their loss values shown in boldface.

to speed up energy approximation is viable and that the Atomic Convolutional Neural Network is a more than capable approach to estimating reaction energies. When trained separately, the average difference from the original UFF energies (as we used a logarithmic scale) was within an order of magnitude for both the ligand and the pocket region further strengthening the argument for the use of such methods for energy estimation for greatly varying sizes of input molecules.

I believe that the use of ACNNs for this purpose holds plenty of promise and has a lot of untapped potential for the future. A handful of ideas are outlined below amongst the enhancement proposals in the final chapter, Conclusion.

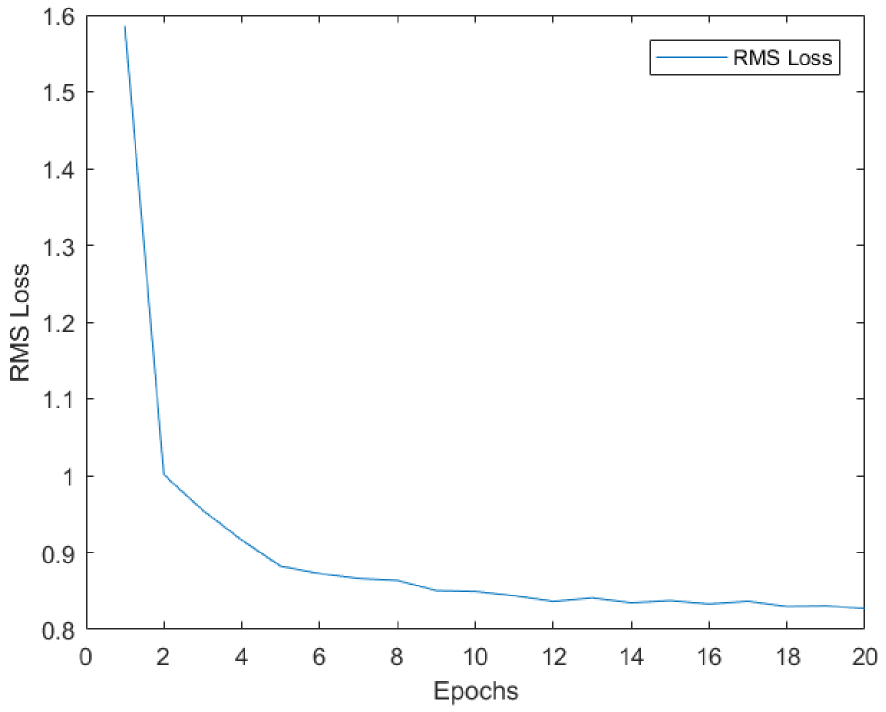

Figure 4.3.: The per epoch average RMS loss graph of the ligand network in the separate energy strategy. The per epoch values can found in table 4.3

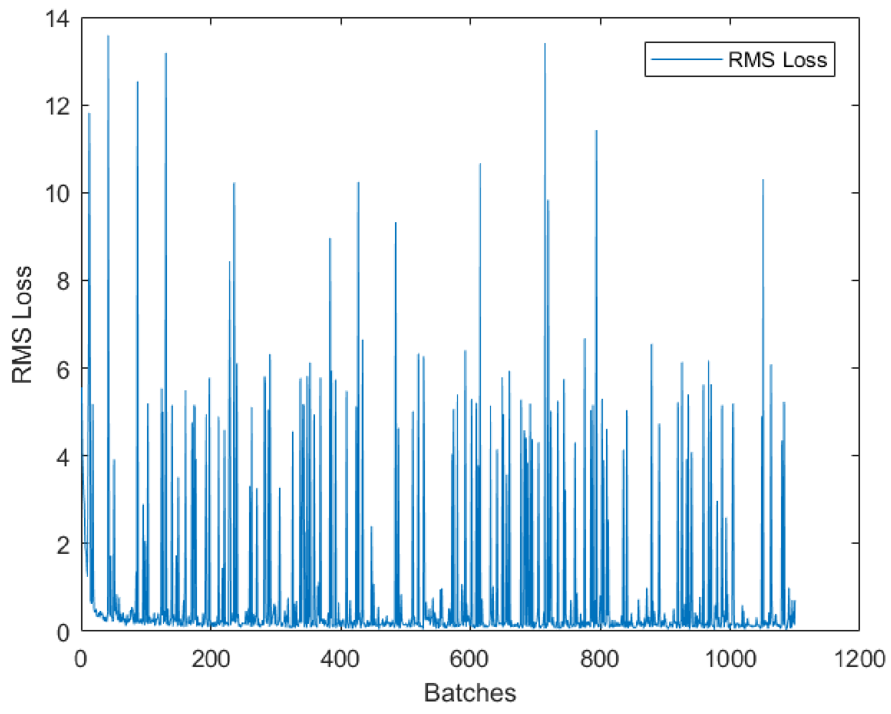

Figure 4.4.: The per batch average RMS loss graph of the ligand network in the separate energy strategy.

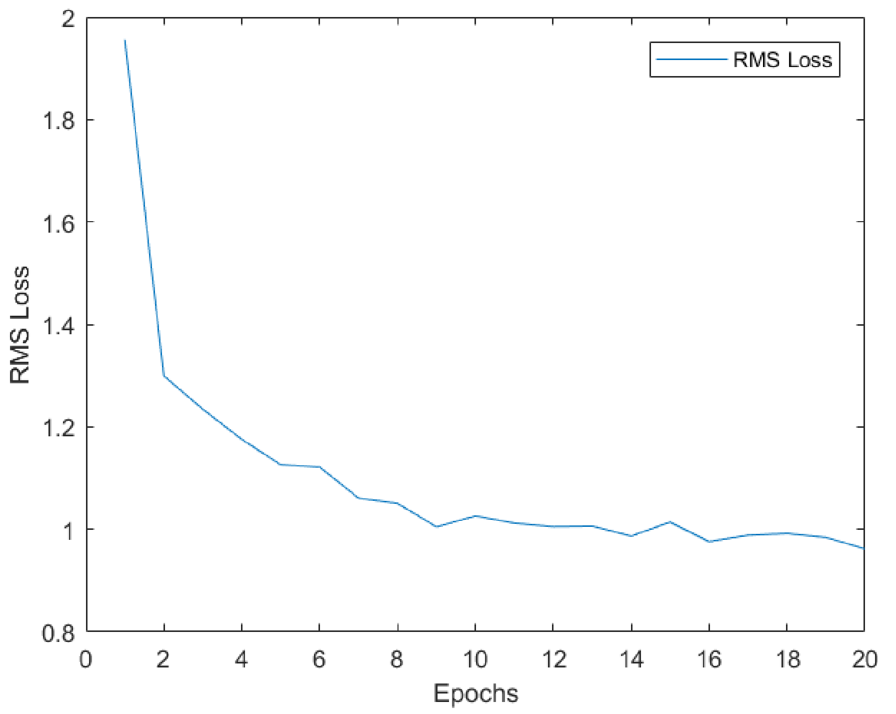

Figure 4.5.: The per epoch average RMS loss graph of the pocket network in the separate energy strategy. The per epoch values can found in table 4.3

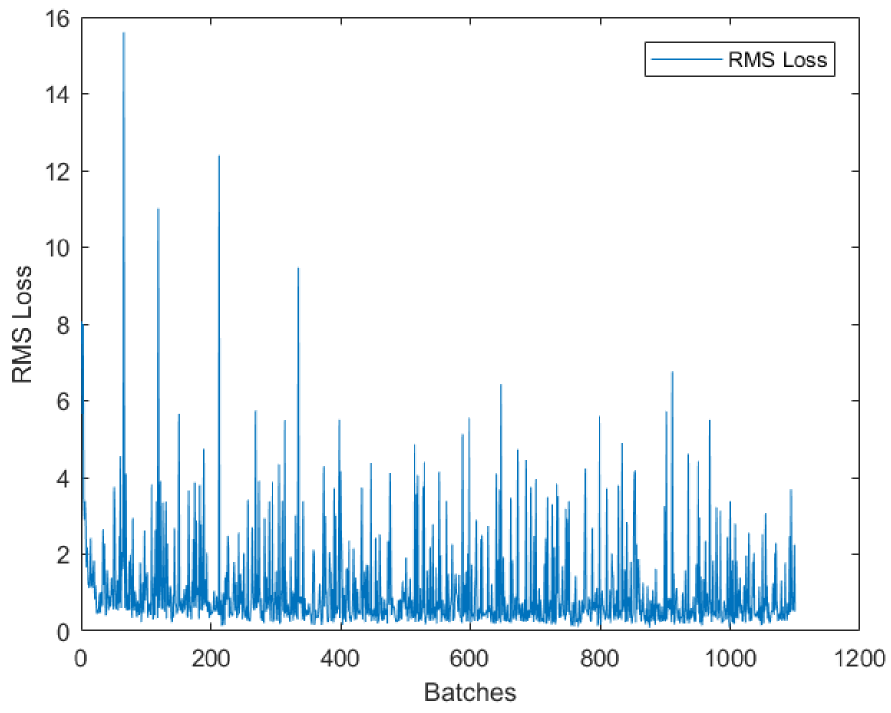

Figure 4.6.: The per batch average RMS loss graph of the pocket network in the separate energy strategy. The spike magnitudes are decreasing over time, suggesting the network learns to deal with outliers.

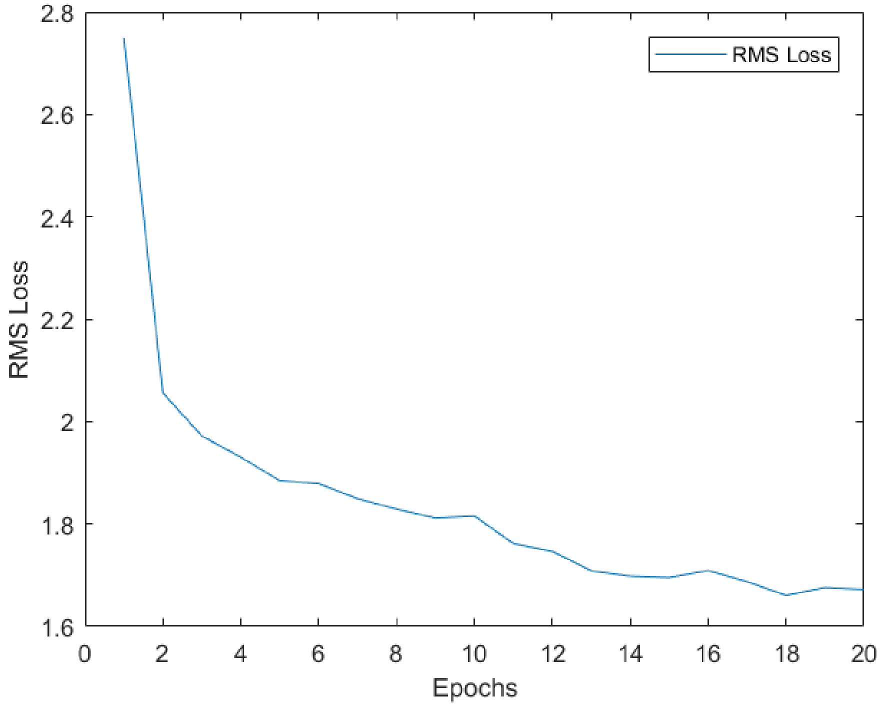

Figure 4.7.: The per epoch average RMS loss graph of the pocket-ligand complex network in the separate energy strategy. The per epoch values can found in table 4.3

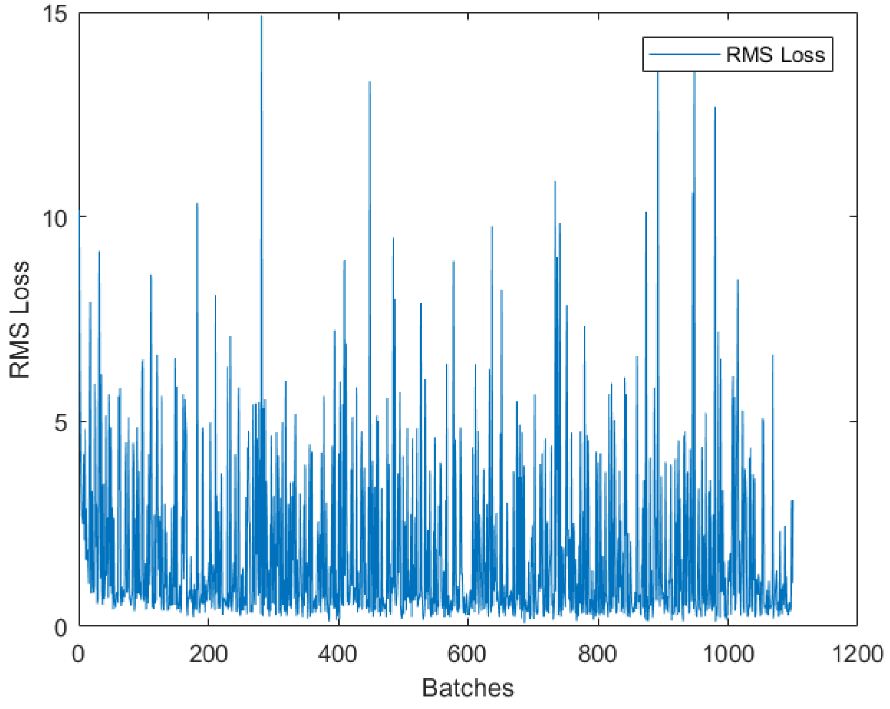

Figure 4.8.: The per batch average RMS loss graph of the pocket-ligand complex network in the separate energy strategy.

5. Conclusion

I have succeeded in completing the tasks at hand and solving a machine learning task relevant for Computer Aided Drug Design. The Atomic Convolutional Neural Network [4] architecture was shown capable of dealing with various compounds of differing sizes, although its performance in this case was held back by a low amount of samples. I compared the results of two training strategies on a dataset constructed from samples available in the PDBBind 2016 Database [2] and augmented using methods I have implemented and optimized. While evaluating the results of both training strategies various enhancement ideas have come to mind, a few of which I'd like to present.

An idea stemming from reviewing the results of the strategies above would be to use the separately trained ACNNs as components in the first strategy where they were trained against the reaction energy. It seemed clear that three-way architecture had difficulties in determining its actual goal (estimating the reaction energy) without its components being told their subtasks (estimating the respective potential energies) beforehand. By pre-training these networks for a few iterations (ideally stopping right before overfitting becomes dominant) and then following up by a longer training session for the reaction energy each ACNN could not only perform better, but possibly take over somewhat for the other two networks and reduce error in unexpected ways due to the inherent plasticity present in neural networks. The effectiveness of this method also scales with the amount of data available, as more pre-training iterations are possible for the ACNNs without overfitting setting in.

It would seem trivial that to increase the performance of the system more samples should be made available both for training and evaluation. Seeking out databases similar in structure to the PDBBind Database [2] while also gathering new samples from the growth of currently known datasets (such as PDBBind itself) would decrease overfitting in the network and increase its robustness in dealing with families of samples not found in the dataset at all. But acquiring the three-dimensional structure of these samples requires experimental efforts which can ensure high quality samples such as X-Ray Crystallography [8], which is a high-cost process (and some compounds can't be crystallized at all), making this far from trivial.

All the above proposals have a common requirement: the necessity of high amounts data to train on. It is not far-fetched to assume that if an abundantly large dataset would be available, similar or better results could be achievable through the use of less specialized approaches. As such, if these results would be iterated on with improvement in mind the first step should be the expansion of the dataset with new samples.

An approach exploiting the increased performance when trained for the reactants separately would be to use a single protein and its several possible ligands as a dataset. This is actually more appropriate for a CADD task as lead detection consists of finding fitting drug candidates for a single protein, although this would mean re-establishing a dataset for each protein and its ligands and re-training the architecture. The resulting system would be highly specialized for each protein instead of being a be-all-end-all solution for all possible proteins and ligands.

I am hopeful that the work of both doctors and researchers can be assisted by the integration of machine learning methods into medicine or pharmaceutical research. Neural networks have already shown their prowess in various fields, including but not limited to Computer Vision [24], Bio-informatics [44] and Biomedical Signal Processing [45]. The insight I gained during my studies and while working on this thesis points towards that they indeed can find their place in these fields as well and hold plenty of as-of-yet untapped potential for the future.

Bibliography

[1] Z. Liu, Y. Li, L. Han, J. Li, J. Liu, Z. Zhao, W. Nie, Y. Liu, and R. Wang, "PDB-wide collection of binding data: current status of the PDBbind database", *Bioinformatics*, vol. 31, no. 3, pp. 405–412, 2015. DOI: `10.1093/bioinformatics/btu626`. eprint: `/oup/backfile/content_public/journal/bioinformatics/31/3/10.1093_bioinformatics_btu626/2/btu626.pdf`. [Online]. Available: `+http://dx.doi.org/10.1093/bioinformatics/btu626`.

[2] R. Wang, X. Fang, Y. Lu, C. Y. Yang, and S. Wang, "The PDBbind database: Methodologies and updates", *J. Med. Chem.*, vol. 48, no. 12, pp. 4111–4119, 2005, ISSN: 00222623. DOI: `10.1021/jm048957q`.

[3] A. K. Rappé, C. J. Casewit, K. S. Colwell, W. Goddard III, and W. Skiff, "UFF, a Full Periodic Table Force Field for Molecular Mechanics and Molecular Dynamics Simulations", *J. Am. Chem. Soc.*, vol. 114, no. 25, pp. 10 024–10 035, 1992, ISSN: 0002-7863. DOI: `10.1021/ja00051a040`.

[4] J. Gomes, B. Ramsundar, E. N. Feinberg, and V. S. Pande, "Atomic Convolutional Networks for Predicting Protein-Ligand Binding Affinity", pp. 1–17, 2017, ISSN: 0148396X. DOI: `10.18653/v1/P16-1228`. arXiv: `1703.10603`. [Online]. Available: `http://arxiv.org/abs/1703.10603`.

[5] C. M. Song, S. J. Lim, and J. C. Tong, "Recent advances in computer-aided drug design", *Brief. Bioinform.*, vol. 10, no. 5, pp. 579–591, 2009, ISSN: 14675463. DOI: `10.1093/bib/bbp023`.

[6] F Ooms, "Molecular modeling and computer aided drug design. Examples of their applications in medicinal chemistry.", *Curr. Med. Chem.*, vol. 7, no. 2, pp. 141–158, 2000, ISSN: 09298673. DOI: `10.2174/0929867003375317`.

[7] (). RCSB Protein Data Bank, [Online]. Available: `https://www.rcsb.org/pdb/home/home.do` (visited on 11/28/2017).

[8] (). RCSB Protein Data Bank, [Online]. Available: `http://www.rcsb.org/pdb/statistics/holdings.do` (visited on 11/28/2017).

[9] (). Amazon EC2, [Online]. Available: `https://aws.amazon.com/ec2/` (visited on 11/28/2017).

[10] B. Booth. (2017). Four Decades Of Hacking Biotech And Yet Biology Still Consumes Everything, [Online]. Available: https://www.forbes.com/sites/brucebooth/2017/04/26/four-decades-of-hacking-biotech-and-yet-biology-still-consumes-everything/#5d282427779e (visited on 12/03/2017).

[11] (2017). Horizon 2020 - European Commission, [Online]. Available: https://ec.europa.eu/programmes/horizon2020/ (visited on 12/03/2017).

[12] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network", in *Advances in neural information processing systems*, 1990, pp. 396–404.

[13] R. W. Brause, "Medical analysis and diagnosis by neural networks", pp. 1–13, 2001.

[14] T. Sun, "Spam Filtering based on Naive Bayes Classification", *Archive of research papers at Babes Bolyai University*, 2009.

[15] P.-C. database. (2017). Welcome to PDBbind-CN Database, [Online]. Available: http://www.pdbbind.org.cn/ (visited on 11/30/2017).

[16] (2017). Open Babel, [Online]. Available: http://openbabel.org/wiki/Main_Page (visited on 11/30/2017).

[17] P. Hohenberg and W. Kohn, "Inhomogeneous electron gas", *Physical review*, vol. 136, no. 3B, B864, 1964.

[18] S. J. Clark. (2003). Bond Stretching Interactions, [Online]. Available: http://cmt.dur.ac.uk/sjc/thesis_dlc/node72.html (visited on 11/29/2017).

[19] ——, (2003). Bond Angle Bending Interactions, [Online]. Available: http://cmt.dur.ac.uk/sjc/thesis_dlc/node73.html (visited on 11/29/2017).

[20] (). change-in-tetrahedral-bond-angle.png (PNG Image, 208 by 278 pixels), [Online]. Available: http://images.tutorvista.com/cms/images/44/change-in-tetrahedral-bond-angle.png (visited on 12/02/2017).

[21] ——, (2003). Torsional Angle Interactions, [Online]. Available: http://cmt.dur.ac.uk/sjc/thesis_dlc/node74.html (visited on 11/29/2017).

[22] (). (PNG Image, 175 by 333 pixels), [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/9/97/ (visited on 12/02/2017).

[23] ——, (2003). Non-bonded Interactions, [Online]. Available: http://cmt.dur.ac.uk/sjc/thesis_dlc/node75.html (visited on 11/29/2017).

[24] (2017). Imagenet Large Scale Visual Recognition Challenge 2017, [Online]. Available: http://image-net.org/challenges/LSVRC/2017/results (visited on 12/03/2017).

[25] (2016). Researching for tomorrow | DeepMind, [Online]. Available: https://deepmind.com/applied/deepmind-health/working-nhs/health-research-tomorrow/ (visited on 12/03/2017).

[26] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.", *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[27] M. Collins. (). Convergence Proof for the Perceptron Algorithm, [Online]. Available: `http://www.cs.columbia.edu/~mcollins/courses/6998-2012/notes/perc.converge.pdf` (visited on 11/28/2017).

[28] K. Hornik, "Approximation Capabilities of Muitilayer Feedforward Networks", *Neural Networks*, vol. 4, pp. 251–257, 1991.

[29] M. Nielsen, *Neural Networks and Deep Learning.* 2017. [Online]. Available: `http://neuralnetworksanddeeplearning.com/` (visited on 11/23/2017).

[30] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors", *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models", in *Proc. ICML*, vol. 30, 2013.

[33] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[34] (). 1*XxxiA0jJvPrHEJHD4z893g.png (PNG Image, 726 by 292 pixels), [Online]. Available: `https://cdn-images-1.medium.com/max/1600/1*XxxiA0jJvPrHEJHD4z893g.png` (visited on 12/03/2017).

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[36] A. Godó, "GPU based Machine Learning Methods in Medical Image Processing", 2016.

[37] R. P. D. Bank. (2017). PDB File Format, [Online]. Available: `https://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html` (visited on 11/30/2017).

[38] (). Tripos Mol2 File Format, [Online]. Available: `http://chemyang.ccnu.edu.cn/ccb/server/AIMMS/mol2.pdf` (visited on 11/30/2017).

[39] (). MDL MOLfiles, RGfiles, SDfiles, Rxnfiles, RDfiles formats, [Online]. Available: `https://docs.chemaxon.com/display/docs/MDL+MOLfiles%2C+RGfiles%2C+SDfiles%2C+Rxnfiles%2C+RDfiles+formats#MDLMOLfiles,RGfiles,SDfiles,Rxnfiles,RDfilesformats-sdfSDFiles` (visited on 11/30/2017).

[40] J. Liang, C. Woodward, and H. Edelsbrunner, “Anatomy of protein pockets and cavities: Measurement of binding site geometry and implications for ligand design”, *Protein Sci.*, vol. 7, no. 9, pp. 1884–1897, 1998, ISSN: 09618368. DOI: 10.1002/pro.5560070905. [Online]. Available: http://doi.wiley.com/10.1002/pro.5560070905.

[41] (2017). MongoDB for GIANT Ideas | MongoDB, [Online]. Available: https://www.mongodb.com/ (visited on 11/30/2017).

[42] (2017). TensorFlow, [Online]. Available: https://www.tensorflow.org/ (visited on 12/02/2017).

[43] (2017). DeepChem, [Online]. Available: https://deepchem.io/ (visited on 12/02/2017).

[44] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, “Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning”, *Nature biotechnology*, vol. 33, no. 8, pp. 831–838, 2015.

[45] S. Kiranyaz, T. Ince, and M. Gabbouj, “Real-time patient-specific ECG classification by 1-D convolutional neural networks”, *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 3, pp. 664–675, 2016.

Appendix A.

This work is also available in digital form (as a PDF file) on the CD found in the sleeve on the back cover. Furthermore it contains the source code necessary to reproduce the results detailed within. The files and their function is as below (in the order of sensible use):

- *create_db.py*: creates a MongoDB from the compressed form of the PDBBind 2016 Database (also bundled as *dataset.npy*)
- *gen_neg_ex.py*: uses a line search algorithm to create negative samples from the MongoDB
- *pocketize.py*: cleaves pocket regions from the negative examples
- *pocket_energy.py*: calculates UFF energies for the cleft pocket regions
- *dbproc.py*: pre-processes and neighbour lists samples for use with the ACNN
- *own_models.py*: some models based on the ACNN architecture, including the separate energy one
- *train_nn.py*: deals with training and testing the selected model