Deep Neural Networks for predicting molecular properties

Raphael Peer

supervised by

Univ.-Prof. Dr. Roland Kwitt

University of Salzburg

Salzburg, Austria

2021

Abstract

This thesis investigates the application of deep neural networks to predict molecular properties from structures of small organic molecules. The raw data consists of atom coordinates in 3D space, while the target variables are either atomic interactions (Section 2.1) or properties of molecules (Section 2.2). Two machine learning competitions provided the datasets as well as the opportunity to benchmark the results against other machine learning practitioners.

While numerous approaches can be chosen to tackle this challenge, the project at hand uses the strategy of representing molecules as graphs and training deep neural networks to predict the target variables. The type of neural networks that takes graph-structured data as input is called graph neural networks, but most of the models follow an architecture described as Message Passing Neural Networks (MPNNs) [1] or Graph Convolutional Neural Networks [2]. This model class has interesting commonalities with regular convolutional networks, but also differs from them in important aspects. This thesis provides an overview of this model class and presents an example model-architecture. The investigation indicates that the representation of the 3D geometry (which is lacking in regular MPNNs) is one of the key obstacles when applying MPNNs to molecular structures. Several changes in model architecture to address this shortcoming are proposed and evaluated (Section 2.2.2 and 2.2.6). Moreover, the results provide evidence that the pure deep learning approach of using raw data only as model input (which is common practice in computer vision and natural language processing) is superior to manual feature engineering in chemical applications as well (Section 2.2.5). Finally, the experiments also show that the choice of input data representation can be more important than the choice of model architecture (Section 2.2.1).

In summary, this work shows some interesting challenges when using MPNNs in chemical applications and presents results that point towards potential improvements.

Contents

CHAPTER 1

Introduction

Motivation

In the past decade, deep neural networks have been responsible for several breakthroughs in different areas, ranging from computer vision to natural language processing and medical image analysis [3]. These astounding achievements suggest that deep neural networks have the potential to lead to many more groundbreaking applications of machine learning. However, a closer look at these breakthroughs reveals that most of the applications focus on one of two kinds of data: images and natural language. An image (in the most general sense) can be described as data-points (pixels) on a two-dimensional grid. Natural language on the other hand has a purely one-dimensional structure. In text-form, it can be described as a sequence of words, syllables or characters. In the form of speech, it is a sequence of sounds. In either case, the structure of the data is purely sequential, meaning that each data point (e.g., a word or a character) is fully described by its position in the 1D sequence.

Two classes of deep neural architectures were responsible for the breakthroughs: convolutional neural networks (CNNs) for computer vision and recurrent neural networks (RNNs) for natural language [3]. Both model classes have seen tremendous improvements over these last years, but all of them are built on the same basic architectures [1]. For this reason, the basic strategy in applied deep learning can be over-simplified as: Use the latest convolutional architecture that suits the task at hand in computer vision and use the latest RNN-based (e.g., LSTM) or Transformer architecture in natural language

[1]Recently, *Transformer*-architectures [4] have largely replaced RNN-based architectures in natural language processing. Up until this point, advanced RNN-architectures, such as *Long Short Term Memory networks* (LSTMs) [5] have been responsible for the advances in the field.

processing.

However, if the data for the use-case at hand does not fit into either of those two categories, the situation is much more complicated. In other areas, deep neural networks have not outperformed traditional methods by as large a margin and the choice of approach is not as obvious. Examples include documents that comprise both, text and visual elements, such as invoices, legal documents, etc. These documents cannot be fully described as a text-sequence because the position of words, amounts or dates on the page strongly affects their interpretation. (In contrast to pure text, such as a novel, where a word's meaning obviously depends only on the words before and after in the 1D sequence of words and not on its position on the page.) However, they also cannot be treated merely as images as the textual information is vital. Thus, they represent textual information on a 2D grid for which there is no readily available deep learning architecture yet [2]. Another example is 3D data, such as point-clouds [7] or molecular structures. In theory, CNNs can be extended to three or higher dimensions, but in practice this approach is computationally not feasible due to the cubic number of data points in three dimensions.

From these considerations, it becomes clear that it is not trivial to extend the success of DNNs from images and language to other kinds of data. Nevertheless, the potential for progress is tremendous and warrants research into neural networks for other kinds of data.

One of the most-versatile data structures is a graph. A graph contains a set of data points called vertices or nodes and a set of edges that define relations between the nodes. As such, a graph is inherently unordered and thus suited for modeling data without predefined order. The only kind of structure that has to be imposed is the definition of the edges. Thus, the only kind of inductive bias of a graph are the relations between nodes [8]. This relational inductive bias is a rather weak bias and allows many kinds of data to be modeled as a graph without imposing some kind of structure onto the data

[2]An interesting approach to combine convolutional networks with textual information is the so-called chargrid [6]. In short, each pixel on the image is mapped to character (including a 'null-character'). These characters are then represented as channels in the input to the convolutional network. Just like a regular image has three color channels, a chargrid with n characters has n-input channels. The same concept can be expanded to words or syllables as well. While the results are promising, this model-class does not outperform more traditional machine learning models with manual feature engineering (e.g., decision tree-based models) by a large margin. A similar breakthrough as in computer vision where all previous techniques have been made obsolete by the invention of CNNs has therefore not yet been achieved for 2D documents.

that is not there.

The class of neural networks for graph structured data are called Graph Convolutional Networks (GCNs) [9] [2] or Message Passing Neural Networks (MPNNs) [1]. The terminology is different between GCNs and MPNNs, but the underlying principles are the same and most neural network architectures for graphs can be described in either framework. While the MPNN terminology is somewhat more intuitive to understand, the graph convolution terminology has the advantage of highlighting the parallels to regular CNNs. In this thesis, we will mostly adhere to the more common message passing terminology, but also borrow terms from the graph convolution terminology to highlight the parallels with and differences from regular CNNs.

Obvious applications of MPNNs are molecular structures (the topic of this thesis) and social media networks (or any system where users / people interact with each other) [10]. Moreover, many kinds of data that do not seem to have an obvious graph structure at first sight can be modeled as graphs and thus analyzed with MPNNs as well. Examples include 3D point clouds [7] or invoices (with words as nodes and the spacial relationship between words as edges) [11].

In the remainder of this section, we will mostly refer to molecular structures as examples of graph structured data, but keep in mind that most principles apply to other kinds of graphs as well. In particular, the most important challenge when modeling molecular structures as graphs is the loss of some of the 3D structural information, as discussed in Section 1.3.3.1. This challenge is exactly the same for any other 3D data represented as a graph.

Message Passing Neural Networks (MPNNs)

In this section we will review Message Passing Neural Networks (MPNNs) in general. While all those considerations also apply when the input graphs represent molecules, Section 1.3 will show the specific challenges for this use case.

Challenges of graph structured data for deep neural networks

There are several challenges when designing neural networks for graphs.

No fix-sized dimension (globally). Each graph can have a different dimension in terms of number of nodes and edges. This, per se, is nothing graph-specific. After all, images also come in different sizes. However, while images can be cropped and scaled to have the same dimension if needed, there is no easy way to do something equivalent with graphs. For instance, in a dataset of 3D molecular structures, the molecules typically have different numbers of atoms (nodes) and there is no reasonable way to change this.

No fix-sized dimension (locally). In a regular graph (without isolated nodes), a node can have 1 to n neighbor-nodes. This has important implications for constructing a graph convolution operation because it requires a function that is invariant to the number of nodes in the neighborhood. We cannot just define a fixed-sized kernel function and apply it to parts of the graph analogous to regular CNNs.

No inherent order of nodes. In a graph, there is no inherent order of nodes. We cannot define a first node, second node, etc. and compare those across graphs (for comparison, note that this is possible in images as every image has an upper left corner, a center, etc.). The same reasoning applies to the neighborhood of any one node: there is no unambiguous way of ordering the neighbor-nodes that allows them to be compared to the neighborhood of another node. Again, this has important implications for graph convolution because it requires a function that is invariant to the order of the nodes.

From the last two paragraphs, one can already suspect that any neural network for graph structured data will require some sort of aggregation function, such as *mean*, *sum* or *max*. These functions work with a varying number of input nodes and are invariant to their order (permutation invariant).

Now that we briefly described the properties of the input data, we will examine the possible output formats of a MPNN in the next section. With the input data and the output defined, we will then focus on the transformations that map the input to the output in MPNNs.

Output format of MPNNs

MPNNs can be used to predict three kinds of target variables.

Graph-properties. The target variable is a scalar or vector per graph. For instance, this could be properties of a molecule to be predicted from its 3D structure. This is the only type of output format where the output-dimension is constant across graphs, even though the graphs themselves have different dimensions.

Node-properties. The target variable is some property of a single node. Thus, for each node in the graph, a scalar or vector has to be predicted. In the example of molecular structures, this would be a property of individual atoms.

Edge-properties. In this case, the target variable is a scalar or vector-valued edge-property, i.e., a property of the interaction between two nodes. In molecules, an edge-property could for instance be the interaction energy between two atoms.

Note that for the second two types of target variables, the output dimension depends on the dimension of the input graph, while only the graph-properties as target variables produce output with a constant dimension akin to a regular CNN.

Components of MPNNs

The basic building block of a MPNN is a so called message passing step that maps a given node-representation and the neighborhood of the node to an updated node representation. This step can also be called a graph convolution layer, as it shares some commonalities with regular convolution layers. Here, we will use the message passing terminology suggested by Gilmer et al. [1] to describe this step, but refer to it as graph convolution occasionally to highlight the analogy to regular CNNs.

The input data can be described as a graph G with a set of node features and (optionally) a set of edge features. The features of a given node v are denoted as x_v, while e_{vw} describes the features of the edge between node v and node w. For simplicity, we will assume undirected graphs where e_{vw} is equal to e_{wv}, which is reasonable for modeling 3D structural data. However, directed graphs can be described in the same framework if the application requires directed edges. Finally, let T be the number of graph convolutional layers or the number of message passing steps and denote a specific step as $t \in \{1, 2, ..., T\}$. Furthermore, the node representation at any hidden layer t shall be denoted as h_v^t. Note that compared to an image in regular CNNs, the node features x are equivalent to a pixel in the image, while the hidden node representations (also called hidden states) h are equivalent to the feature maps after the first convolutional layer. Finally we define the neighborhood of a node v as $N(v)$ which denotes the set of nodes which share an edge with node v (which means that, for every node w, $w \in N(v)$ if and only if there is an edge between w and v).

During each message passing step, the hidden representation of each node h_v^t is updated to h_v^{t+1} using the following procedure:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(\ h_v^t,\ h_w^t,\ e_{vw}\ ) \tag{1.1}$$

$$h_v^{t+1} = U_t(\ h_v^t,\ m_v^{t+1}\ ) \tag{1.2}$$

The above two steps are comprised of three functions:

1) The message function M computes the 'message' from node w to node v using the hidden representations of both nodes as well as the features of the connecting edge. A specific architecture may not use all the arguments of the message function in the general framework. For instance, the message function may not use the receiving node hidden state h_v^t (which is used in the next step anyway) or the edge features. In the simplest case, the message function could just be a concatenation of the arguments, such as $m_v^{t+1} = (h_w, e_{vw})$ [12]. A more complex message function could be a multilayer perceptron (MLP), which takes the concatenation (h_v, h_w, e_{vw}) as input [13].
2) An aggregation function aggregates the messages from all the neighboring nodes, $N(v)$, of node v. Any function that is permutation invariant and works for any number of nodes, such as *sum*, *mean*, *min* or *max*, can be used at this step. However, in practice, the reasonable options are mostly limited to summation and the arithmetic mean. In other papers, aggregation is described as a separate step and the aggregation function denoted with a specific symbol. Here, as well as in Gilmer et al. [1], we simply use the summation (Equation 1.1) and note that it could be replaced by the arithmetic mean if appropriate. The aggregation is not a function with learned parameters and has to be chosen by the architect of the MPNN. Moreover, it can lead to considerable loss of information. However, this step is inevitable because the neighborhood of a given node can consist of any number of nodes and they do not have an inherent order, as discussed in Section 1.2.1.
3) The update function U (Equation 1.2) updates the hidden state h^t based on m_v^{t+1}, the aggregated messages from its neighboring nodes. Examples of update functions include for instance $h_v^{t+1} = \sigma(H_t^{deg(v)} m_v^{t+1})$, where $H_t^{deg(v)}$ defines a matrix for step t and the degree of node v, $deg(v)$. This essentially corresponds to passing the

message for node v through a 'one-layer neural network' while having a separate network for each message passing step t and each node degree $deg(v)$ [12] [3]. Another example of an update function is an MLP, which takes the concatenation (h_v, x_v, m_v) as input [13].

Apart from these essential components of the message passing framework, an edge-update function can be added as an optional step. This function updates the edge-representation at each message passing step analogous to the node update function defined in Equation 1.2. The edge update function can be defined as $e_{vw}^{t+1} = E(h_v,\ h_w,\ e_{vw}^{t})$, where E can be defined in many different ways just like the node update function [14] [15]. However, edge-updates are not frequently used in the MPNNs.

Finally, if the target variable is a property of the whole graph, a readout function is required to map the node representations to the target.

$$\hat{y} = R\ (\ \{\ h_v^T\ \mid v \in G\ \}\ ) \tag{1.3}$$

Because the readout function is a function of all nodes in the graph, it has to handle varying numbers of nodes and be permutation invariant. Thus, just like the message function defined in Equation 1.1, it has to include some sort of aggregation, such as *sum* or *mean*. This aggregation could occur as the very last step, such as in Schütt et al., where all hidden node representations are passed separately through an MLP and then summed up [2]. In contrast, in Chen et al., the final step of the readout function is an MLP, which takes the already aggregated node representations as input [16].

In most architectures, the weights of both functions are shared, such that M_t and U_t are the same for all message passing steps t. However, this is not a necessity and one can also build a MPNN with different weights for each message passing step. Section 2.2.4 presents an experiment comparing the two approaches.

In the first message passing step, only first degree neighbor nodes have an impact on the updated node hidden states. In the second step, second degree neighbors have an indirect impact, as they already influenced the hidden state of the first degree neighbors in the previous step. Thus, at each step t, h_v^t is influenced by nodes with a path length

[3]Of course, this definition only makes sense in graphs where the maximum node degree is a reasonably low number and varies from node to node. In most experiments in this thesis (Section 2.2), complete graphs are used where each node has the same degree (the number of nodes in the graph), such that $deg(v) = |G|\ \ \forall\ v \in G$. For such a setting, this definition of the update function would of course not be applicable.

of at most t edges apart from node v. If the number of steps T is sufficiently large, eventually, the whole graph might have an indirect influence on the hidden state h_v^T. However, the first degree neighbors will always have the most direct influence, followed by the second degree neighbors, etc.

With a block of T message passing steps, one has successfully constructed a mapping to create higher-level features akin to the convolutional layers of a regular CNN. Now, we can examine how these higher-level features can be mapped to the desired output. While the message passing part of the MPNN can be the same for each of the three types of output format discussed in Section 1.2.2, the head of the MPNN needs to be tailored depending on the desired output format.

1) **Graph properties**. To predict graph properties, the higher-level node features computed by the convolutional layers have to be mapped to the target variable using a readout function R defined in Equation 1.3. Again, due to the variable number of nodes and the lack of any inherent order, this requires an aggregation function, which is permutation invariant and works for different numbers of nodes. Similarly to aggregation during message passing, summation and the arithmetic mean are typically the only viable options (The extensive review from Gilmer et. al. [1] contains no example of another aggregation function and the author's own literature search did not find one either). Just as during message passing, the aggregation step in the readout function potentially looses valuable information. However, due to the inherent challenges of graph structured data (Section 1.2.1), it is difficult to find a better solution. As the nodes can be regarded as an unordered set, *Set2Set* [17] and similar methods can be used in the readout function.
2) **Node properties**. In this simple case, the readout requires only a mapping from the final node hidden state h_T to the output variable, which can again be achieved with a shallow MLP.
3) **Edge properties**. To predict edge properties, a readout function needs to take the two connected node's hidden states (and optionally also the edge features) as input and map it to the desired output. [4]

In summary, a MPNN is composed of two parts: message passing (graph convolution)

[4]Some architectures use edge-updates analogous to update of node representation in regular MPNNs [15]. In this case, a mapping from the final hidden state to the desired output format can suffice as a readout function - equivalent to how node properties are predicted.

and readout function. The framework introduced here allows to describe highly different architectures using the same terminology. In Section 3.2.2, the basic architecture of a MPNN used in this thesis is described in detail.

Predicting molecular properties from 3D structure with MPNNs

Deep neural networks in chemical applications

3D structures of small chemical compounds as well as large biomolecules are readily available on a large scale [18] [16] [19]. However, just knowing the 3D structure of a molecule does not go a long way towards understanding its biological function (in the case of biomolecules) or its molecular properties and potential applications (in the case of of small organic molecules). For instance, in drug discovery, a huge amount of effort goes into laborious and costly experiments on a large scale to find the right small organic molecule for the given therapeutic target [20]. Speeding up and improving this process is highly desirable for obvious reasons. One way to address this issue is to create purely computational experiments. In a life science context, these experiments are often referred to as in-silico, analogous to in-vitro (in a test-tube) and in in-vivo (in a living organism).

Representing molecules as graphs

Figure 1.1 shows an illustration of a 3D molecule structure. The raw data contains only coordinates and atom types. The presence or absence of bonds between atoms can be derived from the types of atoms and the distance between them. Thus, the raw data of a molecule is only an unordered set of tuples of the form $(a, \vec{c})$ with $\vec{c} \in \mathbb{R}^3$ being the coordinate vector and $a \in \{1, 2, ...N_a\}$ representing the atom type with N_a being the number of atom-types in the dataset - which is seven in the dataset at hand [5]. The cardinality of the set is the number of atoms in the molecule and varies from molecule to molecule.

[5]Hydrogen (H), carbon (C), nitrogen (N), oxygen (O), fluoride (F), sulfur (S) and chlorine (Cl)

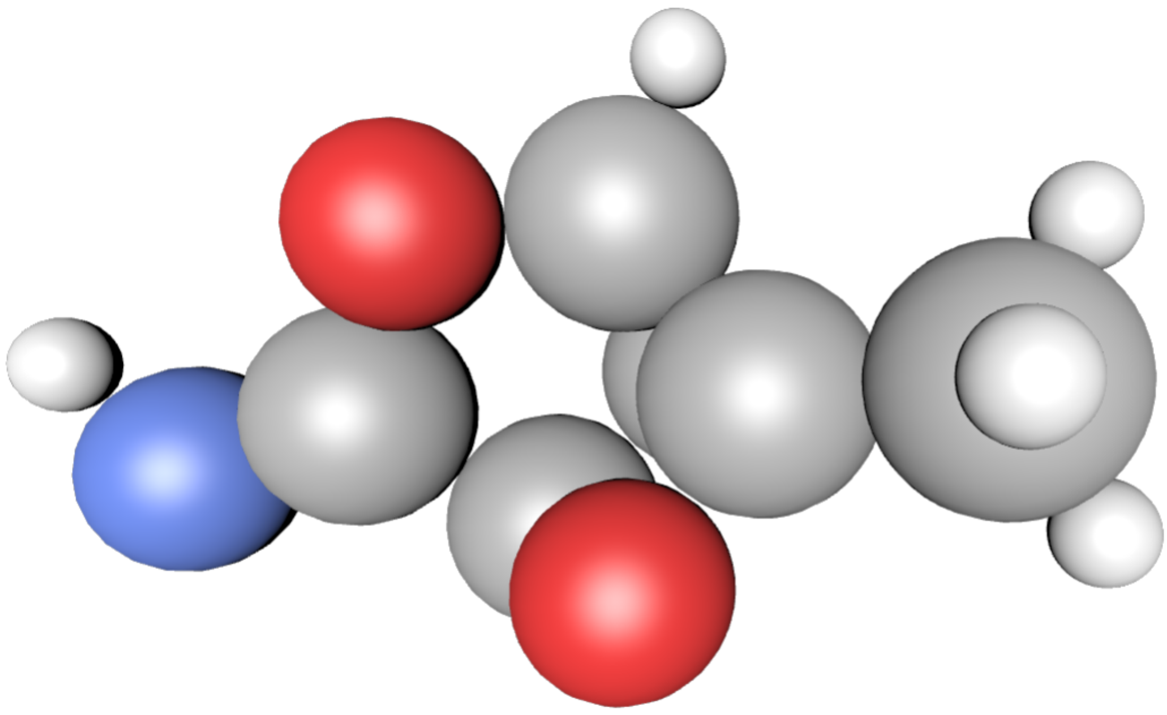


Figure 1.1: A 3D structure of a small organic molecule. Carbon and hydrogen atoms are shown in gray and white, respectively. The red spheres show oxygen atoms and the blue one shows a nitrogen atom, according to the common coloring convention. The size of the spheres is chosen such that covalently bonded atoms have no or almost no space between them.

When it comes to 3D structures, there are really only two basic ways of representing them for neural networks. The first is to describe them as 3D grids and to use 3D convolutional networks simply extending the same architectures used in computer vision by an additional dimension [21]. In this approach, you would literally use the 3D color image shown in Figure 1.1 as model input. The problem with this approach is two-fold: First of all, the cubic number of data-points makes this method computationally very expensive. The second problem is that the information encoded in all these data-points is highly redundant. In molecular structures, coordinates and types of atoms are fully sufficient to describe the molecule. All other information - such as atomic radius, etc. - are functions of the types of atoms (hydrogen, carbon, etc.). Thus, when, for instance, a ten-atom molecule is fully described by ten coordinate vectors and ten atom types (40 numbers), it is excessive and inefficient to model it as a 3D image with tens or hundreds of thousands of pixels. The advantage of the 3D convolutional approach is that 3D structure information is provided to the model, which is a challenge for MPNNs, as explained in the next section.

The second approach of representing molecules for neural networks is to use graphs. The idea of using graphs to model molecules comes quite naturally when looking at the most common way of representing small molecules in text books: so called molecule graphs, such as the one shown in Figure 1.2. A molecule is usually defined as a number of atoms connected by covalent bonds. These kind of bonds involve the sharing of one or more electron-pairs between the bonded atoms and are chemically very stable - it takes considerable energy or enzymes to break them up [22]. In laymen's terms, covalent bonds can be regarded as 'stable under normal conditions'. In the commonly used molecule graph representation, atoms are shown as letters denoting their type and the covalent bonds are drawn as edges between them, as shown in Figure 1.2. Thus, defining a graph where atoms are the nodes and covalent bonds are the edges seems like the most natural way of modeling molecules. However, upon closer examination, the situation is somewhat more complicated.

Atoms interact not only if they are covalently bonded. There is a wide array of non-covalent interactions that have a large influence on molecular properties, such as Van der Waals forces (a simple example being positive and negative partial charges) [22]. The strength of non-covalent interactions is a function of atom types, distance and the local environment in the 3D structure. Thus, only including covalent bonds leads to considerable loss of information. For instance, two atoms can be close in space and interact non-covalently, even though the path length in the covalent graph may be quite large.

With these considerations, another way of representing a molecule as graph is to simply define an edge between every pair of atoms if the distance between them is below a certain threshold. Yet another way of constructing the graph is to draw an edge between any given atom and its k nearest neighbors.

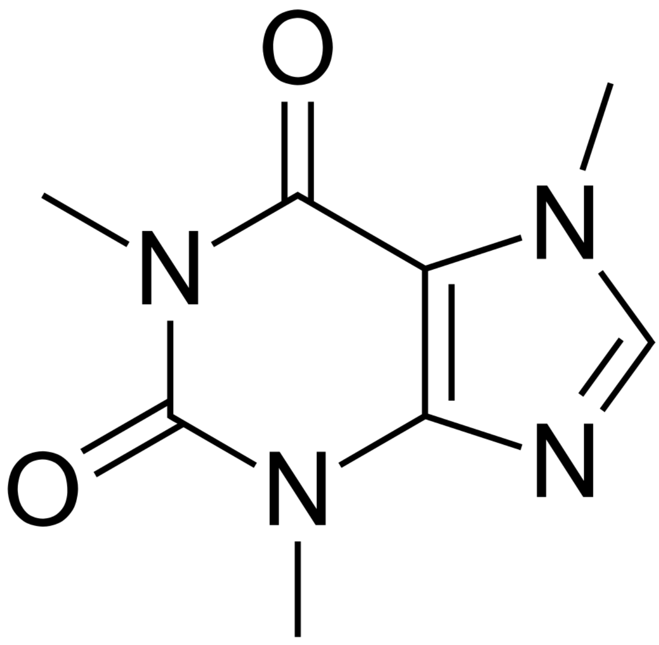


Figure 1.2: The standard molecule graph representation of caffeine. The lines represent edges in the graph. Line intersections or end-points show atoms, which represent nodes in the graph. Most of these atoms are carbon atoms as in (almost) all organic molecules, which are not indicated by a letter in the molecule graph - instead every line intersection or endpoint without a letter represents a carbon atom. Other heavy atoms are denoted by their single letter code, such as Oxygen and Nitrogen shown as O and N, respectively. By convention, hydrogen atoms are not drawn at all because their presence can be deduced from the information shown and is thus redundant. (For instance the carbon atom represented by the bottom-most line endpoint has three hydrogen atoms attached to it. That can be deduced form the fact that each carbon atom engages in four bonds with other atoms. As only one such bond is shown as the edge between the bottom-most carbon and the bottom-most nitrogen atom, three bonds are left. If they are not drawn explicitly, they are assumed to be occupied by hydrogen atoms by convention. These considerations become relevant when trying to omit hydrogen atoms from the input data, as described in Section 2.2.3.) Furthermore, double- and triple-bonds are shown in such graphs as two or three lines between two atoms. These bonds are stronger and the inter-atomic distance is shorter than in the case of single bonds [22]. In the graph representation, these types of bonds are represented as edge-attributes because there is no such thing as several edges between any two nodes in a graph. Instead, every edge has the categorical feature ”bond-type” which can be either single, double or triple.

All approaches that define the presence or absence of edges by certain rules suffer from inductive bias: we decide in advance which relations between nodes in the graph (atoms) are relevant and which are not. Hence, the model cannot learn this information from the raw data, but is restrained by predefined rules. Ideally, we would prefer to allow the model to learn everything from the raw data, which is the reason behind the success of deep neural networks in other fields. One way to get rid of this inductive bias is to simply add an edge between any two atoms in the molecule - thus creating a complete graph - and to add the distance between the atoms as an edge feature. This way, the model can 'decide' whether a relation between two atoms is relevant based on the distance. The disadvantage of this approach is the high computational cost, which prevents it from being applied to large molecules. This trade-off between inductive bias and computational feasibility lies at the core of one of the key experiments in the thesis described in Section 2.2.1.

These considerations highlight an important difference between the theory of MPNNs and their application. In theory, the starting point is graph structured data and all the effort goes into the construction of the best possible model to extract information from this data. For practical application, the data has to be modeled as graphs in a way that captures the available information in the best possible manner. Arguably, the choices made in this process may have more influence on the quality of the predictions than the choice of MPNN architecture. After all, if incomplete information is fed into the model, even the best model will yield disappointing results. In the next section, we will examine some challenges that arise when modeling molecules as graphs for MPNNs.

Limitations of MPNNs for molecular structures

Lack of 3D structure representation

In principle, 3D coordinates could simply be added to the node features to obtain a 3D graph. However, this is not meaningful for representing 3D molecular structures. First of all, there is no logical reference point for the origin of the coordinate system. The molecule can be translated arbitrarily along one, two or all three spacial axes without changing anything of relevance. The second problem is that molecules have no natural orientation - i.e., they can be arbitrarily rotated where every rotation is just as valid as any other one (in contrast to everyday objects, such as cupboards or cars, where certain rotations are unlikely to be encountered in everyday life). Figure 1.3 shows two randomly chosen rotations of a small organic molecule. While the two images represent the exact same information, the actual atom coordinates are completely different.

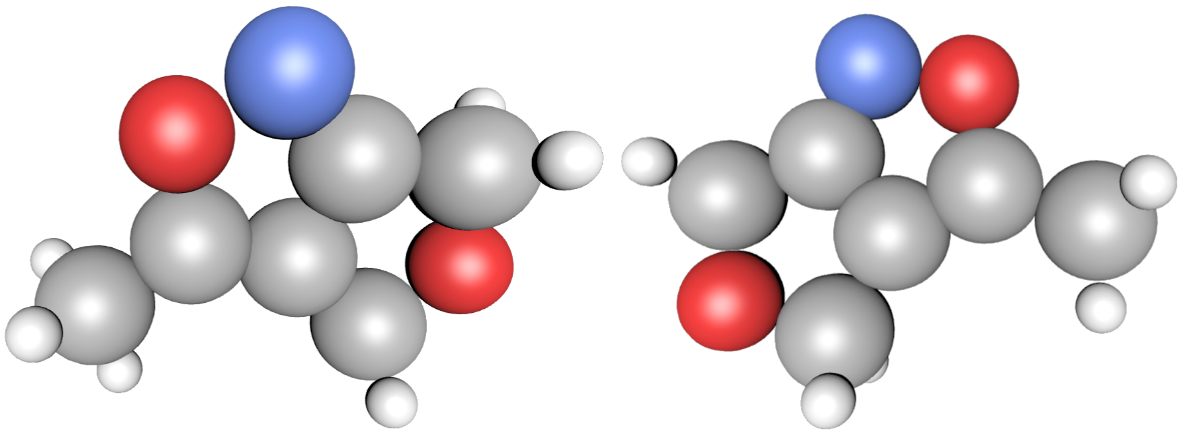


Figure 1.3: Two randomly chosen rotations of the same molecule. The values of the atom coordinates are completely different between the two rotations, even though the underlying information is the same.

For these reasons, the concrete values of the 3D coordinates, which depend on this arbitrary positioning and rotation of the coordinate system, do not posses relevant information as such. The translation issue could be addressed by instead using directional vectors from one atom to another as edge-features. These vectors are invariant to the arbitrary translation of the molecule in the coordinate system. However, they depend on the arbitrary rotation of the molecule, rendering them also not useful to encode the 3D structure, as shown in the experiment of Section 2.2.6.

In most MPNN architectures, the Euclidean distance is provided as an edge feature. This encodes part of the 3D structure in the graph, but still looses vital information. Consider, for instance, the case of the neighborhood of the blue nitrogen atom in Figure 1.3. The set of distances to the other atoms is unambiguously derived from the 3D structure (irrespective of the rotation). However, only by knowing the distance to each other atom in the molecule, the structure cannot be derived. There could be a completely different structure giving rise to the same set of distances. For instance, the fact that the blue nitrogen atom is at the margin of the molecule and not in the center is not contained in the information provided by the distances to all other atoms.

It does not take much chemical expertise to conjecture that not providing the full 3D structural information to the model limits its capability to make accurate predictions. However, there is no easy way to avoid this dilemma and, to my knowledge, no satisfying solution has been found so far.

No down-sampling during feature extraction

In computer vision problems with regular CNNs, feature maps are typically down-sampled to a lower dimension (while expanding the number of channels) every few layers. It makes perfect sense that higher-level features correspond to larger parts of the original image, while at the same time, requiring more channels to represent the information. Intuitively, one can think of a data-point on a feature map close to the end of the convolutional part as containing features that summarize a larger region of the original image. The connection between higher-level features and lower dimension feature maps is so central to regular CNNs that it is hard to imagine it any other way. However, no analogous process exists in MPNNs.

In MPNNs, the higher-level features (i.e., the node hidden states) are always associated to their respective nodes throughout all message passing steps. While each of the node hidden states contains information from the node's environment, there is no representation that summarizes a whole group of nodes. However, after the graph convolutional layers, all the node representations are immediately aggregated to give a graph representation.

In the case of molecules, such down-sampled representations could represent groups of atoms instead of just individual atoms. In organic chemistry, there are is a limited number of chemical groups, called *functional groups*, that appear frequently in organic molecules and strongly determine their properties and function. Many of those functional groups are composed of only two to four atoms and appear very frequently in organic molecules (Carbonyl, Hydroxyl, Amide, Amines, etc.) [22]. A model with the ability to down-sample would likely learn representations of these important functional groups. Another benefit of such a mechanism would be that it would facilitate the incorporation of angle-information into the model. Note that angles are neither node- nor edge-features. Instead, an angle is a property of a group of three nodes (alternatively, it can also be seen as a property of two edges). Having a representation for a group of - for example three - nodes would allow to add the angle as a feature thus providing more structural information than only distances. For these reasons it is my strong believe that future successful MPNN architectures will likely have some sort of down-sampling mechanism that allows learning representations of groups of nodes.

While no MPNN architecture that reduces the dimensionality of the graph (i.e., groups nodes together) has been found in the literature, some similar approaches have been developed. One of the winning teams of the *Alchemy*-competition [6] found a way to add angles to the message passing framework [23]. The winning solution in the Kaggle-competition 'Predicting Molecule Properties' [7] used atom-triplets and even atom-quadruplets alongside regular atom-nodes in a sort of meta-graph. The result is a very interesting model - however, it proved to be far too large to be trained with the computational resources available for this thesis. Future MPNNs for molecular structures will likely have similar elements to represent features of groups of atoms.

[6] `https://alchemy.tencent.com/`

[7] `https://www.kaggle.com/c/champs-scalar-coupling/discussion/106575`

CHAPTER 2

Results and Discussion

Predicting atomic interactions from 3D structure

This section describes my experiments for the Kaggle competition 'Predicting Molecular Properties' hosted by CHAMPS (CHemistry And Mathematics in Phase Space) [1]. Dataset and evaluation are described in Section 3.1.1. Here, it suffices to say that the raw data are 3D structures very much like the ones in the *Alchemy*-dataset (Section 3.1.2), but the target variable is an atomic interaction - i.e., an edge property in graph terminology.

Experiment	**Section**	**score***
LightGBM [24] with distance and atom type	2.1.1.1	-1.19
LightGBM with distance, angle and atom type	2.1.1.2	-1.38
SchNet DGL implementation	2.1.2	0.52
SchNet Chainer-implementation (fine-tuned)	2.1.2	-1.95

Table 2.1: Summary of the results of different experiments from the Kaggle competition 'Predicting Molecular Properties'. *The score is based on the MAE and defined in Equation 3.1 (in short, a lower score is better).

Boosted tree models with manually engineered features

Machine learning models based on manually engineered features, such as various boosted tree-models, can give surprisingly good results with short training time. They do not reach the performance of well tuned MPNNs, but easily surpass not yet fully

[1]`https://www.kaggle.com/c/champs-scalar-coupling`

optimized MPNNs. The next two subsections will briefly outline two experiments with manually engineered features used for a Light Gradient Boosted Model (*LightGBM*) [24].

Distance and atom-type features

The first type of engineered features were calculated in the following way: The center point between two atoms whose J-coupling constant has to be predicted was calculated (on a straight line). Then, the distances to the ten closest other atoms were calculated and added as features along with the types of those nearest atoms as categorical variables. Finally, the distance between two J-coupled (interacting) atoms was added to the features as well, yielding a total of 21 features. These features were fed into eight different LightGBMs for the eight different J-coupling types to predict the J-coupling constant (see Section 3.1.1 for more information about the data and the target variable). The models are fitted quickly and give reasonable results without hyper-parameter tuning. The experiment can be reproduced by running the publicly available Kaggle-notebook `https://www.kaggle.com/rpeer333/super-simple-10-nearest-atoms-features`. The result was decent given the simplicity of the approach, but markedly below a well tuned MPNN (see Table 2.1).

Several factors limit the effectiveness of this simple approach.

Lack of geometric information. The features only specify an atoms distance from the center point and not its position relative to the J-coupled atoms.

Features are not permutation invariant. As atoms in a molecule have no objective order that makes them comparable to other atoms in other molecules, the feature order is somewhat arbitrary limiting the model's ability to generalize [2]. In the general case, the nth closest atom in an arbitrary interaction is not equivalent to the nth closest atom in another interaction.

Relations between the features are not represented. Obviously, the features are not just 21 independent scalars. Instead, the closest atom's distance to the center point is related to this atom's type, etc. However, this relation cannot be represented in the input data. Instead, the features have to be input into the model as 21 independent variables, which limits their usefulness dramatically.

[2]Granted, some patterns are shared between interactions. E.g., in the case of a J2XX interaction where the J-coupled molecules are two bonds apart, the atom closest to the center point is most likely the atom to which both J-coupled atoms are bound, making the first distance and the first atom-type feature comparable across interactions.

Distance, atom-type and angle features

Regarding the three limitations in the previous section, this experiment attempted to address the first one (lack of geometric information). Recall that the distance was calculated from the center point of the straight line between the two J-coupled atoms. In this approach, an additional set of features was added to indicate whether a given atom is closer to one or the other of the j-coupled atoms. We define the J-coupled atoms as atom-0 and atom-1. E.g. for the interaction type 1JHC, the hydrogen atom H is always atom-0 and the carbon atom C is always atom-1. Then, the cosine of the angle between any given atom-x, the center point and atom-1 is added to the features. This feature is automatically scaled between 0 and 1. A value close to -1 indicates that atom-x is closer to atom-0 while a value closer to 1 indicates proximity to atom-1. The experiment can be reproduced by running the following Kaggle-notebook: `https://www.kaggle.com/rpeer333/only-distance-type-and-angle-of-10-nearest-atoms`

Adding this set of features to the ones described in the previous section gave 31 features. While some more (but not all) geometric information is provided in this approach, the other two issues described in the previous section are not addressed by this change. The results were slightly better than in the previous experiment, but the difference is not outstanding (see Table 2.1) indicating that the main issues are indeed the two problems of the features not being permutation invariant and the lack of representation of the relations between features (the type of a given atom, its distance to the center-point and its cosine).

The same issues would apply to any other manually engineered features. Thus, these results provide good motivation to explore a MPNN approach shown in the next section.

MPNNs for edge property prediction

The *SchNet* [2] model is described using the graph convolution terminology, but it could equally well be described in the message passing framework proposed by Gilmer et al. [1]. The most relevant part of the architecture is the use of what they call *continuous-filter convolutional layers*. In message passing terminology, this refers to the use of the distance as a continuous edge feature. In the application of this model to the Kaggle-competition at hand, this feature is not thresholded. This means that an edge with distance as edge feature is defined between any two atoms in the molecule. In that manner, a complete graph is created where every atom is connected to every other atom in the molecule. It stands to reason that the success of the model might rely on the use of complete graphs rather than on the use of the continuous distance feature. The next section will show a similar example.

In this experiment, a *SchNet* model was implemented using the Python library DGL (deep graph library). An implementation in the *Chainer* framework for the same competition was used as a template [3]. Several changes from the template as well as from the paper [2] were implemented and tested, but the performance was far off the one from the template as well as the performance of the LightGBM models described above in Section 2.1.1 (see Table 2.1). For this reason, no further experiments were conducted with this architecture. The full implementation of the model is available in this notebook: `https://www.kaggle.com/rpeer333/pytorch-dgl-schnet-a-graph-convolutional-net`.

Instead, using the template from Footnote 3 and creating eight different models for the eight different J-coupling types (instead of just one model in the template) was much more effective. Furthermore, adapting the layer-dimension (smaller dimension for less complex interaction types and vice versa) as well as longer training time gave good enough results to achieve a placement in the top 10% in the competition (see Table 2.1) [4]. The experiments described in this and the previous sections show that achieving competitive results with a truly novel approach would take more experience and more time while simply by taking a working solution and adapting it properly to the use-case at hand can give good results within reasonable time.

Predicting molecular properties from 3D structure

During the practical work for this thesis, I participated in the *Alchemy* competition [5]. This competition provided a labeled training- and validation-set as well as an unlabeled test-set of 3D molecular structures described in more detail in Section 3.1.2. All remaining experiments in this thesis have been conducted with this dataset.

Despite many experiments (see Table 2.2), I achieved the best results simply fine-tuning the model from Chen et. al. [16] and finished 26th out of 53 participants (where

[3]`https://www.kaggle.com/toshik/schnet-starter-kit`

[4]Finishing as 198th out of 2749 teams (and receiving a virtual Kaggle-medal).

[5]`https://alchemy.tencent.com`

best MAE was 0.0162 and the best few solutions were very close to this value) [6]. The fact that half the participants did not beat the baseline published in the official competition paper shows that the approach is indeed quite effective and warrants a closer examination.

Experiment	**Section**	**MAE**
Baseline (Chen et. al. [16])	2.2.1	0.0569
root-node (graph-state)	2.2.2	0.0608
implicit hydrogen representation	2.2.3	0.091
independent message passing weights	2.2.4	0.0595
raw data only	2.2.5	0.0573
direction vectors as edge-features	2.2.6	0.0559

Table 2.2: Summary of the results of different experiments with the *Alchemy* dataset. All experiments until the raw data experiment include a few manually engineered chemical features to allow a direct comparison with the baseline. After the raw data experiment, these features were omitted in all further experiments, as they were shown to have no effect on the results.

Message passing gives poor results on molecular structures when restricted to the local neighborhood

Before going into the details, let us remember that locality is one of the key principles of regular convolutional networks [7]. Likewise, message passing has the same principle of locality that a node receives messages from its neighboring nodes only. It is exactly for this reason that message passing is also called graph-convolution. However, we will see that the performance is poor when following this principle and the best results are instead achieved when each node receives messages from all other nodes at every message passing step. Hence, the title of this section could also be formulated as ”Using message passing in the way it is meant to be used gives poor results on molecular structures”.

In the paper Chen et al. [16], the authors modify a well performing MPNN for quantum chemistry published by Gilmer et al. [1]. The authors expand this model by using

[6]`https://alchemy.tencent.com/#leaderboard`

[7]Recall that a convolution layer can be viewed conceptually as starting with a fully connected layer, removing non-local connections and using weight sharing across the remaining connection

both, categorical bond type and euclidean distance as edge features and observe considerably better performance. At first glance, the use of both kinds of features - which seems to be implied in the paper - looks like a plausible explanation for the success. However, upon closer inspection, the reasoning does not add up. Bond types almost completely determine the euclidean distance. E.g. every carbon-carbon single bond has a distance of 1.54Å (1 Ångström = 0.1nm), every carbon-oxygen double bond is 1.20Å long, etc. [22] (The same applies to the reverse: knowing the distance between two atoms as well as the atom types allows to deduce the type of covalent bond or lack thereof.) Therefore, no additional information is added by including the euclidean distance as a feature. A closer inspection of the implementation reveals another possible reason for the good performance: Not only is the euclidean distance added as a feature to existing edges, every atom is connected to every other atom in the molecule with the euclidean distance being the only not-null feature [8]. Thus, the structure of the graph is changed dramatically from the graph of covalent bonds to a complete graph where every node is connected to every other node. The complete graph is a special case of graph and is rather untypical because in most applications, the presence or absence of an edge between two nodes is the most important kind of information. Simply connecting all nodes with each other seems to diminish this information (although in this example, the euclidean distance allows to distinguish between important and less important edges). With these considerations in mind, it is surprising but very interesting that the approach of Chen et al. [16] works so well.

As explained in the introduction (Section 1.3.2), we can define the neighborhood of a node as all other nodes within a certain distance threshold. We refer to this threshold as neighborhood radius. In the first experiment, we compare the learning curves of graphs when defining the neighborhood with different thresholds. First of all, covalent bonds are always represented as edges in the graph. Then, for increasing radii, edges to all non-covalently bonded atoms within the radius are added to the graph. A neighborhood radius of zero thus means that no additional edges are added to the graph - the graph only has edges between covalently bonded atoms. At a radius of 2 Ångström (1Å = 0.1nm), an edge is added between any two non-covalently bonded atoms with a distance of at most 2Å. The same is done for 3Å, etc. For perspective, most covalent bonds in

[8]`https://github.com/tencent-alchemy/Alchemy`

organic molecules have a distance of around 1 - 1.5 Å [9]. 2Å is very close for non-covalently bonded atoms, such that only few edges between non-covalently bonded atoms will be added to the graph. With a 3Å radius, the number of added edges by far exceeds the original number of edges in the graph. With a 4Å radius, many atoms will be included in the neighborhood that have very little to no interaction with the central atom. At 5Å, for many small molecules, almost every atom is considered a neighbor of almost every other atom - applying this radius is therefore similar to creating a complete graph. Finally, with an infinite radius, a complete graph is obtained.

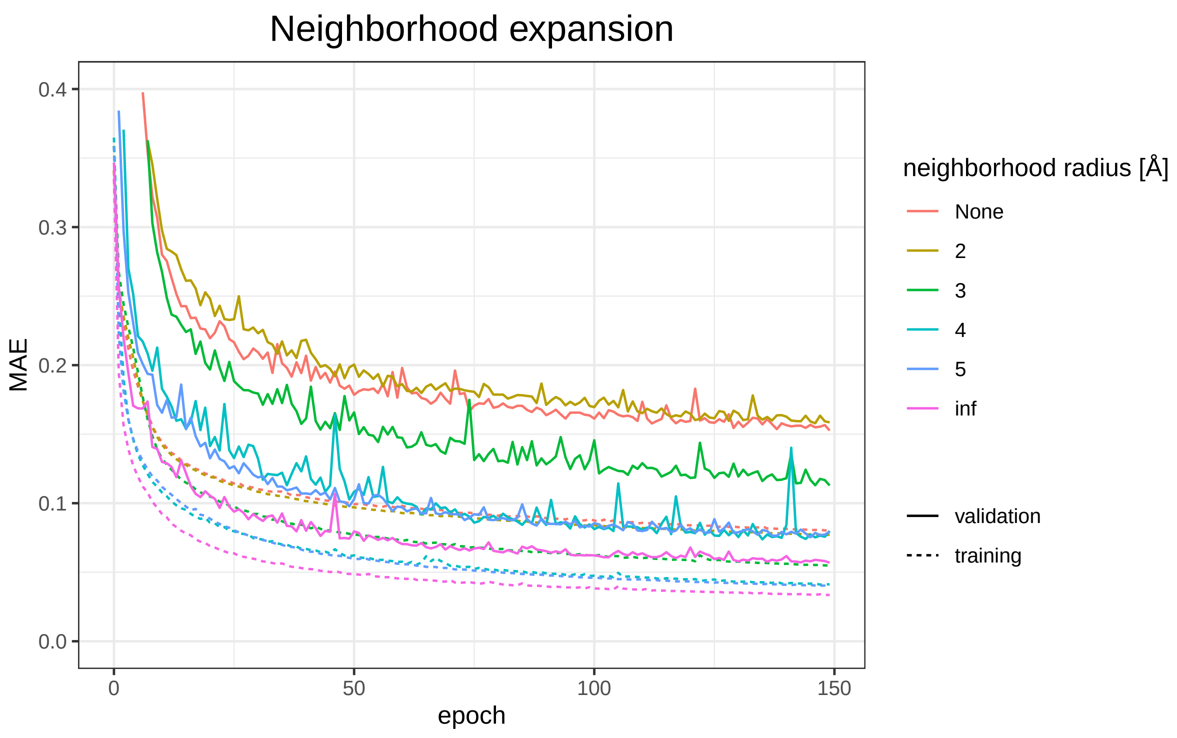


Figure 2.1: Learning curves of models trained on molecule graphs with different neighborhood radii are shown over the course of 150 epochs. The color corresponds to the neighborhood radius. Dotted lines show the training- and solid lines the validation-errors. Note that for each radius, there is a training- and a validation-error curve. Each curve is an average of three learning curves with the same neighborhood radius. All curves were generated using a simple optimization scheme described in Section 3.4. Both, validation- and training-error could be lowered considerably by employing a more sophisticated optimization scheme. However, this would come at the expense of comparability between curves (see Section 3.4).

As Figure 2.1 shows, the best performance is indeed achieved with the complete graph - connecting every atom with every other atom in the molecule with an edge. In general, the larger the neighborhood radius, the better the performance. This result is in line

[9]`https://courses.lumenlearning.com/suny-potsdam-organicchemistry/chapter/1-3-basics-of-bonding`

with the suspicion that the real reason for the good performance of the model of Chen et al. is the addition of new edges rather than the addition of the euclidean distance feature. The result is somewhat counterintuitive and in contrast with the principle of message passing. As described in the introduction, in theory, message passing updates node-representations based on the local neighborhood of each node. Eventually, after t layers, each node representation would contain indirect contributions from all other nodes with t or fewer edges apart. However, the results at hand show that the performance is best if every message passing step considers all atoms. This contrast between theory and empiric results empirically shows that much remains to be discovered about the use of MPNNs for molecules.

Naturally, in the wider MPNN literature this does not receive much attention, as the use of complete graphs is not computationally feasible in most applications. Only in the case of small molecule structures is it possible to do so, but, even in this field, the question is not frequently addressed. As discussed in the introduction, complete graphs are used, but not discussed, in the baseline architecture of the *Alchemy*-competition [16]. The paper describing the 3rd best solution - the only winning solution with a published paper and almost as good results as the best solution - does not explicitly state if a distance cutoff is being used for the *Alchemy*-dataset (i.e., only defining an edge between atoms withing a given distance) [23]. However, given the approach in the baseline architecture, I suspect that they use complete graphs as well or at least a very large distance cutoff that approximates a complete graph. Furthermore, the winning solution of the Kaggle-competition 'Predicting Molecular Properties' [10] also considers interactions between any two atoms at every step (they do not use a traditional MPNN, but nevertheless represent pairs of atoms in the model without a distance cutoff - which is equivalent to the use of complete graphs in MPNNs).

For small molecules and with considerable computational power available, one could simply accept this finding as it is and represent molecules as complete graphs. However, for larger molecules with hundreds of atoms, this would be computationally not feasible, as the number of edges increases quadratically ($\frac{n(n-1)}{2}$ edges for n nodes). Therefore, the next experiments will aim at finding an alternative way that captures the advantage of the complete graph-approach without the excessively high computational cost.

[10] `https://www.kaggle.com/c/champs-scalar-coupling/discussion/106575`

Introducing a root-node does not improve results

The use of complete graphs in graph MPNNs essentially ensures that during every node update, a representation of every other node contributes to the update - even nodes that are too far away to have any direct interaction with the updated node (atom). A possible explanation for this behavior is that the node update benefits from having access to information about the whole graph (molecule). This interpretation begs the question if a similar effect can be achieved with a different architecture that does not require the use of computationally expensive complete graphs.

An interesting experiment to test this hypothesis is to include graph-level information at every node update. One such idea is the use of a root node. A root node (called master node in Gilmer et al. [1]) is a virtual node that is connected to every other node in the graph. In the molecule case, the root node does not represent a real atom. Instead, it can be used to represent information about the whole molecule. Then, as every node is connected to the root node, every node update has access to information about the whole molecule. One could initialize the root node representation with molecule level features (such as number of atoms, electric dipole, etc.). However, all those features are implicit in the 3D molecular structure. Sticking to the pure deep learning philosophy, the root node representation can also be learned solely from the raw data.

Ideally, the use of such a root node would allow to achieve a similar performance as using complete graphs without the high computational cost. While the number of edges increases quadratically with the number of nodes in the complete graph, the root node only introduces one new edge per node. At the very least, the negative effect of increasing error with decreasing neighborhood radius shown in Figure 2.1 should be less pronounced if the root node has the desired effect.

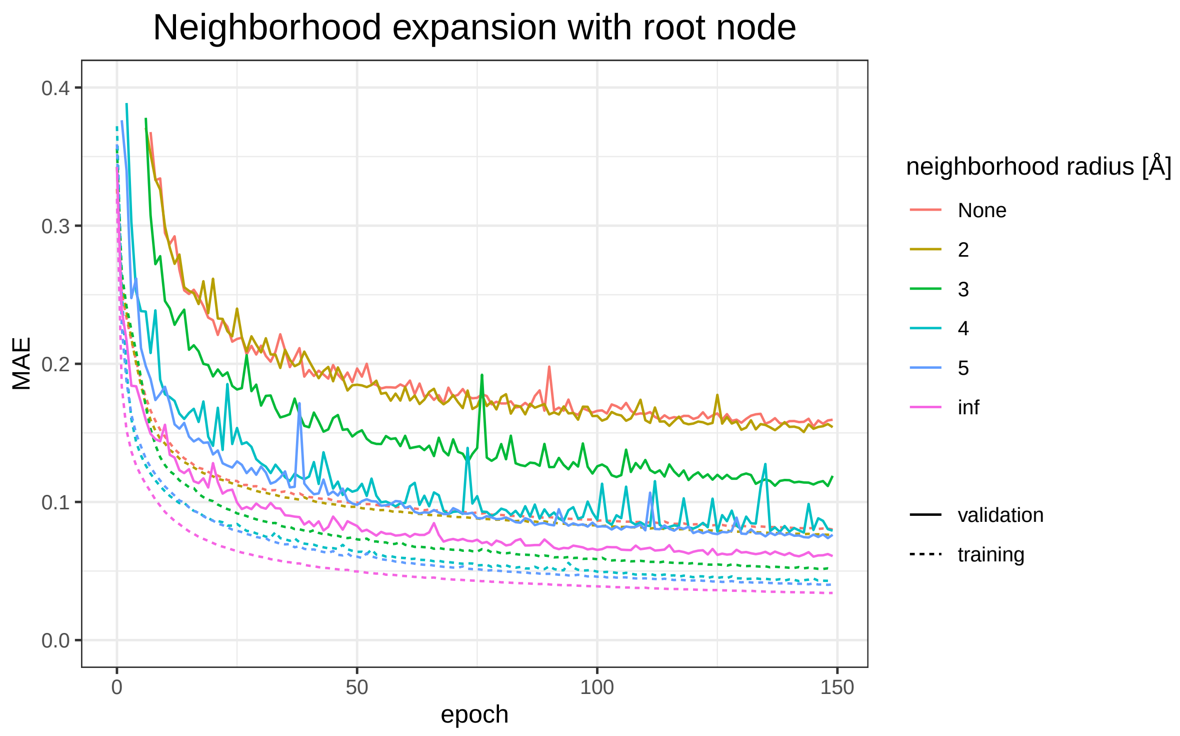


Figure 2.2: The results illustrated by this figure have been produced in exactly the same manner as Figure 2.1 with the addition of a root node as the only difference. This allows a direct comparison between the figures, such that any difference could be attributed to the presence or absence of a root node. The results show no difference beyond the small stochastic variation due to the random initialization of the weights. Thus indicating that the addition of a root node is no substitute for the use of complete graphs when representing molecules.

Figure 2.2 shows the same experiment as Figure 2.1 with the addition of a root node as the only difference (Section 3.2.2 describes how the concept of the root node is implemented in practice in the model at hand). Unfortunately, there is no noticeable difference between the results of the two experiments. This indicates that the superior performance of complete graphs is not solely due to the fact that every node update includes input from the whole molecule. A possible reason for the failure of the root node to replace a complete graph approach could be that the root node does not contribute geometric information about non-neighboring atoms. On the other hand, in the complete graph, each node's representation is considered in the update in conjunction with its distance from the updated node. Recall Equation 1.1 which defines the message from a node to another node as a function of both nodes as well as the connecting edge. While a root node representation can encode information about the whole molecule, such as number and types of atoms, their distance to the updated node (atom) is missing.

Excluding hydrogen atoms from the molecule-graph speeds up training, but increases the error

Another possible preprocessing step is to disregard hydrogen atoms. The rationale behind this step is that the chemical properties of a hydrogen atom depend strongly on the heavy atom to which it is bound. While some information is lost during this step, the average number of atoms in the molecule is reduced from 21.6 to 9.7 in the training-set. This modest reduction in the number of nodes leads to a very dramatic reduction in the number of edges. Naturally, this effect is most pronounced in complete graphs where the number of edges increases quadratically with the number of nodes ($\frac{n(n-1)}{2}$ edges for n nodes). While the average number of edges using complete graphs is 455.5 in the training-set, this number drops to 85 after excluding hydrogen atoms. Thus, the data representation becomes much more compact leading to a dramatic decrease in training time from around three minutes per epoch to only 0.9 minutes. This is a strong incentive to investigate whether ignoring hydrogen atoms in the molecular structure is a viable option. Figure 2.3 clearly shows that this is not the case. The loss is far higher when using the hydrogen-less structure compared to the full structure. Similar findings have also been reported in the literature [1]. For these reasons, every other experiment in this thesis uses full structures including hydrogen atoms.

Further information: Why a hydrogen-less structure contains implicit information about hydrogen atoms. Despite dropping all hydrogen atoms from the molecule structure, the information regarding the number of hydrogen atoms and their bonds is still implicitly contained in the structure. (For the same reason, hydrogen atoms are also excluded from standard molecule graphs, such as the one shown in Figure 1.2.) In short, it is a way of summarizing the raw data that speeds up training considerably while loosing some information.

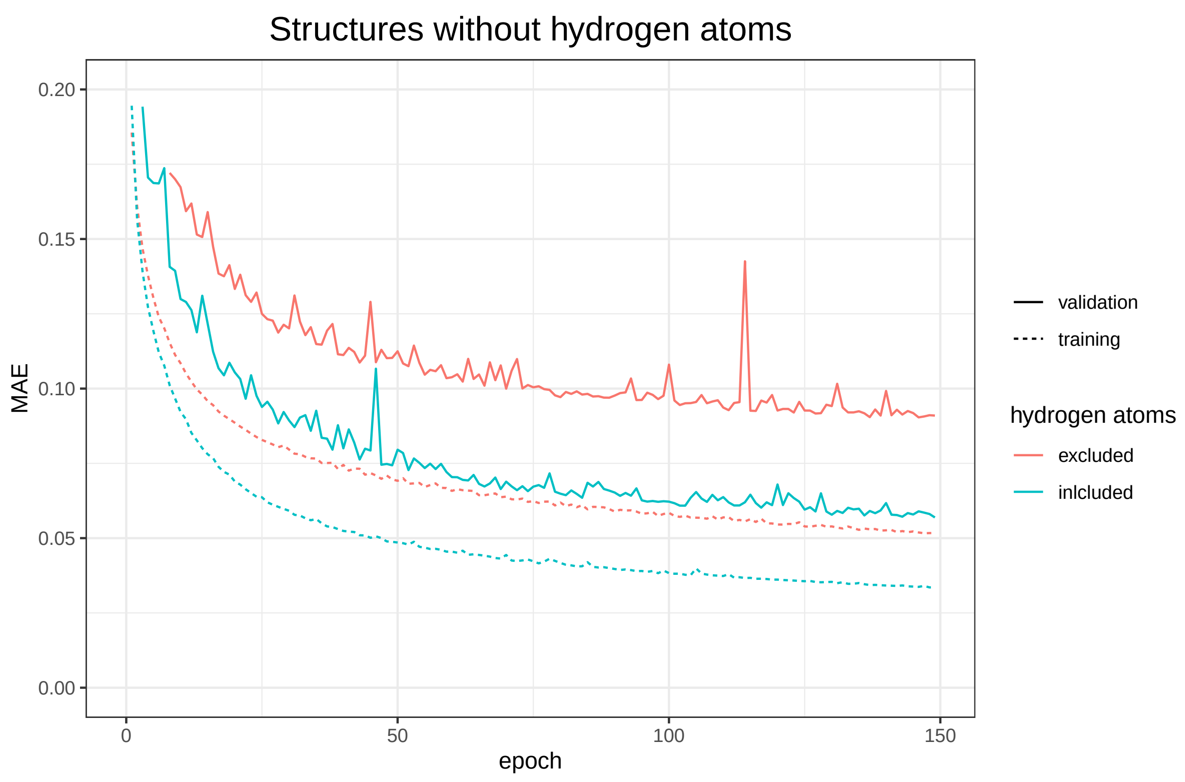


Figure 2.3: Input graphs with (explicit representation) and without hydrogen atoms (implicit representation). The model architecture is exactly the same for both learning curves, such that the lack of hydrogen atoms is the only factor causing the considerable difference. The result shows the importance of including hydrogen atoms in the graph representation of organic molecules.

The only piece of knowledge from organic chemistry required to understand this is that every given atom type, such as carbon or oxygen, only engages in a fixed number of bonds: four for carbon, two for oxygen, one for hydrogen, etc. (with double or triple bonds counting twice or thrice, respectively). This number is called valence and is dependent on the number of electrons in the outer orbital - called valence electrons [22]. In a molecular structure without hydrogen atoms, the number of bonds to heavy atoms (i.e.„ non-hydrogen atoms) is given for each atom as the number of its neighbor nodes. The number of hydrogen atoms bound to any given atom is thus its valence minus the number of neighbor nodes. It is for this reason that omitting hydrogen atoms from the structure can be thought of as equivalent to adding the feature 'number of hydrogen atoms' to each heavy atom. While this number is implicit in the structure, the exact position of the hydrogen atoms is lost when they are omitted from the structure.

Message passing functions with and without shared weights give the same results.

Recall the message function M_t and the (node-)update-function U_t defined in Equation 1.1 and 1.2, respectively. In this general definition, each graph-convolution layer (each message passing step) t has its own M_t and U_t. However, in most implementations of MPNNs, $M_t = M \ \ \forall\, t$ and $U_t = U \ \ \forall\, t$, which simplifies the two equations 1.1 and 1.2 to

$$m_v^{t+1} = \sum_{w \in N(v)} M(h_v^t, h_w^t, e_{vw}) \tag{2.1}$$

and

$$h_v^{t+1} = U(h_v^t, m_v^{t+1}). \tag{2.2}$$

In plain English, this means that the weights between the message passing steps are shared. This allows for efficient training and works well in practice. However, there is no theoretical reason why weight sharing should be necessarily superior. For instance, it would be conceivable that non-shared weights would work better for extracting different kinds of features at different message passing steps. Recall that in the first step, the input for a given node update consists only of direct neighbor nodes, at the second step, the update indirectly contains information from all second-degree neighbor nodes, etc. Hence, learning different weights for different message passing steps might be beneficial. Note also that in regular convolutional networks used for computer vision, every layer has non-shared weights even within blocks of convolutional layers with the same dimension (where weight sharing would be possible in theory). It is therefore far from clear that weight sharing has to be the preferred approach for graph-convolution.

In order to investigate the possible benefit of non-shared weights, the next experiment compares the MPNN described in Section 3.2.2 with an architecture without weight sharing. The two architectures are identical in every other aspect to isolate the effect of weight sharing. Figure 2.4 shows that there is no noticeable difference between the learning curves.

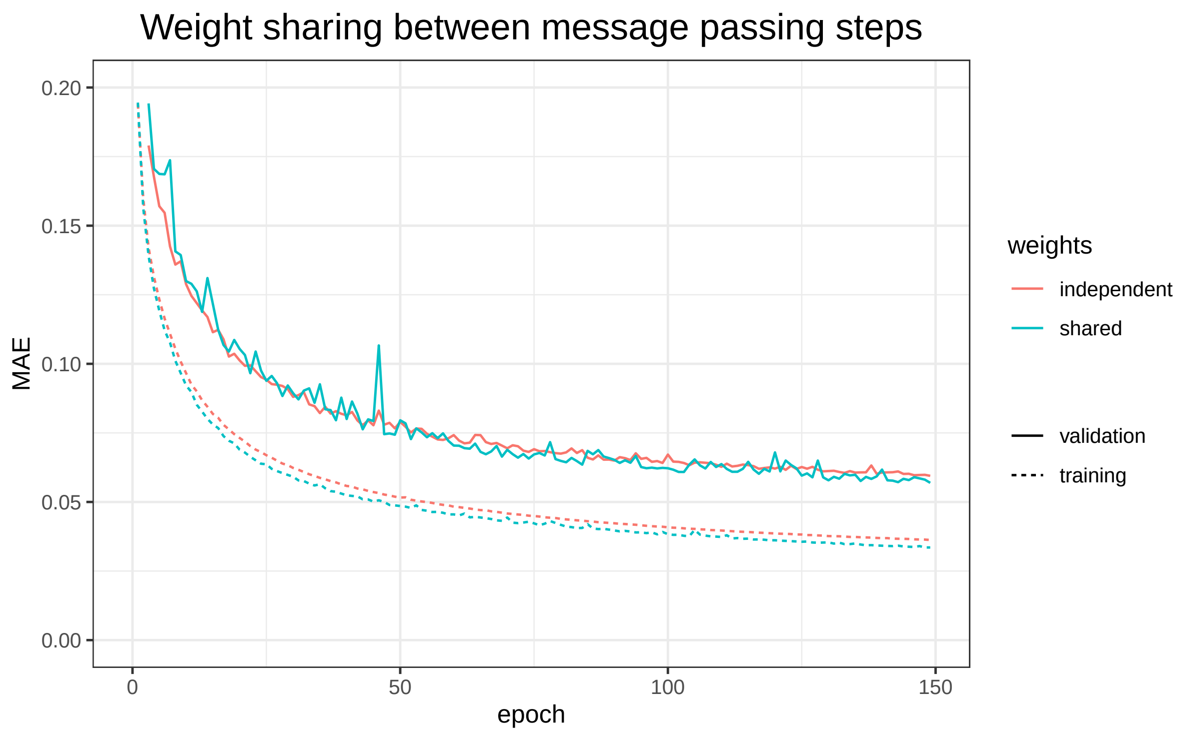


Figure 2.4: The result of using non-shared weights between message passing steps is the same as when using shared weights. The two architectures are identical except for the weight-sharing in the message passing functions. Complete graphs were used as the input data because this was shown to be the best data representation in earlier experiments (see Section 2.2.1). All learning curves were generated using a simple optimization scheme described in Section 3.4.

Another interesting observation is that the learning curves are identical not only in terms of the lowest MAE they achieve, but also in the speed at which they converge to this MAE. It would not have been surprising for the non-shared weights to require more training time, as each message passing step's weights are only updated once for a given batch in a given epoch. In weight-sharing architectures on the other hand, each message passing step is updated T times for each batch and epoch in a model with T message passing steps. Nevertheless, Figure 2.4 clearly shows that the two approaches are equal in terms of training-time.

One caveat worth mentioning is that only because non-shared weights were not found to be beneficial for the architecture at hand, this does not mean that they will never be beneficial in other architectures.

Furthermore, while we can see that they do not decrease the loss, we still do not know why exactly that is the case. This question is not addressed frequently in the literature - instead, the use of weight-sharing is most commonly accepted without much investigation into why it is used [16] [2] [23]. Other studies achieve good results without weight-sharing, but do not comment on the reasoning for choosing this approach or whether the model would work equally well with weight sharing [15]. It is far from obvious why there is no difference between shared and non-shared weights in MPNNs. Finding out why this is the case might also lead to insights about how to build better MPNNs.

Summing up, the results confirm the common practice of using weight-sharing across message passing steps, but the question, why weight-sharing works well in MPNNs, is still left open and worth investigating.

Manual feature engineering does not improve results compared to using only raw data

The previous experiments included some hand-crafted features calculated with the computational chemistry Python library *rdkit* [11]. This was done in order replicate the approach of Chen et al [16]. In the literature, there are conflicting indications about the use of manually engineered chemical features as input for MPNNs. For instance, the three best solutions of the *Alchemy* competition show almost equally good results [12], but differ in their assessment of the benefit of manually engineered features. While the winning model uses such features and the authors claim that they improved their results, the second and third solutions (with virtually the same MAE) state that manually engineered features provided no benefit and were thus not used [23] [13]. In most cases, the literature lacks a direct comparison between using only raw data (atom-type, position and edges) and adding manually engineered features. It is therefore often hard to deduce whether the good results are due to an effective model or due to the smartly engineered features used as input.

The next experiment trains two almost identical MPNNs: one uses the same chemical features as Chen et. at. [16], while the other one uses only atom-type, edge-type and distance as input. The two nets are identical in every aspect except of course for the

[11]`https://www.rdkit.org`

[12]`https://alchemy.tencent.com/#leaderboard`

[13]*Alchemy* competition winners:
1th: `https://alchemy.tencent.com/data/2019/1st_solution-ape-MPNN_NJU_Chem.pdf`
2nd: `https://alchemy.tencent.com/data/2019/2nd_solution-SchNet_and_SchNet_with_Edge_Updates_1117.pdf`
3rd: [23]

different dimension of the input layer. All chemical features [14] are computed from the structure without additional information. Hence, if they are relevant for predicting the target variable, they are exactly the kind of information that message passing should extract from the structure. If that's the case, the addition of the computed chemical features is expected to have no effect on the model's loss. Indeed, Figure 2.5 shows no difference between the two types of input data.

Therefore, this allows to conclude that the message passing functions are able to extract relevant chemical information to predict the target variable. The results do not preclude that another architecture might benefit from the addition of manually engineered chemical features or that more sophisticated features might provide some benefit for the model at hand. Nevertheless, the results show that manual feature engineering is far from essential and strengthen the believe that the deep-learning approach of using raw data only is also preferable for chemical applications of MPNNs.

All further experiments are conducted without calculated chemical features and only use the raw structural data instead.

[14]The chemical atom (node) features are comprised of three boolean variables indicating whether the atom is an electron donor, an electron acceptor and whether it is part of an aromatic ring. Furthermore, they include a categorical variable indicating the hybridization type. As all of those are functions of the atom-type or the local structure, it comes as no surprise that they are redundant.

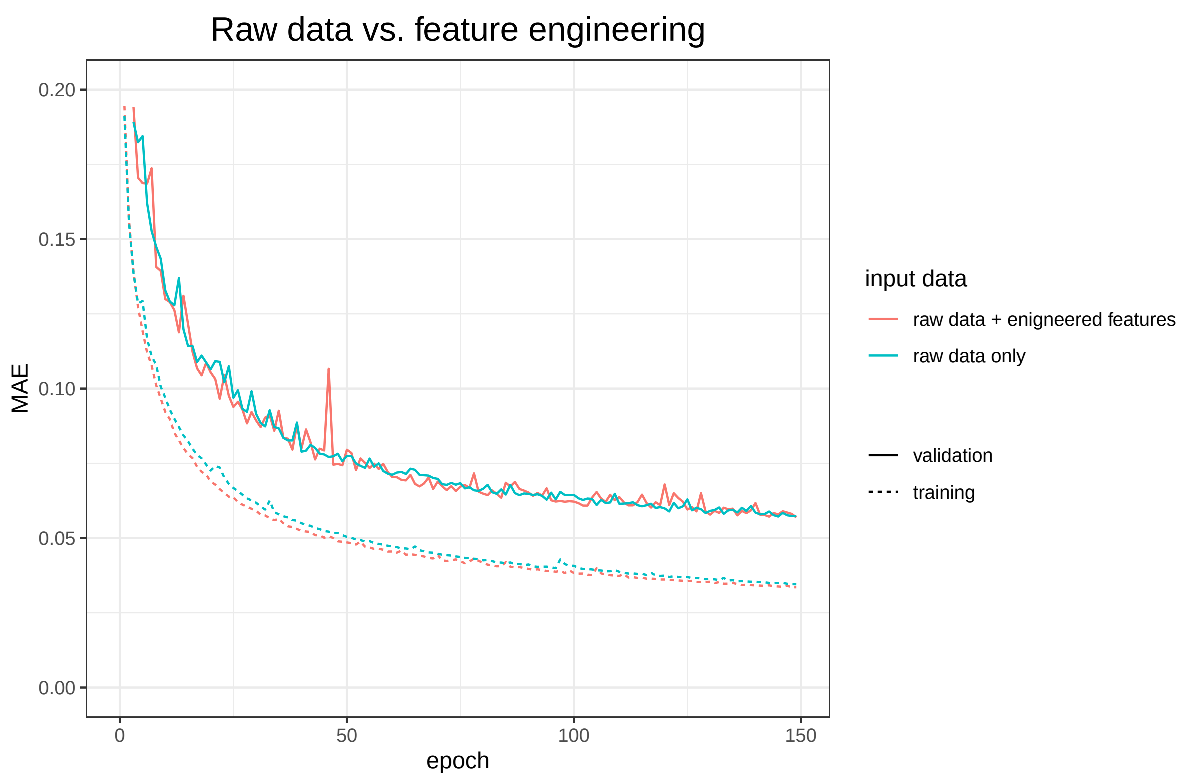


Figure 2.5: Raw data alone gives equally good results as raw data plus manually engineered features. As in all experiments in this thesis, each learning curve is an average of three independent learning curves using the simple optimization scheme described in Section 3.4.

Direction vectors as edge features have no impact

It stands to reason that one of the main limitations of using MPNNs for data embedded in spacial dimensions is the loss of a part of the spacial information, as explained in Section 1.3.3.1. (While this thesis focuses on molecules, representing any other 3D data as graphs would suffer from the same limitation).

In an attempt to address this limitation, an additional (vector-valued) edge-feature was included as input into the message-function M (Equation 1.1): the direction vector from the sending atom h_w to the receiving (i.e.„ to be updated) atom h_v was added to the features of edge e_{wv}. This vector is invariant to translation of the molecule making it potentially more informative than using raw atom-coordinates as node-features [15]. However, the direction vector is still not invariant to the arbitrarily chosen rotation of the molecule in the coordinate system.

[15] Adding the raw coordinate values to the node features gave no improvement at all. As this is expected given that the coordinates depend on the arbitrary positioning and rotation of the coordinate system, the results are not shown here.

In order to provide the direction vector with some useful information, the molecule's center was chosen as the origin of the coordinate system. Thanks to this transformation, a direction vector $pos(h_v) - pos(h_w)$ from atom h_w to atom h_v always provides the information if h_w lies closer to the center (positive values) or closer to the periphery of the molecule (negative values) in relation to the updated atom h_v.

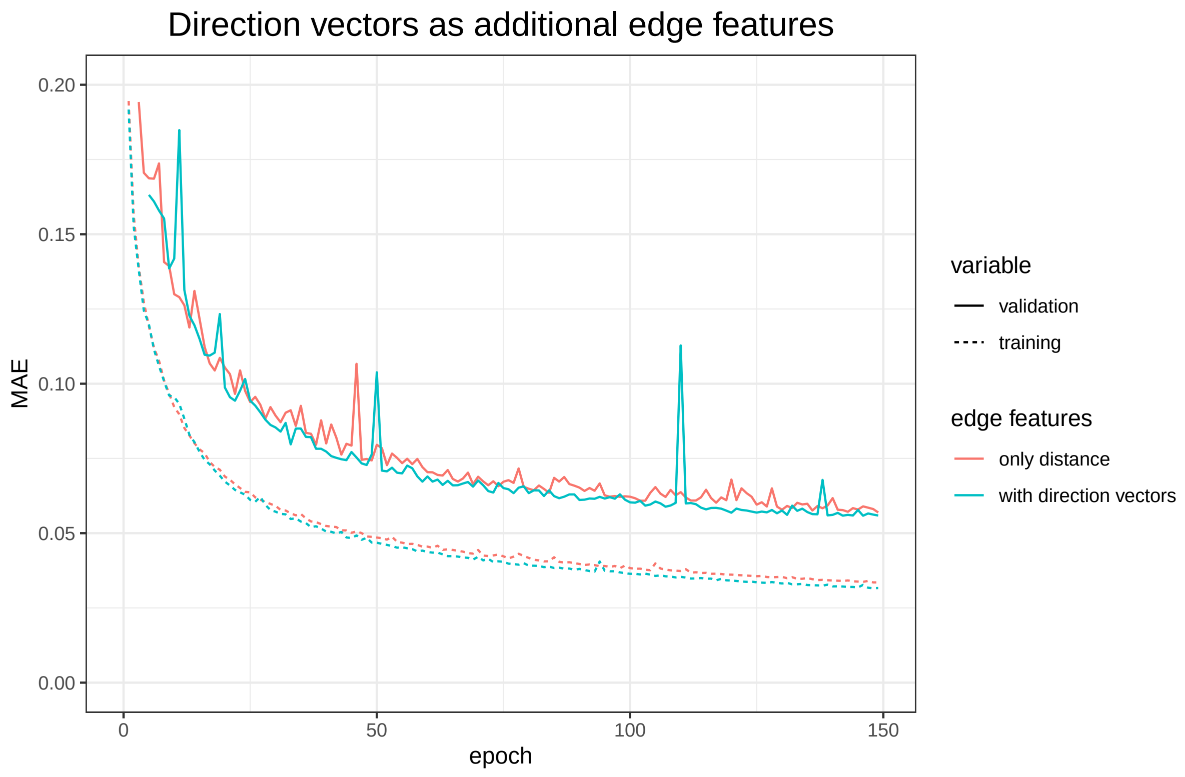


Figure 2.6: Direction vectors as edge features do not improve the results. As in the previous experiments, each line is an average of three independent learning curves, which were obtained using the simple optimization scheme described in Section 3.4.

For this reason, the direction vector could contain useful information despite the lack of rotational invariance. Nevertheless, no improvement was achieved by the addition of the direction vector to the edge features, as shown in Figure 2.6. The results do not give evidence whether the additional edge features do not contain relevant information at all or whether the message passing functions were just not able to extract useful higher-level features from this information. The possibility that another graph-convolution architecture might benefit from using direction vectors as edge-features cannot be ruled out completely, but the results at hand do not provide evidence for it.

An interesting attempt at modeling 3D structural information in MPNNs is the 3D-embedded graph convolutional network of Cho et al. [25]. However, upon closer inspection, the approach seems to suffer from the same issue as the experiment above - the lack rotation-invariance in the input-data. Adapting this model for the *Alchemy* dataset gave only disappointing results. After 200 epochs, the validation MAE plateaued at around 1.8, which is about three times higher than the MAE achieved with the other models in this thesis summarized in Table 2.2. Even the training error did not even go below the validation error achieved with the simple MPNN, but plateaued at around 0.85.

In summary, incorporating 3D structural information in MPNNs is a highly complex task, which requires a lot of further research, but has the potential of achieving substantial improvements.

CHAPTER 3

Data, models and evaluation

Data

The CHAMPS dataset

This dataset has been published for the Kaggle competition 'Predicting Molecular Properties' organized by CHAMPS (CHemistry And Mathematics in Phase Space) [1]. The results from experiments using this dataset are presented in Section 2.1.

The target variable is a magnetic interaction between two atoms measured by Nuclear Magnetic Resonance (NMR) - called scalar coupling constant or J-coupling constant [26]. In graph notation, this represents an edge-property (see Section 1.2.2). This interaction does not have to be predicted for any two atoms in the molecule, but rather for a predefined set of atom-pairs. These can be directly bonded to each other or be two or three covalent bonds apart. Furthermore, they can involve different atom types. The interactions are categorized into eight types by the involved atom types and the number of bonds between them. For instance, $2JHC$ is the J-coupling type of a hydrogen atom (H) and a carbon-atom (C) with two bonds between them.

The training-set contains 85012 molecular structures with 4659076 labeled J-coupling interactions, while the test has 45777 molecules with 2505190 relevant interactions (whose labels were not visible to the user at the time of the competition, but only used for automatic evaluation). The dataset was split on molecule level meaning that either all J-coupling interactions in a given molecule are part of the training set or all of them are in the test-set. For a more detailed investigation, please follow the links to two

[1]`https://www.kaggle.com/c/champs-scalar-coupling/overview`

Kaggle-notebooks in the footnotes [2] [3]. In a general model, the J-coupling type would be the most important feature as the ranges of the J-coupling constant differ markedly between the eight types. However, a better approach is to train a different model for each type. This approach has been shown to be effective with boosted tree-models as well as MPNNs and was therefore used in both cases.

Apart from the raw structural data, the dataset also includes calculated chemical features. However, these are all just functions of the structural data and the J-coupling type and thus not used in the experiments at hand. The same approach was also chosen for the main dataset described in the next section.

The evaluation metric is based on the mean absolute error (MAE). As with the regular MAE, a lower score means better performance. The additional complexity serves the purpose of ensuring that all eight J-coupling types contribute roughly equally to the score. [4]

$$score = \frac{1}{8} \sum_{t=1}^{8} log(\frac{1}{n_t} \sum_{i=1}^{n_t} |y_i \; - \; \hat{y}_i| \;) \tag{3.1}$$

The *Alchemy* dataset

The new quantum chemistry dataset *Alchemy*, created by Tencent Quantum Lab [16], was used for the bulk of the experiments in this thesis. The dataset is similar to the well established QM9 dataset [19] with the same quantum mechanical properties as target variables and a similar number of molecules. The main advantage of the new *Alchemy*

[2]`https://www.kaggle.com/rpeer333/molecular-properties-detailed-eda`

[3]`https://www.kaggle.com/rpeer333/champs-competition-baseline-trivial-predictions`

[4]To understand this equation, note that the central element is $|y_i \; - \; \hat{y}_i|$ which is the absolute error for an individual J-coupling interaction, with y_i being the real J-coupling constant and $\hat{y}_i$ the prediction. Keeping in mind that there are eight different kinds of J-coupling in the dataset, n_t denotes the number of J-coupling interactions of type t. Thus, $\frac{1}{n_t} \sum_{i=1}^{n_t} |y_i \; - \; \hat{y}_i|$ is simply the mean absolute error (MAE) of all J-coupling interactions of type t. Now, it is easy to see that the whole term is just the arithmetic mean of the log-MAEs of all eight J-coupling types. This somewhat more elaborate score is required because the number of interactions per J-coupling type varies considerably in the dataset. For the simple MAE over all J-coupling interactions, the score would be largely dominated by the more frequent J-coupling types, thus favoring a model focusing predominantly on those types. Calculating the log-MAE for all types independently and then averaging ensures a more equal contribution of all eight J-coupling types to the score. Finally, the only question remaining is, why the log-MAE is used instead of the simple MAE. The reason for this choice is most likely that the eight J-coupling types have very different variances. Thus, naturally, the MAE will be larger for the J-coupling types with larger variance. To avoid the score being dominated too much by those terms, the logarithm decreases the difference in contribution between low-variance and high-variance J-coupling types. Again, this leads to a more equal influence of the eight J-coupling types on the final score and requires the final model do reasonably well an all of them.

dataset is that it also contains larger molecules of up to 12 heavy atoms (i.e., non-hydrogen atoms), as opposed to the maximum of seven heavy atoms in the QM9 dataset. This property makes it more relevant for medical research as potential drugs are not limited to only very small molecules. There are 119487 3D structures of molecules in the dataset. These structures provide the 3D coordinates as well as the atom-type (carbon, hydrogen, etc.) for each atom. Furthermore, the information which atoms are covalently bonded with each other and the type of the bond (single, double, triple) is also provided (even though this information could be calculated from the atom-coordinates if needed). Finally, twelve important quantum mechanical properties [16] are given for each molecule [5]. The task is to predict those twelve properties [6] from the raw data.

Features

There are two different approaches to predicting the target variables. The pure deep learning approach is to use only coordinates and bonds and rely on the model to learn any features required to predict the target properties. After all, coordinates, atom types and bonds contain all the available information (actually the bonds could be reliably calculated from coordinates and atom types and thus, strictly speaking are also redundant information).

The other approach is to calculate all sorts of chemical features using chemistry Python libraries and feed them into the model alongside the raw data. These include node (atom) features (whether the atom is an electron donor or acceptor, whether it is part of an aromatic ring, its hybridization type, etc. [22]) as well as bond (edge) features. The fact that these features are calculated from the raw data without any additional information means that the model should also be able to learn them if required. Hence, calculating features and feeding them into the model should in theory be obsolete, but could slightly improve predictions if the model is unable to learn those features. Section 2.2.5 presents an experiment assessing the benefit of manually engineered features.

[5]Unfortunately, due to the issue described in Section 3.1.2.3, the test set labels could not be used for reliable evaluation. The remaining dataset consists of 99776 molecules in the training-set and 3951 molecules in the validation set.

[6]The 12 target variables (Dipole moment, Polarizability, HOMO, LUMO, gap, R^2, Zero point energy, Internal energy, Internal energy at 298.15 K, Enthalpy at 298.15 K, Free energy at 298.15 K, Heat capacity at 298.15 K) are shown in Table 2 of Chen et al. [16].

Benchmark and evaluation

The target variable in this dataset is a real-valued vector of length twelve. The error-metric used throughout this work is the mean absolute error (MAE) - which was also the metric in the competition. As the twelve quantum mechanical properties have vastly different ranges, they have to be normalized - otherwise the properties with larger ranges would contribute far more to the MAE than the properties with smaller ranges. Normalization was performed by subtracting the mean and dividing by the standard deviation.

Difference between competition data and full data in the *Alchemy* dataset

Overview. Unfortunately, the results obtained on the competition dataset cannot be directly compared to results achieved on the full dataset published after the *Alchemy*-competition. This section presents the result of an investigation into the data quality issues and explains why the test-set (in particular the target variables) could not be used for the experiments. If the reader is only interested in the main topic of the thesis rather than inconsistencies in the particular dataset at hand, this subsection may be skipped. On the other hand, if the reader intends to train a model on this dataset, then reading this section can save a lot of time.

Figure 3.1 shows the learning curves of models being trained on both, the competition dataset and the full dataset. The MAE values are considerably lower on the competition dataset.

Train-test split. The competition dataset is split into training-set (called dev-set by the hosts), validation- and test-set with the ground truth being provided only for the first two sets. The split is not random. Instead, the validation- and test-set have an intentional bias towards larger molecules compared to the training-set. Building models that can extrapolate from smaller molecules to larger molecules is useful for pharmacological research [16].

Figure 3.1 shows learning curves for both, competition-data split and random split to eliminate any differences in validation MAE hailing from the different dataset splits. The figure shows two counter-intuitive results. First of all, the MAE is higher on the full dataset despite its larger size. Naturally, the larger dataset should give better or at least equally good results. Secondly, the impact of the splitting method varies between the two dataset. The expected result would have been a lower MAE on randomly split data because the training- and validation-molecules come from the same distribution. Having a strong bias towards higher weight molecules in the validation set is expected to increase the MAE. For the full dataset, this is indeed the case. However, for the competition dataset, the random split gives a higher MAE than the biased competition-split. As this result contradicts the expectation, it was thoroughly tested and validated.

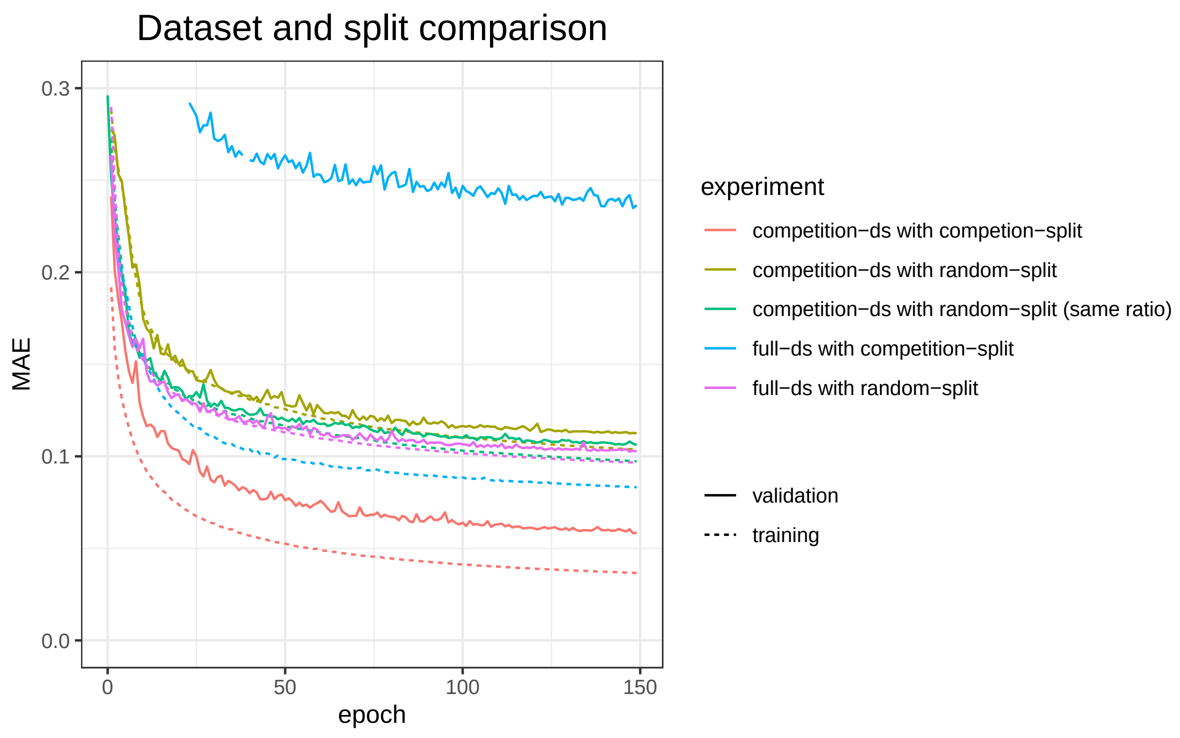


Figure 3.1: Comparing the full (post-competition) and the competition-dataset using various methods for splitting the data into training- and validation-set. Both datasets are evaluated using a random split (70% training, 15% validation, 15% testing) and the train-validation split as it was defined in the competition (with a bias towards higher weight molecules in the validation set). In addition, the competition data is also subjected to a random split where the fraction of the validation-set is exactly the same as in the competition-data-split. (This last experiment was conducted to make sure that the observed difference is due to the method of splitting and not due to the size of the validation set.)

Target variable normalization. In the competition dataset, the values are already normalized and the raw values are not available. It is not clear whether the mean and standard deviation used for normalization where calculated separately for the training- and validation-set or if the they are based on all molecules. Furthermore, the order of the target variables is unknown (by design) and their names are replaced with the dummy-names property-1, property-2, etc.

The full dataset released after the competition contains the raw values and the real names of the target variables. For training and evaluation, they were normalized in the same manner as in the competition (subtraction of mean and division by standard deviation). However, a direct comparison with the ranges of the competition dataset is not possible, as the names of the target variables are not provided there.

Molecule data. The raw data and thus the input into the model is exactly the same in the full dataset as in the competition dataset. This has been verified by thorough testing.

Conclusion. With the input-data as well as the data-splits being identical, the reason for the difference in MAE can only be explained by differences in the target variables. As discussed above, the absence of raw data in the competition dataset prevents a direct comparison of the target variables between the two datasets. Thus, no further information can be obtained to pinpoint the difference or to find a possible explanation. The only conclusion to be made is that for some reason the target variables in the new data prevent the model from learning to predict them as well as in the competition data. Therefore, only the training- and validation-set from the competition were used in the experiments. Furthermore, the same training-validation split as in the competition was chosen because the MAE of the competition-data with this split could not be achieved by any other splitting method. While it is unfortunate not to know the root of this counterintuitive finding, it will have to remain a mystery in order to focus on MPNNs, which are the core topic of this thesis.

Model architecture

SchNet

As this model has not been used for many experiments, we shall refer to the literature for a detailed description. The implementation follows the architecture proposed by Schütt et al. [2]. Several changes were made in this project's DGL-implementation [7] compared to the template *Chainer*-implementation [8]. In particular, a graph state was added. This is similar to the concept of a root- or master-node shown in Section 2.2.2. Furthermore, the residual short-cuts (first described in *ResNets* [27]) in the graph convolution

[7] `https://www.kaggle.com/rpeer333/pytorch-dgl-schnet-a-graph-convolutional-net`
[8] `https://www.kaggle.com/toshik/schnet-starter-kit`

layers were replaced with dense short cuts analogous to the ones from *DenseNets* [28]. Neither of those changes is responsible for the poor performance of the model, as the results were the same without them.

Base MPNN architecture

The basic architecture used as a starting point for most of the experiments in this thesis (Section 2.2) comes from the MPNN variant from Chen et al. [16] whose code is available on GitHub [9]. This model, in turn, is based on the MPNN proposed by Gilmer et al. [1].

Node embedding. The network starts with an embedding of the atom-type of each node. In this particular implementation, that is achieved by passing the one-hot encoded atom type through a regular linear layer. This first layer essentially extracts the n$^{\mathrm{nt}}$ column from the weight matrix when the n$^{\mathrm{th}}$ position of the input vector is one, which is easy to see from the following example.

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix} \tag{3.2}$$

Thus, the first linear layer essentially corresponds to a mapping from a categorical, integer-valued variable to a (learned) vector $\in \mathbb{R}^d$ where d is the output dimension of the linear layer, which defines the dimension of the node hidden state.

Message passing. The message passing function used in the experiments was the so called edge network message function, which was found to be the best among several options by Gilmer et al. [1]. The approach is described in detail and nicely illustrated by Simonovsky et al. [29]. This message passing layer is implemented in the *PyTorch Geometric* class NNConv [10].

The message function has two components. The first one is a two-layer MLP (multi-layer perceptron) $h_{\mathbf{\Theta}}$ taking the edge features as input and outputting a vector of dimension d^2, which is reshaped into a d by d matrix (recall that d is the dimension of the

[9] `https://github.com/tencent-alchemy/Alchemy`

[10] `https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.NNConv`

node hidden state).

$$\Theta_{vw} = reshape(\ h_{\Theta}(e_{vw})\ ) \tag{3.3}$$

In the second step, the sending node representation h_w is multiplied with this edge-conditioned weight matrix. Finally, an aggregation over all sending (neighboring) nodes gives the message for node v.

$$m_v^{t+1} = \sum_{w \in \mathcal{N}(v)} h_w^t \cdot \Theta_{vw} \tag{3.4}$$

It is easy to see how the two functions can be combined in a single message function. Finally, in the default setting, the update function is simply

$$h_v^{t+1} = m_v^{t+1} \tag{3.5}$$

If the root-node concept is used (by setting the parameter root-weight to 'True')[11], the update function becomes

$$h_v^{t+1} = \Theta_{root} h_v^t \ + \ m_v^{t+1} \tag{3.6}$$

Oddly enough, in the simple case of Equation 3.5, the updated node hidden state h_v^{t+1} is not a function of the previous hidden state h_v^t. While that is counter-intuitive, it was shown to work well in practice. Section 2.2.2 compares the same model with and without the root-node and finds no difference in performance.

Readout function. Finally, the readout function is composed of a *Set2Set* [17] [12] layer mapping the set of node hidden states to an aggregated representation with the dimension $\mathbb{R}^{2d}$. Then, the output from *Set2Set* is mapped to the target variable with a two-layer MLP. The readout-function can be formalized as

$$\hat{y} = MLP(\ Set2Set(\ \{\ h_v^T \ \mid v \in G\ \}\ )\ ) \tag{3.7}$$

[11] Technically, as Equation 3.6 shows, there is no root-node in this architecture. There is just the root weight-matrix Θ_{root}, which can be thought of as a representation of an edge between the virtual root node and any real node in the graph. However, that is just an implementation-detail of this particular architecture. It is still valid to see Θ_{root} as the concept of a root node, which can be extended to other architectures as well.

[12] `https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.glob.Set2Set`

Software

Most of the models presented in this thesis (Section 2.2) were implemented using the Python deep learning library *PyTorch* [13] and its extension for graph neural networks, *PyTorch Geometric* [14]. The *SchNet* model (Section 2.1.2) was implemented using the Python library DGL (deep graph library) [15]. In my humble opinion, *PyTorch Geometric* is by far superior to DGL. Its API is much more in line with *PyTorch*, the documentation is clear and the code works reliably. On the other hand, DGL is not straightforward to use and the work on the *SchNet* model was severely hampered by a memory leak. In order to fix this issue, all caches had to be cleared after each batch leading to extremely slow training and making it impossible to try different approaches within limited time. Summing up, *PyTorch Geometric* seems to be the best *PyTorch*-compatible library for MPNNs.

The Python library *mlflow* [16] was used extensively to track all parameters and results for each experiment. It essentially creates a small local database containing all information the user wishes to save for each experiment. The data can be browsed via a dashboard or accessed via a Python API. This combination of automatic logging, a human readable dashboard and an API makes it the perfect tool for reproducible machine learning research and is highly recommended by the thesis author.

The code of the models trained on the *Alchemy* dataset is available at github: `https://github.com/raph333/masters_project`.

Training

All experiments presented in this thesis use Adaptive Moment Estimation (Adam) for optimization [30]. For certain experiments, other optimizers might give slightly better results. However, comparison between differences in data processing and model architectures would be impeded by employing different optimization schemes because one would never know if the different outcomes are a result of the model or the optimization. For this reason, optimization strategies were kept as constant as possible for all experiments.

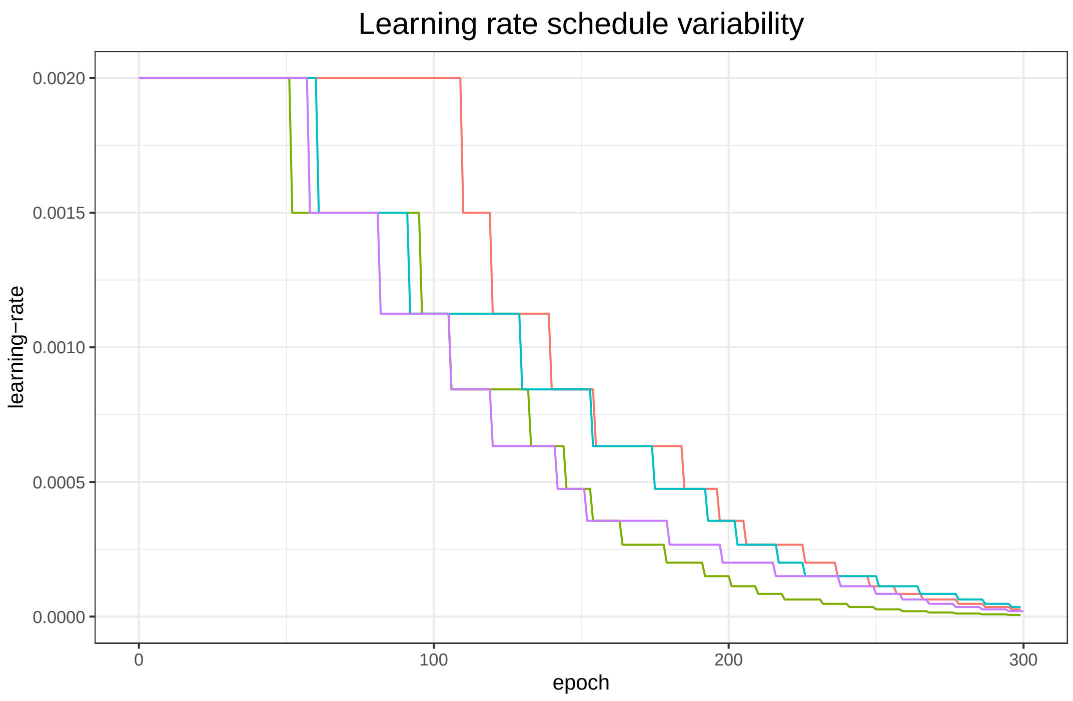
[13] `https://pytorch.org/`
[14] `https://pytorch-geometric.readthedocs.io/en/latest/`
[15] `https://www.dgl.ai/`
[16] `https://mlflow.org/`

The same considerations were applied to learning rates and learning rate scheduling. Two different optimization schemes were used for different purposes. To obtain the lowest possible test-set error, a learning rate schedule that reduces the learning rate after the validation error plateaus has found to be most effective [17]. This optimization scheme suffers from the drawback of requiring many training epochs. Furthermore, the resulting learning-rate schedule varies considerably even between trainings with the same parameters (Figure 3.2a), which leads to considerable variation in the validation error (Figure 3.2b).

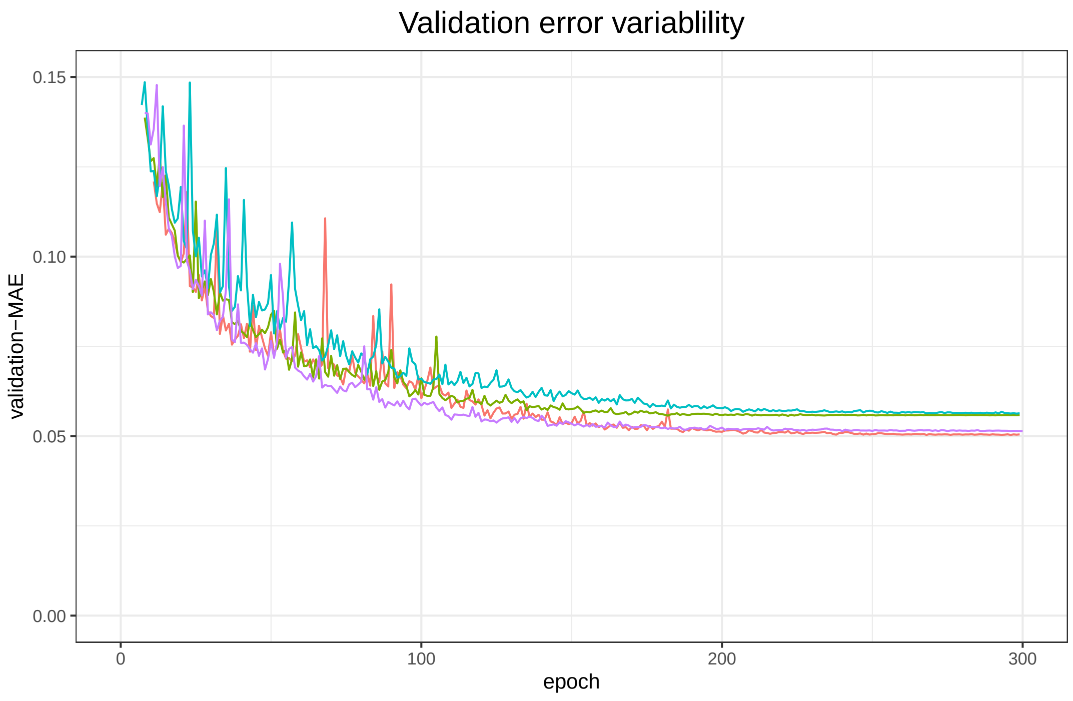
These differences as well as the long training time make it unsuitable for comparing learning curves with different parameters (however, it is a good strategy to achieve good results without spending a lot of time fine-tuning the optimization hyper-parameters). For comparing different learning curves, a simpler, exponential learning rate decay was chosen [18]. The advantage of this method is that at each epoch, all curves have the same learning rate making them comparable. The final validation error is usually higher than would be achieved with reduce-on-plateau learning rate scheduling (furthermore, it also requires a few trials to find suitable hyper-parameters). However, this drawback is accepted because the main point of the experiments is to compare different model architectures, parameters or data-transformations rather than to achieve the lowest possible error.

[17] Specifically, the following parameters were chosen: The initial learning rate of 0.002 is reduced by a factor of 0.75 if the validation error did not decrease by a value of 10^{-4} or more within the last 8 epochs. The training duration was 300 epochs.

[18] The initial learning rate of 0.001 is reduced by a factor of 0.995 after every epoch and the experiment is run for 150 epochs.



(a)



(b)

Figure 3.2: The two graphs show metrics from four identical training runs with the reduce-on-plateau learning rate schedule using the parameters given in Footnote 17. Despite the same parameters, the resulting learning rate schedule varies markedly, as shown in **(a)**. The reason for this behavior is that small stochastic variations of the validation error early in the optimization determine when the learning rate is lowered first, which, in turn, influences the entire further optimization process. As a consequence, the resulting learning curves of the validation error also show substantial variation, as shown in **(b)**. Therefore, this optimization strategy is not well suited for comparing different models because it is not possible to determine whether the difference is due to the models or just the normal variation of this optimization schedule.

Experiments

Most experiments show learning curves with validation- and training-error rather than just the final errors. This choice was made because, in addition to showing the final error-rates, learning curves allow to estimate whether the model suffers from high bias (when the training-error plateaus and is not much better than the validation- error) or high variance (when the training-error decreases rapidly, but the validation-error does not follow suit) [31]. Furthermore, learning curves also indicate if prolonged training could lead to further improvements or if a plateau has been reached. In contrast, a single error-metric does not show how this metric was obtained and thus offers little information to interpret the results. Furthermore, learning curve averaging of multiple training runs was employed to reduce the impact of noise. Some variation in the learning curves is to be expected even in the absence of any difference in model architecture, input data or optimization strategy. Therefore, any learning curve shown in this thesis is an average of three independent learning curves using the exact same models parameters and input data. Conducting multiple training runs for each setting and averaging the results reduces the impact of random variation and allows to see the systematic differences instead.

CHAPTER 4

Conclusion

In this master's project, I had the pleasure of investigating the exciting field of Message Passing Neural Networks (MPNNs) for chemical applications. Two different machine learning competitions [1] [2] provided an excellent starting point for experimenting with different data representations and models to predict atomic interactions (Section 2.1) or molecular properties (Section 2.2). Furthermore, they allowed to benchmark the results against word-class machine learning practitioners and see their fascinating solutions after the competitions.

Graph neural networks are one of the most interesting frontiers of deep learning for a number of reasons. As stated in the Introduction (Section 1.1), most of the breakthroughs in deep learning occurred using images or natural language (text and speech). Graph neural networks are one of the most promising methods for extending this success to other kinds of data. Another fascinating aspect of graph neural networks is that their applications encompass a lot of completely different fields from social networks, to documents and molecular structures. For this reason, the model-architectures are also required to be very diverse to fit the different use-cases.

This project investigated the challenges associated with using MPNNs for 3D data in general and molecular structures in particular. We have seen that incorporating 3D structural information into MPNNs in a way that increases the model's predictive power remains a big challenge (Section 2.2.6). Furthermore, the results of this thesis support the pure deep learning approach of only using raw data as model input (Section 2.2.5). However, the fact that manual feature engineering and raw data are still used alongside each other indicates that deep learning models for chemical applications are not yet

[1] `https://www.kaggle.com/c/champs-scalar-coupling`
[2] `https://alchemy.tencent.com`

as mature and efficient as in other fields (e.g., computer vision and natural language processing) where manual feature engineering has been made completely obsolete by end-to-end differentiable models using raw data only. Therefore, a lot of research remains to be done before MPNNs reach their full potential.

Finally, the project also showed that some of the key factors for the success of a machine learning project are not directly machine learning-related at first glance. One of them is data quality (or the lack thereof), which required a lot of attention in the project at hand (Section 3.1.2.3). Furthermore, it is vital to make experiments easily reproducible, to be able to run multiple experiments automatically and to store all parameters and results in a systematic manner. To this end, the experimentation-process was largely automated [3] and the python library *mlflow* was found to be very useful to store and access all experiment-parameters and results. These factors are even more decisive in large organizations, but provide substantial benefits in one-person projects as well.

Summing up, the project was a great learning-opportunity for designing deep neural networks and structuring machine learning projects.

[3]The code is available on github: `https://github.com/raph333/masters_project`

Bibliography

[1] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," *Proceedings of the 34th International Conference on Machine Learning*, 2017. [Online]. Available: http://arxiv.org/abs/1704.01212

[2] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "SchNet: A continuous-filter convolutional neural network for modeling quantum interactions," *31st Conference on Neural Information Processing Systems*, no. 1, pp. 1–11, 2017. [Online]. Available: http://arxiv.org/abs/1706.08566

[3] A. C. Ian Goodfellow, Yoshua Bengio, *Deep Learning*, 2016. [Online]. Available: http://www.deeplearningbook.org

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *Journal of Chemical Theory and Computation*, vol. 15, no. 6, pp. 3678–3693, jun 2017. [Online]. Available: https://pubs.acs.org/doi/10.1021/acs.jctc.9b00181http://arxiv.org/abs/1706.03762

[5] J. S. Sepp Hochreiter, "LONG SHORT-TERM MEMORY," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] A. R. Katti, C. Reisswig, C. Guder, S. Brarda, S. Bickel, J. Höhne, and J. B. Faddoul, "Chargrid: Towards understanding 2D documents," *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pp. 4459–4469, 2020.

[7] L. J. G. Charles R. Qi, Hao Su, Kaichun Mo, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652–660, 2017.

[8] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *CoRR*, pp. 1–40, 2018. [Online]. Available: http://arxiv.org/abs/1806.01261

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pp. 1–14, 2017.

[10] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph Neural Networks: A Review of Methods and Applications," *arXiv*, pp. 1–22, 2018.

[11] P. Riba, A. Dutta, L. Goldmann, A. Fornes, O. Ramos, and J. Llados, "Table detection in invoice documents by graph neural networks," *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, pp. 122–127, 2019.

[12] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *Advances in Neural Information Processing Systems*, vol. 2015-Janua, pp. 2224–2232, 2015.

[13] P. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," *Advances in Neural Information Processing Systems*, pp. 4509–4517, 2016.

[14] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of Computer-Aided Molecular Design*, vol. 30, no. 8, pp. 595–608, 2016.

[15] P. B. Jørgensen, K. W. Jacobsen, and M. N. Schmidt, "Neural Message Passing with Edge Updates for Predicting Properties of Molecules and Materials," *32nd*

Conference on Neural Information Processing Systems, no. 2005, 2018. [Online]. Available: http://arxiv.org/abs/1806.03146

[16] G. Chen, P. Chen, C.-Y. Hsieh, C.-K. Lee, B. Liao, R. Liao, W. Liu, J. Qiu, Q. Sun, J. Tang, R. Zemel, and S. Zhang, "Alchemy: A Quantum Chemistry Dataset for Benchmarking AI Models," *ICLR 2019*, pp. 1–11, 2019. [Online]. Available: http://arxiv.org/abs/1906.09427

[17] O. Vinyals, S. Bengio, and M. Kudlur, "Order Matters: Sequence to sequence for sets," *ICLR 2016 6*, pp. 1–11, 2015. [Online]. Available: http://arxiv.org/abs/1511.06391

[18] L. Ruddigkeit, R. van Deursen, L. C. Blum, and J.-L. Reymond, "Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17," *Journal of Chemical Information and Modeling*, vol. 52, no. 11, pp. 2864–2875, nov 2012. [Online]. Available: https://pubs.acs.org/doi/10.1021/ci300415d

[19] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific Data*, vol. 1, no. 1, p. 140022, 2014. [Online]. Available: https://doi.org/10.1038/sdata.2014.22

[20] US Food and Drug Administration, "The drug development process," 2018. [Online]. Available: https://www.fda.gov/patients/learn-about-drug-and-device-approvals/drug-development-process

[21] I. Wallach, M. Dzamba, and A. Heifets, "AtomNet: A Deep Convolutional Neural Network for Bioactivity Prediction in Structure-based Drug Discovery," pp. 1–11, 2015. [Online]. Available: http://arxiv.org/abs/1510.02855

[22] S. W. Jonathan Clayden, Nick Greeves, *Organic Chemistry*, 2012.

[23] J. Klicpera, J. Groß, and S. Günnemann, "Directional Message Passing for Molecular Graphs," *ICLR 2020*, no. 1, pp. 1–13, mar 2020. [Online]. Available: http://arxiv.org/abs/2003.03123

[24] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 3147–3155, 2017.

[25] H. Cho and I. S. Choi, “Three-Dimensionally Embedded Graph Convolutional Network (3DGCN) for Molecule Interpretation,” pp. 1–39, 2018. [Online]. Available: http://arxiv.org/abs/1811.09794

[26] J. Keeler, *Understanding NMR Spectroscopy*, 2010.

[27] K. H. Sun, X. Zhang, S. Ren, and Jian, “Deep Residual Learning for Image Recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7780459

[28] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2017-Janua. IEEE, jul 2017, pp. 2261–2269. [Online]. Available: http://ieeexplore.ieee.org/document/8099726/

[29] M. Simonovsky and N. Komodakis, “Dynamic edge-conditioned filters in convolutional neural networks on graphs,” *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 29–38, 2017.

[30] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[31] A. Ng, *Machine learning yearning*, 2018. [Online]. Available: https://www.deeplearning.ai/machine-learning-yearning/