OPTIMIZATION OF THE FORMULA FOR VALENCE CALCULATION USING THE CRYSTALLOGRAPHY OPEN DATABASE

Kyrylo Birkin, Oleg Malanyuk

Abstract

This report explores the effectiveness of the Bond Valence Sum (BVS) model in predicting atomic valence, using a comprehensive dataset of crystal structures. Aiming to optimize key parameters, r0 and B, our work grappled with the computational complexity of optimizing 376 variables, and sensitivity to local minima. The lack of a universal definition for 'neighbors' in a crystal structure posed another challenge. These findings illuminate limitations in the BVS model, namely its oversimplification of atomic interactions and inability to consider unique factors for different compounds. Further research is suggested, including redefining 'neighbors,' considering all atomic distances, and rounding individual terms in the sum. The need for more nuanced modeling techniques to accurately portray atomic interactions in crystalline structures is emphasized.

Literature reference

The project drew upon the article “Bond-Valence Parameters Obtained from a Systematic Analysis of the Inorganic Crystal Structure Database” by I. D. Brown and D. Altermatt. This work presented a method for calculating valences and provided the foundation for the optimization approach used in our project.

The referenced paper highlighted a program named SINDBAD, essential for determining the chemical connectivity in an inorganic crystal. The SINDBAD program's working principle relies on certain parameters for calculating bond valences, primarily expressed through two equations:

$$s = (r/r_0)^{-N}$$

and

$$s = \exp[(r_0 - r)/B]$$

where r_0, N and B are empirically determined parameters.

The relationship between the bond length (r) and the bond valence (s) is expressed by these equations. The SINDBAD program provides an ideal resource to refine these parameters by fitting the bond valence sum rule (equation 3) to the environments found around a number of cations. Reliable values can be found with 10 to 20 cation environments, enhancing the quality of the parameters.

Notably, equation (2) demonstrates an advantage over equation (1) due to the consistency of B value across different atom pairs. This feature allows B to be refined (i.e., selecting the value of B that minimizes the error value σ') and subsequently used to calculate the parameter r_0' for each cation environment. This method led to the development of the formula:

$$V_i = \sum_j s_{ij} = \sum_j \exp\left(\frac{r_0' - r_{ij}}{B}\right)$$

Introduction

Materials science, at its heart, is the study of the properties of solid matter in various forms, ranging from bulk materials to nanomaterials. Crystallography, a branch of materials science, involves the study of the arrangement of atoms in crystalline solids, providing crucial insights into their properties and potential applications. With the advent of large, open crystallographic databases, it has become possible to conduct high-throughput screening of materials, speeding up

the discovery of new materials with desirable properties. However, the sheer scale of these databases presents a challenge in extracting meaningful and useful information.

This study addresses this challenge by employing an computational approach to analyze a large crystallographic database, specifically the Crystallography Open Database (COD). This database houses an impressive collection of around 500,000 crystallographic information files (CIF), which provide valuable insights into the crystal structures of various materials.

Our focus is on applying the bond valence sum (BVS) method, a widely used empirical approach to estimate bond lengths in crystal structures. While traditional usage of the BVS method often involves manually inputting bond valence parameters, our approach is decidedly more data-driven and is intended to automate and optimize this process across a large dataset.

The first part of this study involves sifting through the vast dataset to filter out the files of interest, utilizing Python for developing a two-step filter. We zero in on files containing specific elements of interest (halogens and oxygen) and those that adhere to a specific format compatible with the pymatgen library's CIF parser.

Once the data of interest is isolated, the study then moves on to composing equations for each atom and its neighbors within the crystal structure and calculating initial values for the ensuing optimization process. This is followed by the implementation of a gradient descent algorithm, a popular optimization technique, to optimize the bond valence parameters.

By utilizing TensorFlow, a robust open-source platform for machine learning, this optimization process is carried out with high efficiency, and its implementation is geared towards minimizing the difference between calculated and actual atom valences.

This study stands at the intersection of materials science, computational crystallography, and machine learning, providing a unique perspective on the analysis of large crystallographic databases. The outcome presents an intriguing potential for streamlining the discovery process of new materials, thereby marking a significant stride in the realm of computational materials science.

Understanding the Formula

The foundation of our work lies in the comprehension of the Bond Valence Sum (BVS) formula, a mathematical representation that facilitates our understanding and prediction of atomic valence. The formula is expressed as:

$$V_i = \sum_j s_{ij} = \sum_j \exp\left(\frac{r_0' - r_{ij}}{B}\right)$$

where,

- V_i is the valence of the atom under consideration.
- s_{ij} represents the valence sum between the atom and each of its neighbors, summed over all neighbors 'j'.
- r_{ij} is the distance between the atom 'i' and its neighbor 'j'.
- r_0' is the mean distance between two specific elements when they have a single bond.
- B is a constant.

The parameter of r_0 in this formula is to aid in the convergence towards the exact valence. If r_0 is equal to r_{ij}, i.e., the mean distance is equivalent to the actual interatomic distance, the exponent yields one, signifying a single bond.

However, in the situation where r_{ij} is less than r_0, it could indicate that there's more than one bond between the two elements. In such a case, the difference within the exponent would be positive, and the formula would yield a value greater than one, resulting in a valence sum of 2, 3, and so on, depending on the number of bonds.

On the contrary, if r_{ij} exceeds r_0, the exponent returns a smaller number, indicating that the bond is weaker or less significant. Consequently, this contributes a smaller fraction to the overall valence. This variation allows the BVS formula to approximate the actual atomic valence more accurately by considering both the mean distance and the actual interatomic distance.

This understanding of the BVS formula is the backbone of our study, guiding our methodology, and informing our interpretations of the results. It's essential to remember that our project was not solely about optimizing parameters but scrutinizing the formula's robustness and exploring whether we could find parameters that would ensure its optimal functioning. The comprehension of the formula has also been instrumental in illuminating its limitations, which has been crucial for understanding the results and deriving meaningful conclusions.

Project Methodology

This project aimed to reverse engineer this formula and optimize the constants to minimize the error in calculating valence, focusing on key quantities such as valence and distance to neighboring atoms. Notably, only those atoms having Oxygen or elements of group 7 as neighbors were considered.

Methods and Implementation

This project relied heavily on computational resources to manipulate and analyze an extensive database. The Crystallography Open Database (COD), the primary data source for this project, consists of more than 500,000 crystal lattice files in the CIF format. Each file in the database contains experimentally calculated parameters for each lattice. Key values such as atomic valence and distance to neighboring atoms are included in these files. To process this extensive database, both computational power and efficient coding were required.

1. Computational Resources and Database Management

Given the massive size of the database, managing and processing the files presented a significant challenge. The database's memory footprint was approximately 87 GB. This size proved too large for processing on a personal computer, necessitating the use of the Piz Daint supercomputer. Utilizing a supercomputer provided the necessary resources to handle such a large volume of data.

2. File Filtering

Firstly, the program scans all the files in the database, checking if the file ends with ".cif". The relevant files are then filtered using a two-step validation process. The first validator checks for the presence of certain elements (Oxygen or Gallogens from Group 7). This information is stored in a 'cif' file within the markers '_loop' and tags starting with '_atom_site'. Files passing this check are copied for further processing.

```
def file_filter(validator, dst_folder, file_path):
    if file_path.endswith('.cif'):
        # print(f"Processing: {file_path}")
        if validator(file_path):
```

```
            dst_file_path = os.path.join(dst_folder, os.path.basename(file_path))
            shutil.copy(file_path, dst_file_path)
            print(f"Copied: {file_path}")
```

- First validator

```
def contains_valence_info(filename):
    with open(filename, 'r', errors='ignore') as file:
        loop_found = atom_site_found = False
        for line in file:
            if line == "loop_\n":
                loop_found = True
                atom_site_found = False
            if loop_found and line.startswith("_atom_site"):
                atom_site_found = True
            if loop_found and atom_site_found and re.match(r"^[a-zA-Z]{1,2}\d+\s[a-zA-Z]{1,2
                return True
    return False
```

After this first filtering stage, the number of files dropped from 500,000 to 5,168.

The second validation step checks the structure of the 'cif' files. While 'cif' files generally have a universal structure, in reality, files may vary as they are filled by different individuals, leading to inconsistencies. Files were therefore validated for parsability using the Pymatgen library. Files causing warnings were skipped, reducing the data pool to 2,042 files.

- Second validator

```
def parsable(filename):
    parser = CifParser(filename)

    try:
        with warnings.catch_warnings(record=True) as w:
            warnings.simplefilter("always")
            structure = parser.get_structures()[0]

            cnn = CrystalNN()

            for i, site in enumerate(structure):
                neighbors = cnn.get_nn_info(structure, i)

        if len(w) > 0:
            return False
```

```
        return True
    except:
        return False
```

3. Selection of initial values

This part of the code is dedicated to preparing the dataset for the optimization process, primarily through the composition of equations and the computation of initial values for the gradient descent.

To optimize the critical r0 value in the BVS formula, it was imperative first to establish a sensible starting position. This required a thoughtful analysis of the atomic and elemental properties detailed in the cif files. For every atom of interest and its neighboring atoms, an index was created. This index was essentially a pair (index1, index2), consisting of two numbers representing the atomic numbers of the two elements, as they appear in the Mendeleev's Periodic Table.

Next, an exhaustive search across all crystal lattices was conducted to find all identical pairs and their corresponding inter-atomic distances. These distances were then added together, and the sum was divided by the total number of these pairs, yielding an average distance from a given element to a specific neighbor. This average inter-atomic distance provided the initial value for optimizing r0. This process was carried out for all elemental pairs of interest in our study, generating a valuable dataset for the subsequent optimization stage.

3.1. Composition of Equations

The compose_equations function processes each file in the filtered CIF dataset. It uses the CrystalNN (Crystal Nearest Neighbors) method from the pymatgen library to identify neighboring atoms for each atom in the crystal structure. It then creates an equation for each atom which is not an element of interest (not a halogen or oxygen) and has not been visited before.

For every equation, the atom's valency (absolute oxidation state) and its atomic number from the Mendeleev table is saved. Also index that is a pair (index1, index2) was created. Additionally, the atomic number and distance to all neighboring atoms (which are elements of interest) are stored. This set of equations forms the basis for the optimization process.

```
def compose_equations(_, file_path):
    parser = CifParser(file_path)

    structure = parser.get_structures()[0]

    elements_of_interest = ("F", "Cl", "Br", "I", "At", "Ts", "O")

    cnn = CrystalNN()
    visited = set()
    equations = []

    for i, site in enumerate(structure):
        if str(site.species.elements[0]).startswith(elements_of_interest):
```

```
            suitable_elems = cnn.get_nn_info(structure, i) # finding the neighbors
            for elem in suitable_elems:
                index = elem['site_index']
                if not str(elem['site'].species.elements[0]).startswith(elements_of_interest) and inde
                    visited.add(index)
                    valency = abs(elem['site'].species.elements[0].oxi_state)
                    equation = [ valency, elem['site'].species.elements[0].number, [] ]
                    neighbors = cnn.get_nn_info(structure, index)
                    for neighbor in neighbors:
                        current_distance = elem['site'].distance(neighbor['site'])
                        equation[2].append([neighbor['site'].species.elements[0].number, current_dista

                    equations.append(equation)
    return equations
```

3.2. Initial Values for Gradient Descent

Next, the code calculates the initial variable values needed for the gradient descent optimization process. It does this by computing average bond lengths for each element pair across the dataset. It keeps a running total of the distances and the count of each element pair it encounters. The final average distance is then computed by dividing the total distance by the count for each element pair. This data forms the initial input for the gradient descent optimization process.

```
def calculate_average_distance(_, file_path):
    # print(file_path)
    parser = CifParser(file_path)

    structure = parser.get_structures()[0]

    cnn = CrystalNN()
    average_distance = dict()

    for i, site in enumerate(structure):
        elements_atomic_number = site.specie.number
        neighbors = cnn.get_nn_info(structure, i)
        for neighbor in neighbors:
            neighbor_atomic_number = neighbor['site'].species.elements[0].number
            current_distance = site.distance(neighbor['site'])
            key = (min(elements_atomic_number, neighbor_atomic_number), max(elements_atom

            if key in average_distance:
                dist, number = average_distance[key]
                dist += current_distance
```

```
            number += 1
            average_distance[key] = (dist, number)
        else:
            average_distance[key] = (current_distance, 1)

    return average_distance
```

4. Evaluate equations

This part of the code is responsible for implementing the optimization process.

4.1. Evaluate Equation

The evaluate_equation function takes an atom and its neighbors and evaluates the sum of all the exponential terms in the bond valence sum (BVS) equation. The function uses Tensorflow operations to perform this computation for efficient and fast calculation.

```
def evaluate_equation(main_index, terms):
    secondary_indices = np.array(terms)[:, 0]
    current_distances = np.array(terms)[:, 1]
    keys = np.stack([np.minimum(main_index, secondary_indices), np.maximum(main_index,
    masks = tf.reduce_all(tf.equal(r0_keys[:, None, :], keys), axis=-1)
    indices = tf.argmax(tf.cast(masks, tf.float32), axis=0)
    return tf.reduce_sum(tf.exp(tf.gather(r0_values, indices) - current_distances) / b)
```

4.2. Loss

The loss function evaluates the difference between the calculated BVS and the atom's valency (which we know from the CIF files). This difference is used to create a loss function, which is needed for the optimization process. The loss function is the total of all these differences for all atoms in the dataset, which the gradient descent algorithm will attempt to minimize.

This approach ensures a data-driven way to determine bond-valence parameters. It's important to note that the gradient descent process should be set up with suitable learning rates and iteration numbers to ensure that the process converges to a minimum. The exact specifics of the gradient descent implementation and the final optimized values of the bond-valence parameters will likely depend on further parts of your code, which are not included here.

The final optimized values could be tested for their performance by comparing the calculated bond valences with the known valences for a test dataset. This would provide a way to measure the effectiveness of the optimized parameters.

```
def loss():
    total_loss = 0
    for equation in equations:
```

```
        cur_diff = tf.abs(evaluate_equation(equation[1], equation[2]) - equation[0])
        total_loss += tf.exp(cur_diff) - tf.constant(1.)
    return total_loss
```

Conclusions

This project embarked on an intricate journey to scrutinize the functionality of the Bond Valence Sum (BVS) formula, a mathematical model widely used to represent atomic valence - a cornerstone concept in crystallography and materials science. With a specific focus on Oxygen and elements from Group 7, the primary task was not merely to optimize the parameters r0 and B within the BVS formula. Instead, the challenge was to discern whether it is possible to identify such parameters that would ensure the successful operation of the formula, thereby attesting to its applicability and effectiveness.

The methodology included working with more than 500,000 crystal lattice files from the Crystallography Open Database (COD). Processing such a colossal dataset demanded high computational resources, where the Piz Daint supercomputer was employed to handle the data efficiently. The first part of the project revolved around data extraction and calculation of average inter-atomic distances, providing a starting point for optimizing the r0 variable in the BVS formula.

The system with 376 variables and over 19,000 equations and , posed a formidable computational challenge. The optimization methods typically employed, such as gradient descent, were ill-equipped to handle this high level of complexity. The enormous number of variables led to a multitude of local minima within the solution space. Consequently, the optimization process was frequently caught in these local minima, preventing it from reaching an acceptable solution. Moreover, the learning rate, a crucial parameter in gradient descent algorithms, was found to be highly sensitive and required meticulous tuning.

The study also highlighted the multifaceted nature of atomic valence, which is not solely dependent on the distance between atoms. Instead, it is influenced by a range of factors unique to specific atomic combinations. Factors such as atomic size, electronegativity, orbital hybridization, and the surrounding molecular environment all influence the final valence. While the BVS formula provides a useful approximation, it does not incorporate these multiple aspects of atomic bonding, limiting its precision.

The definition of 'neighbors' in the context of crystal structures posed another challenge. The Pymatgen library determines neighbors based on a specific radius from the atom of interest. This definition, however, is too restrictive and may overlook significant interactions with atoms located just beyond this defined radius.

One of the unexpected results that our learning algorithm showed was that for certain pairs of items, the algorithm consistently produced values that could be interpreted as predicting the impossibility of a connection between these items. This not only highlights the complexity of optimizing our model and the need to define constraints more carefully, but also opens up new opportunities for further research in this area.

The project also encountered constraints related to the physical nature of the problem, such as the fact that inter-atomic distances cannot be negative or infinitesimally small. Therefore, optimizing such a system would require the introduction of a large number of constraints.

Furthermore, the optimization of the B parameter presented its own challenges. Although the original paper postulated the B parameter as 0.37, our optimization process tended to increase B to

almost 2, indicating a significant deviation. This result suggests that the optimization of B requires further research and a more nuanced approach to understand and control its behavior.

Our optimization process resulted in a minimum loss of 5061.92, which is significantly large, indicating a poor fit of the model to the data. Out of over 19,000 total equations, 3,336 were deemed incorrect upon rounding the calculated valence and comparing it with the valence sourced from the CIF files. This substantial discrepancy underscores the limitations of the BVS model in accurately predicting atomic valence. Our results suggest that major revisions or alternative approaches may be necessary to improve the accuracy of atomic valence predictions in crystal structures.

These findings illustrate the need for comprehensive and nuanced modelling techniques that can better represent complex atomic interactions in crystalline structures.

In conclusion, this project underscored the intricate nature and challenges involved in modeling and optimizing atomic properties. It highlighted the limitations of mathematical approximations, the critical role of precise parameter calibration in optimization algorithms, and the need for broader definitions of atomic neighborhoods. Future research may focus on the development of more sophisticated optimization strategies and computational techniques to better navigate the complex landscape of atomic properties. Additionally, specific attention may be warranted for the B parameter, which demonstrated a significant deviation in its optimized value from the postulated value in the literature.

Further Research Suggestions

Our investigation into the BVS formula, its applications, and its limitations has led us to several key insights. While we have pushed the boundaries of our current understanding, there remains room for further exploration. Here are two suggestions that could form the basis of future research:

1. Valence Contributions of Individual Bonds:

Throughout our study, we have calculated the total valence and rounded this value to see if the outcome aligns with the actual valence. A novel approach could involve rounding each term in the valence sum before adding them together. Since each term represents a bond with another atom, this method would provide a clear understanding of each atom's contribution to the total valence. A thorough analysis could reveal how this approach compares with our current method and if it offers any significant benefits or insights.

2. Consideration of All Elements in the Structure:

Currently, the formula takes into account only neighboring atoms when calculating valence. Our understanding of "neighbors" is basically defined by a certain radius, as implemented in the Pymatgen library. However, this definition can lead to complications, such as the exclusion of important elements or the inclusion of non-essential elements, leading to errors in the formula. A noteworthy avenue for future research might be to modify the formula to take into account all atoms in the crystal structure, not just those within a given radius. It would be interesting to see how this broader consideration affects the accuracy of the BVS formula and its ability to approximate atomic valence. Which we tried to do by conducting a small experiment, but it was unsuccessful, the loss is large, it needs more time

3. Refining the Definition of 'Neighbors':

Another potentially fruitful avenue for exploration is the concept of 'neighbors' itself. The current definition, as implemented in the Pymatgen library, is reliant on a certain radius within which an atom is considered a neighbor. However, this definition may not be entirely adequate, leading to the inclusion or exclusion of atoms that significantly impact the error in the BVS formula. Therefore, an in-depth study into more nuanced, context-specific definitions of 'neighbors' could be invaluable. This research could involve exploring various parameters beyond mere distance, to identify 'neighbor' atoms more accurately. The results of such a study could potentially improve the accuracy of the BVS formula, contributing to a better understanding of atomic valence in different materials.

This suggestions could form an essential step in the evolution of the BVS formula, allowing us to tackle the challenges we encountered in our research. Furthermore, a refined understanding of atomic 'neighbors' could have broader implications in the field of crystallography and materials science, potentially changing the way we model and predict material properties.

References:

- doi.org/10.1107/S0108768185002063
- doi/10.1021/cr900053k