Approximating the energy of molecules from their three-dimensional representations using neural networks

Mansur Nurmukhambetov (s4774841)
Thomas Rey (s3977315)
Julius Wagenbach (s3201384)
Matthijs Jongbloed (s4357833)

July 2022

Abstract

In this project we tried to predict the potential energy of organic molecules using feed forward neural networks. The traditional methods of computing the energy are computationally expensive and the cost increases with the number of atoms in the molecules. Neural networks are great function approximators that compute comparatively quickly, which makes using them a promising approach. The input data used are the molecule confirmations, position of the atoms in 3D space. By rephrasing the problem as a constraint satisfaction problem, we reduced confirmations to the number of bond pairs. We did hyperparameter tuning to find the right model architecture and parameters. Then, we trained a multilayer perceptron to solve the regression problem of finding the energy of a molecule. Lastly, we reported that the model has a mean absolute error of ± 1 for big molecules and discussed potential improvements.

Contents

1 Introduction

In chemistry, properties of molecules, such as their potential energy, are used in predicting the outcomes of chemical reactions, which is important in molecule synthesis research. However, traditional computational methods can take from hours to days to achieve accurate energy level approximations for particularly complex molecules.

Erwin Schrödinger introduced the Schrödinger equation to unite properties of electrons and energy in a quantum-mechanical system. One implication of the equation is that solutions to the equation exist only for certain values of energy. Therefore, electrons are quantized — an electron can only possess discrete amounts of energy in an atom, which are called its energy levels.

Molecules are formed when atoms form covalent bonds. Covalent bonds happen when atoms are within the bonding distance (Figure 1) such that electrons can migrate between atoms. The number of electrons shared in the covalent bond is called the bond order. When electrons are shared between two bonded atoms, they change their energy level. Hence, each covalent bond has its own energy associated to it. In other words, to find the energy of the molecule, one can find the number of each type of covalent bond and multiply that by the associated energy.

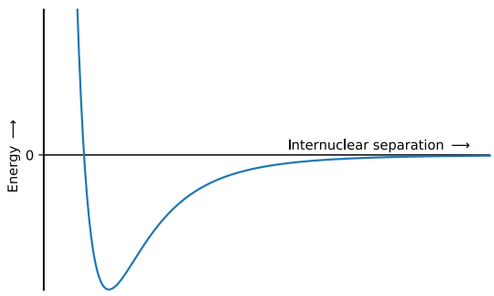

Figure 1: A curve that shows how the potential energy of a molecule changes as the internuclear separation is increased. Energy between two atoms decreases as they are brought within the bonding distance. However, the energy increases due to the Coulomb repulsive force between two positively charged nuclei. Bonding distance is a distance at which energy is not zero.

To simplify the process of figuring out bond orders, Gilbert Newton Lewis came up with the "octet rule", which states that each atom shares electrons with neighbouring atoms to achieve a total of eight valence electrons (Weller, Overton, Armstrong, & Rourke, 2018). A molecule that comes out of the octet rule is called the Lewis structure.

For each molecule, there can be multiple Lewis structures, and by choosing one, data is lost about the molecule. However, it may be restored by introducing resonance, a process in which the structure of a molecule is defined as an average of every possible Lewis structure.

For example ozone, O_3 in Figure 2. Without resonance, the bond orders would be 2 and 1 or 1 and 2. However, with resonance bond orders are 1.5 and 1.5.

Figure 2: Possible Lewis structures for ozone, O_3.

To sum up, if one can analyze the number of covalent bonds in a molecule as well as the corresponding bond orders, and one accounts for resonance, then it is possible to accurately predict the potential energy of said molecule. The neural network approach makes for a relatively computationally efficient solution compared to traditional methods of estimating the molecular energy.

In this report, we try to predict the energies of molecules, given their conformations, i.e. spatial arrangement of atoms in a molecule, using a multi-layer perceptron.

First, we will discuss the dataset used for this learning task in section 2. After this, we describe all the preprocessing and learning algorithm design steps in section 3. Then, in section 4, we will document the results that our learning algorithm yields. Lastly, in section 5 will analyze said results, reflect on them and discuss directions of potential future research.

2 Data

The dataset we are using is from a Kaggle competition "Molecular energy estimation RuCode 5.0" that had a similar goal of predicting energy values from molecule conformations. The dataset is a database with records of position matrices (conformations), atom names and energies of 138365 different molecules.

This dataset is made up of randomly sampled molecules from the MOSES dataset – a benchmarking platform by Polykovskiy et al. to support research on machine learning for drug discovery. The MOSES dataset itself is based on ZINC – a database of commercially available compounds which contains 4591276 molecules. Molecules in these datasets are encoded as strings of atoms called SMILES (Simplified Molecular Input Line Entry System). All molecules that included charged atoms or atoms besides C, N, S, O, F, Cl, Br, H or cycles longer than 8 atoms were excluded. After cleaning, there were 1936962 molecules left in MOSES.

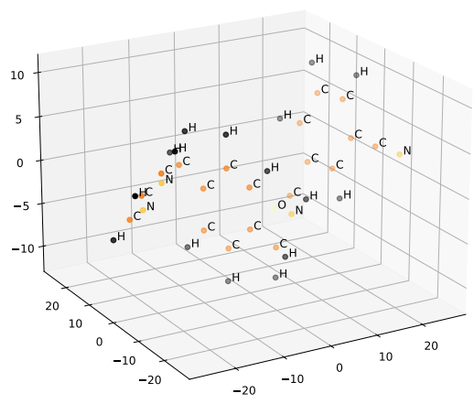

Figure 3: Example of a plotted conformation with each atom named and color coded. Axis units are not given.

SMILESs selected form MOSES were converted into a set of molecular conformations using the function `rdkit.Chem.AllChem.EmbedMultipleConfs()` from the python library RDKit by Landrum et al.. This function decodes a SMILES into a molecule and plots points in 3d space according to chemistry rules. These conformations were clustered and centered at

their centroids. Figure 3 depicts an example of a resulting conformation.

Energies of the molecules, given in kilojoules, were numerically approximated with the density function theory (DFT) function from the psi4 toolkit (Turney et al., 2012). DFT is a molecular energy approximation tool mainly used in the fields of physics and chemistry because of its phenomenal accuracy to cost ratio.

Distribution molecule sizes of the resulting dataset seems to be normal around 40 atoms (Figure 4).

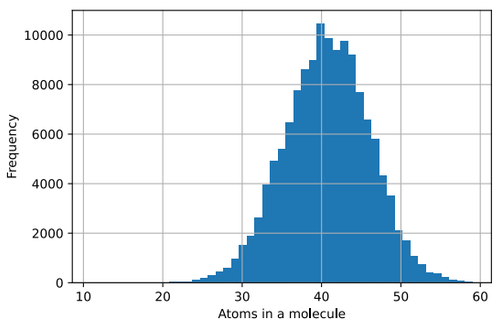

Figure 4: Distribution of data with respect to the number of atoms in a molecule.

3 Methods

3.1 Preprocessing

For our machine learning task we had to find the number of each type of covalent bond in each molecule. We preprocessed the dataset such that it satisfies this input. First, we had to calculate the distance between every pair of atoms for every molecule in the dataset. After, this we compared these distances to the maximum possible length of each bond between two atoms and got an adjacency matrix, which is later referred to as bond matrix. Then, the bond order problem was framed as a constraint satisfaction problem (CSP) and bond matrices along with the corresponding atom names were passed to a CSP solver, which returned all possible bond orders of each molecule. Finally, the number of each covalent bond was counted in each molecule.

Given: a set P of position matrices, such that $|P| = M$ and $P = \{P_i \in \mathbb{R}^n \times \mathbb{R}^3 | n \in \mathbb{N}\}$, where n is the number of atoms in each molecule and M is the number of molecules.

Also given: a set A of atom names, such that $|A| = M$ and $A = \{A_i \in \{\text{C, N, S, O, F, Cl, Br, H}\}^n | n \in \mathbb{N}\}$, where n is the number of atoms that are present in each molecule and M is the total number of molecules.

Wanted: a sample $S = (\mathbf{u}_i, \mathbf{y}_i)_{i=1,...,N}$, where N is the total number of molecules in the original dataset. The input vectors $\mathbf{u}_i \in \mathbb{N}_0^B$ are the counts of covalent bonds, where B is the number of different covalent bonds in the dataset and the output vectors $\mathbf{y}_i \in \mathbb{R}^1$ are the energies of the corresponding molecules.

3.1.1 Interatomic distances

Firstly, we found the interatomic distance matrices for each molecule. To calculate the distance between every atom in a molecule, we looked at all pairs of atoms and calculated their relative distances. This was done by iterating and calculating the euclidean distance between every two atoms in 3 dimensions:

$$d_{1,2} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}, \quad (1)$$

where $x_1, y_1, ..., z_2$ are the corresponding coordinates of atoms 1 and 2. We repeated (1) for all atoms in a molecule and arranged the values in a matrix and got the interatomic distance matrix of that molecule. This was repeated for each molecule and we got a set D of interatomic distance matrices, such that $|D| = M$ and $D = \{D_i \in \mathbb{R}^n \times \mathbb{R}^n | n \in \mathbb{N}\}$, where n is the number of atoms that are present in each molecule and M is the total number of molecules. The pseudocode for this is in the Appendix described in Algorithm 1.

3.1.2 Bond or not bond

After computing the distance matrices, we used them to find whether or not there was a bond between two atoms. For each covalent bond, there is a maximum length it can have. We compared the resulting distances to these values and made a set of bond matrices which are the adjacency matrices of each molecule. This was repeated for each molecule and we got a set B of bond matrices, such that $|B| = M$ and $B = \{B_i \in \{0,1\}^n \times \{0,1\}^n | n \in \mathbb{N}\}$, where n is the number of atoms that are present in each molecule and M is the total number of molecules. The pseudocode for this is in the Appendix described in Algorithm 2. Figure 5 is an example of the molecule structure received from the bond matrix.

Not all of the molecules had a correct bond matrix and this problem is addressed in the following subsection.

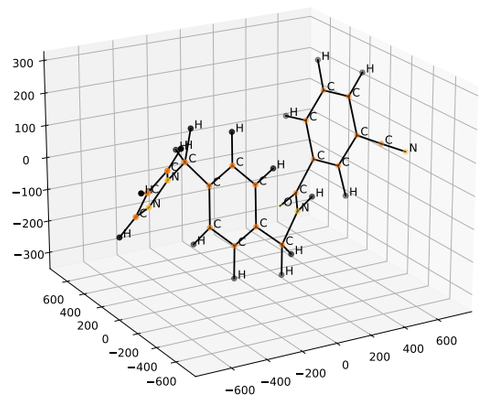

Figure 5: Example of the molecule from Figure 3 with covalent bonds. Axis units are in picometers.

3.1.3 Bond orders

To find the bond order matrix, we implemented the octet rule in code, with one exception. Most of the atoms follow the octet rule, however some atoms can have more than 8 valence electrons. In our dataset, only sulfur (S) can have a total of 8, 10, or 12 valence electrons.

In mathematical terms, the octet rule is a constraint on a graph with discrete valued edges, meaning that we can rephrase this problem as a constraint satisfaction problem (CSP), where the solution is the bond

order matrix. The pseudocode for this is in the Appendix described in Algorithm 3. To account for resonance, final solution is taken as the mean of all the solutions with equal weights. In quantum mechanics, the solutions are weighted by their energies, however we did not account for this and took with equal weights.

Unfortunately, not all the CSP's had a solution, this happened because the lengths of bonds were a little over the maximum bond length and as a result their bond matrix was wrong. The bond matrices were fixed manually and in the end all of the CSP's had a solution.

Finally, we reduced each bond order matrix by counting how many covalent bonds of each type we have. Each row in the resulting matrix is $\mathbf{u}_i$ and the energy from the original dataset is $\mathbf{y}_i$. Preprocessing resulted in a table like in Figure 6.

CC1	CC1.5	CC2	...	OS1	energy
0	9	3	...	0	-1426.3230
0	8	4	...	0	-1426.3182
0	8	4	...	0	-1426.3177
0	8	4	...	0	-1426.3147
0	2	6	...	0	-1388.4845

Figure 6: First five rows of the preprocessed dataset.

3.1.4 Dataset split

Lastly, we took 20 percent of the sample $S = (\mathbf{u}_i, \mathbf{y}_i)_{i=1,\dots,138365}$ as the test set $|S_{\text{test}}| = 27672$ to measure the performance of the model after training. Furthermore, we split the remaining sample with a 80/20 train to validation split, which gave us $|S_{\text{train}}| = 88549$ and $|S_{\text{val}}| = 22138$. We used the validation set for hyperparameter tuning.

3.2 Learning Algorithm

In order to provide an adequate description of the structure underlying the learning algorithm, it is crucial to give a formal specification of the learning task we are faced with.

Given: a training set $S_{\text{train}} = (\mathbf{u}_i, \mathbf{y}_i)_{i=1,\dots,N}$ with $N = 88549$ data points and $\mathbf{u}_i \in \mathbb{N}_0^{26}$, $\mathbf{y}_i \in \mathbb{R}^1$, a validation set $S_{\text{val}} = (\mathbf{u}'_i, \mathbf{y}'_i)_{i=1,\dots,N}$ with $N = 22138$ data points and $\mathbf{u}'_i \in \mathbb{N}_0^{26}$, $\mathbf{y}'_i \in \mathbb{R}^1$ and a test set $S_{\text{test}} = (\mathbf{u}_i^{\text{test}}, \mathbf{y}_i^{\text{test}})_{i=1,\dots,N}$ with $N = 27672$ data points and $\mathbf{u}_i^{\text{test}} \in \mathbb{N}_0^{26}$, $\mathbf{y}_i^{\text{test}} \in \mathbb{R}^1$.

Also given: a learning algorithm $\mathcal{A}$ and a loss function $\mathcal{L}_{\mathcal{A}}$ from which the empirical risk is calculated, given by:

$$R_{\mathcal{A}}^{emp}(h) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{\mathcal{A}}(h(\mathbf{u}_i), \mathbf{y}_i) \tag{2}$$

Wanted: a model $\hat{f}$ whose risk

$$R(\hat{f}) = E[L(\hat{f}(U), Y)] \tag{3}$$

is small.

This learning task is a supervised learning regression task. In our case we use a sample $S = (\mathbf{u}_i, \mathbf{y}_i)_{i=1\dots N}$ in which the input vector $\mathbf{u}_i \in \mathbb{N}_0^{26}$ represents the counts of each covalent bond type in a molecule. This vector is paired with output $\mathbf{y}_i \in \mathbb{R}^1$ which is the total potential energy in a molecule. The sample contains $N = 138365$ data points, each representing a molecule.

We assume there exists a function $f : \mathbb{N}_0^{26} \to \mathbb{R}^1$ from which $(\mathbf{u}_i, \mathbf{y}_i)$ are randomly drawn values with a noise factor v so that: $\mathbf{y}_i = f(\mathbf{u}_i) + v$. While f is unknown, we use S in order to train an estimated model $\hat{f} : \mathbb{N}_0^{26} \to \mathbb{R}^1$. We evaluate the quality of $\hat{f}$ by means of a loss function L.

This function gives us the mismatch between the model's output and desired output. We want to find $\hat{f}$ such that the the risk R of $\hat{f}$, which can be described as expected loss, is minimized. In the formula 3, risk is evaluated with respect to the true joint distributions U and Y from which our input/output samples are drawn. However, we do not know these true distributions. Instead, we minimize the loss averaged over our training sample set S_{train}.

We want to find a learning algorithm $\mathcal{A}$ that, given as input the training sample set S, searches a hypothesis space $\mathcal{H}$ for a model h for which the empirical risk is minimal:

$$\mathcal{A}(S) = \underset{h\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} L(h(\mathbf{u}_i), \mathbf{y}_i). \tag{4}$$

3.2.1 Multi-Layer Perceptron

As our architecture, we used a feed-forward multilayer perceptron (MLP) with one hidden layer. The input vector has a shape of $\mathbb{N}_0^{26}$ and the output vector has the shape $\mathbb{R}^1$, hence the input and output layers have shapes that correspond to this.

The MLP is trained to produce outputs $\mathbf{y}_i \in \mathbb{R}^1$ upon inputs $\mathbf{u}_i \in \mathbb{N}_0^{26}$ in a way that this input-output mapping is similar to the relationships $\mathbf{u}_i \to \mathbf{y}_i$ found in the training data S_{train}. Hence, it can be viewed as a function $\mathcal{N}$, such that

$$\mathcal{N}_\theta : \mathbb{N}_0^{26} \to \mathbb{R}^1, \tag{5}$$

where θ are all the trainable parameters of $\mathcal{N}$.

This function $\mathcal{N}(\mathbf{u})$ is computed as:

$$\mathbf{x}^1 = relu(\mathbf{W}^1\mathbf{u}) \tag{6}$$

$$\mathbf{y} = \mathbf{W}^2\mathbf{x}^1 = \mathcal{N}(\mathbf{u}), \tag{7}$$

where $\mathbf{x}^1$ is the activation vector of the hidden layer, $\mathbf{y}$ is the activation vector of the output layer and $\mathbf{W}^1$ and $\mathbf{W}^2$ are the connection weight matrices from the input layer to the hidden layer and from the hidden layer to the output layer respectively. The activation function $relu$ is the rectified linear unit, which is computed as

$$relu(x) = \max(0, x), \tag{8}$$

introduces non-linearity in the function.

To measure the similarity of $\mathcal{N}(\mathbf{u})$ and $\mathbf{y}$, we introduce the loss function

$$L : \mathbb{R}^1 \times \mathbb{R}^1 \to \mathbb{R}^{\geq 0}, \tag{9}$$

that is defined as a quadratic loss function

$$L(\mathcal{N}(\mathbf{u}), \mathbf{y}) = \| \mathcal{N}(\mathbf{u}) - \mathbf{y} \|^2 \tag{10}$$

because this is a regression task where we need the $\mathcal{N}(\mathbf{u})$ represent $\mathbf{y}$ as possible.

Our MLP updates its weights with a stochastic gradient descent method by Kingma and Ba that is called Adam. Adam is robust and cost efficient optimizer based on adaptive estimates of first-order and

second-order moments. It inherits its positive attributes from its previous successors: gradient descent with momentum and root mean square propagation (RMSP).

The update rule for model's parameters $\mathbf{W}$ at time step t is

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \hat{m}_t(\frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}), \tag{11}$$

where $\hat{m}_t$ and $\hat{v}_t$ are unbiased decaying average of momentum and velocity respectfully and ϵ is a positive number close to zero to avoid division by zero.

Momentum m_t and velocity v_t are computed as follows

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)(\frac{\delta L}{\delta \theta}) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\frac{\delta L}{\delta \theta})^2 \end{aligned}, \tag{12}$$

where β_1 and β_2 are the decay hyperparameters and L is the loss. In the first few iterations m_t and v_t have a bias towards zero, and to counter that the unbiased $\hat{m}_t$ and $\hat{v}_t$ values are used. They are calculated through

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{(1-\beta_1^t)} \\ \hat{v}_t &= \frac{v_t}{(1-\beta_2^t)} \end{aligned}. \tag{13}$$

By default $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and do not need much tuning.

3.2.2 Hyperparameter tuning

To find the optimal hyperparametes we are solving a task of

$$\theta_{opt} = \underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1,\dots,N} L(\mathcal{N}_\theta(\mathbf{u}_i), \mathbf{y}_i), \tag{14}$$

where Θ is a set of all possible parameters.

Of the all the tunable hyperparameters, we chose the number of neurons in the hidden layer and the learning rate. We used random search to navigate through their search spaces. For the number of neurons we took 32 random uniform points from range $[20, 1000]$ with a step 20, and 32 random uniform points from range $[10^{-3}, 10^{-1}]$ scaled by log_{10}.

We trained these 32 models on the validation set S_{val} and calculated the performance as mean absolute error (MAE) using K-fold cross validation with 10 folds. The results are in Figure 7.

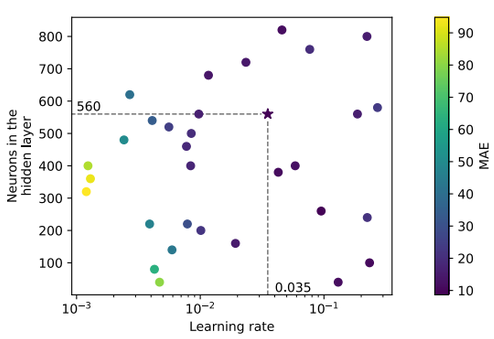

Figure 7: Results of random search for hyperparameter tuning of learning rate and the number of neurons in the hidden layer. Point with the lowest MAE in K-fold is labeled as a star. X-axis is in $\log_{10}$ scale.

From Figure 7 we see that the model with 560 neurons in the hidden layer and a learning rate λ of 0.035 has the lowest MAE. To get the model with the optimal parameters we trained the network with these hyperparameters for 200 epochs as that's the range where the network seems to converge.

4 Results

The MLP with one hidden layer with 560 trained on S_{train} with an Adam optimizer with the learning rate $\lambda = 0.035$ was tested on the S_{test} set. We ran the model on the test set and obtained the predicted energies of the molecules. As the performance metric for the learning algorithm $\mathcal{A}$ we chose mean absolute error (MAE). It is computed as follows:

$$\text{MAE} = \frac{1}{N} \sum_{i=1,\dots,N} \parallel \mathcal{N}(\mathbf{u}_i) - \mathbf{y}_i \parallel \tag{15}$$

In Figure 8 we can see that the absolute error of the model is approximately ± 1 kJ. In Figure 9) the relationship between the MAE and size of the molecule can be observed. We noted that for all molecules with more than 30 atoms, the error is approximately 1 kJ. However, molecules smaller than that can achieve errors up to 3.5 kJ.

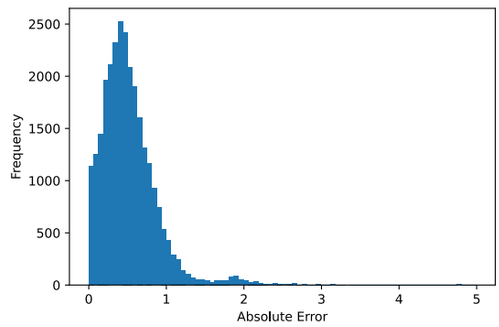

Figure 8: Histogram of the model's absolute error on the test set. We excluded 14 molecules from the plots because they had a MAE over 5 and made the plot underrepresented the data.

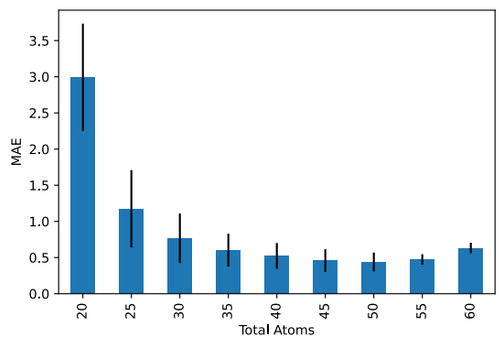

Figure 9: Bar chart of how MAE changes with the increase of amount of atoms in the molecule. Number of atoms were rounded to the nearest multiple of 5. Lines on chart indicate the standard error.

5 Discussion

In this section we will analyze our results, reflect on the development process and talk about what worked best and finally talk about some improvements that could have been made to our project.

5.1 Analysis

The aim of this project was to train a MLP to learn to predict the energy of a molecule, which was done successfully. Our model predicts energy of bigger molecules with an error of ±1 kJ and smaller ones with ±3 kJ, which is good. This difference could be the result of the dataset being unevenly distributed, favouring bigger molecules as seen in 4. However, this should not and did not effect the results a in a fundamentally important way because the final energy of the molecule is not dependent on the size but rather the energies in the bonds. It could be that the model trained on the bigger molecules attributed bigger and smaller energies to each covalent bond and while it cancels out in the bigger molecules, the difference is observable in the small ones.

The model's accuracy is overall fairly reasonable. It gives potential energy levels close to the actual values of the molecules, independent of the size of the molecule. However, these values may not be sufficiently precise for all purposes. If one is to find wider applications for the learning algorithm as described in this report, this needs to be resolved. One potential reason behind the imprecision could be because of how the bond order matrices are made, and addressed in the improvements section.

5.2 Reflection

In the process of designing an algorithm that could estimate the potential energy of a molecule, the main challenge did not necessarily lie in determining the optimal structure of the neural network itself. We settled on creating a MLP by means of predefined python libraries relatively quickly. The most challenging part of the entire process was figuring out how to do the data preprocessing to achieve the best results. More specifically, we underestimated the complexity in the spatial configuration of the atoms in the molecules and first few attempts in preproccessing the data resulted in losing the features that were important for the model. For example the realization that the bond order is a crucial determinant of the energy level in said covalent bond, did not come until relatively late in the project.

5.3 Improvements and Future Directions

To improve this project, rather than using a MLP, we propose using graph neural networks (GNN). Molecules posses a graph like structure as pointed out before. Therefore it naturally makes intuitive sense to implement the learning algorithm by means of a GNN. Furthermore, current research has discovered that GNN's show promising results for predicting molecular properties including potential energy (Schütt et al., 2017; Gasteiger, Groß, & Günnemann, 2020). Although we thought about implementing one from the beginning, it is unfortunately outside the scope of this report.

Improvements could also be made in the preprocessing pipeline. Specifically, the process of figuring out the bond orders can be optimized. As we saw in the section 3.1.3, not all the molecules had a bond matrix that resulted in a solution from the CSP, which was fixed manually by hand. However, we do not know if there are molecules that got a solution from an incorrect bond matrix. These little mistakes maybe do not effect the overall accuracy, but effect the precision of the model. Hence, we invite readers to think about how to find the bond matrix differently (more accurately) or the bond order matrix in general.

During the work on this project we found another way, besides using a CSP, to find the bond orders. We used the mean distance of each type of covalent bond. In other words, if the two atoms are within a certain distance so that they form a covalent bond of some order, then they are bonded as such. This method gave us a MAE of ±0.03 for all the molecules. However, the problem with this method is that it does not account for the octet rule or chemistry theory in any way. And we could not motivate the choices we made.

Acknowledgements

We want to say our thank you to Herbert Jaeger for lecturing the Neural Networks course and providing the materials that helped in writing of this report. We also want to thank Leo Katzenberger for explaining to us the chemistry theory behind and giving ideas on how to approach the project.

Appendix

Algorithm 1 A function to calculate the distance matrix from position matrix.

```
procedure Dist(P_1, P_2)
    d ← Pow(P_1[0] − P_2[0]), 2) + Pow(P_1[1] − P_2[1]), 2) + Pow(P_1[2] − P_2[2]), 2)
    return Sqrt(d)
end procedure

procedure PositionToDistanceMatrix(P)
    D ← []
    n ← length(P)
    for i = 0 to n do
        D[i] ← []
        for j = 0 to n do
            D[i, j] ← Dist(P[i], P[j])
        end for
    end for
    return D
end procedure
```

Algorithm 2 A function to calculate the bond matrix from distance matrix and atom names.

```
1: procedure IsBond(atom_name_1, atom_name_2, d)
2:     bond_name ← atom_name_1 + atom_name_2
3:     if d < MaximumBondLength(bond_name) then
4:         bond ← 1
5:     else
6:         bond ← 0
7:     end if
8:     return bond
9: end procedure
10:
11: procedure DistanceToBondMatrix(D, A)
12:     B ← []
13:     n ← length(D)
14:     for i = 0 to n do
15:         B[i] ← []
16:         for j = 0 to n do
17:             B[i, j] ← IsBond(A[i], A[j], D[i, j])
18:         end for
19:     end for
20:     return B
21: end procedure
```

Algorithm 3 A function to find all possible bond order matrices using CSP a solver.

```
1: procedure CSPSolveBondMatrix(B, A)
2:     V = []                                          ▷ Array of variables
3:     n ← length(A)
4:     for i = 0 to n do
5:         for j = i to n do
6:             if B[i, j] is not 0 then
7:                 V.insert((i, j))
8:             end if
9:         end for
10:    end for
11:
12:    D = {}                                          ▷ Dictionary of domains
13:    for each v in V do
14:        bond_name ← A[v[0]] + A[v[1]]
15:        D[v] ← PossibleBondOrders(bond_name)
16:    end for
17:
18:    C = []                                          ▷ Array of constraits
19:    for i = 0 to n do
20:        total_deg ← AtomTotalDegree(A[i])           ▷ Total degree this atom should have
21:        bonds ← []                                  ▷ Array of bonds from this atom
22:        for all (i, j) in V do
23:            bonds.insert((i, j))
24:        end for
25:        C.insert(AtomDegreeConstraint(total_deg, bonds))
26:    end for
27:    S ← CSPAllSolutions(V, D, C)
28:    return S
29: end procedure
```

References

Gasteiger, J., Groß, J., & Günnemann, S. (2020). *Directional message passing for molecular graphs.* arXiv. Retrieved from https://arxiv.org/abs/2003.03123 doi: 10.48550/ARXIV.2003.03123

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Landrum, G., Tosco, P., Kelley, B., Ric, sriniker, gedeck, ... DoliathGavid (2022, June). *rdkit/rdkit: 2022_03_3 (q1 2022) release.* Zenodo. Retrieved from https://doi.org/10.5281/zenodo.6605135 doi: 10.5281/zenodo.6605135

Polykovskiy, D., Zhebrak, A., Sanchez-Lengeling, B., Golovanov, S., Tatanov, O., Belyaev, S., ... Zhavoronkov, A. (2020). Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models. *Frontiers in Pharmacology*.

Schütt, K. T., Kindermans, P.-J., Sauceda, H. E., Chmiela, S., Tkatchenko, A., & Müller, K.-R. (2017). Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. Retrieved from https://arxiv.org/abs/1706.08566 doi: 10.48550/ARXIV.1706.08566

Turney, J. M., Simmonett, A. C., Parrish, R. M., Hohenstein, E. G., Evangelista, F. A., Fermann, J. T., ... Crawford, T. D. (2012). Psi4: an open-source ab initio electronic structure program. *WIREs Computational Molecular Science*, *2*(4), 556-565. Retrieved from https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wcms.93 doi: https://doi.org/10.1002/wcms.93

Weller, M., Overton, T., Armstrong, F., & Rourke, J. (2018). *Inorganic chemistry.* Oxford University Press. Retrieved from https://books.google.kz/books?id=cmZaDwAAQBAJ