Prediction of Molecular Properties with Graph Convolution-Based Neural Networks

An exploration of parameters and layer definitions within the Message Passing Neural Network framework

by

Jean Chen

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE

in

The Faculty of Science

(Physics and Mathematics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

April 2018

Abstract

The use of machine learning methods to predict molecular properties could potentially replace some computationally expensive methods while maintaining similar accuracy. We set out to create neural networks characterized in the Message Passing Neural Network formalism to predict 12 different thermochemical properties of organic molecules from the QM9 dataset by training on 10k molecules. Molecules are represented as a collection of nodes and edges with features encoded in adjacency matrices, and the neural network is comprised of multiple graph convolution layers in conjunction with a final "readout" layer that output predictions. We explore the effects of defining different readout functions, including different node features, edge representations, as well as the inclusion of explicit hydrogen nodes. We find that initial learning rate is extremely important for error convergence, and could not meaningfully evaluate effects of using each readout function nor edge representation due to difficulty in convergence. The inclusion of hydrogen nodes and all node features, including atomic number, aromaticity, hybridization, and degree in graph (number of edges connected to node), are found to decrease test errors. In comparison to other non-neural network machine learning methods, such as kernel ridge regression and random forest, used on this particular database, the mean absolute errors on test sets for our model are approximately equal to the errors of other methods. Our predictions have an error of 3-9% within DFT values on average depending on the property, but do not achieve chemical accuracy yet and do not perform as well as some existing MPNN variants.

Preface

The formulation of graph convolution layers and adjacency matrices in Chapter 3 is based on the work of Kipf and Welling [1] on Graph Convolutional Networks. I adapted the code from a single-graph-input node-classification task to a multi-graph-input float-value prediction task.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my advisor Dr. Joerg Rottler for his guidance throughout this project, and for generously providing solutions to hardware limitations I encountered. I would also like to acknowledge fellow student Alexander Kyriazis for his input and support in the machine learning aspects of the project, especially with Tensorboard.

Chapter 1

Motivation

The ability to quickly and accurately compute properties of molecules and solids, such as electronic structures, vibrational frequencies, magnetic properties, band gaps, and atomization energies, could accelerate scientific advancements in a wide variety of fields, from chemical biology to condensed-matter physics. Any task requiring knowledge of an unknown property of a system composed of atoms can benefit from lowering computation costs without loss of accuracy. Tasks could range from identifying molecular structures of unknown substances to designing drugs and materials with tailored properties.

In principle, the solution to the Schrödinger equation encapsulates all the information of a system, and therefore any property of interest can be derived from it. In practice, however, the equation is infeasible to solve exactly for many-body problems due to the correlation term that arises from electron-electron interaction. There are many existing methods to approximate the solution, including Quantum Monte Carlo [3], GW approximation [4], and density functional theory (DFT) [5, 6]. Perhaps the most widely used method is DFT, a first principles calculation of the ground state properties of a system. These computation methods offer a good balance between accuracy and computational cost, but is still limited as computation can be slow for large, complex systems.

In recent years, there has been much interest in applying machine learning methods to the problem of electronic structure calculation. At its heart, Machine Learning (ML) involves using large labelled datasets to train an algorithm how to predict some desired values using statistical machinery. It is pattern recognition: fitting a model to existing data to solve complex problems without actually going through the motions of solving a problem in a traditional approach. In the case of predicting molecular and material properties, the traditional approach would be calculations based on ground-truths, approximations of physical terms in equations.

There is an increasing abundance of freely-accessible materials and molecular data, such as the QM9 dataset (2012) [7, 8], Open Quantum Materials Database (2013) [9, 10], Materials Project (2015) [11], MoleculeNet (2017) [12], or High Throughput Experimental Materials Database (2018) [13]. With so many sources of training data at hand, applying ML methods appears to be a promising path. Indeed, many models trained with DFT results are more accurate to the DFT results than DFT is accurate to experimental data [2, 14]. In terms of computation time, there are established machine learning methods for these purposes that can compute as fast as $O(10^{-2})$ seconds while the equivalent calculation with DFT takes $O(10^{3})$ seconds[14].

Existing studies employ a variety of ML algorithms and representations of input data. For molecular systems, Gubaev et al. used an active learning algorithm with an input of atomic numbers, positions, and the neighbours of each atom to introduce local effects of interatomic

interactions [15]. Rupp et al. calculated atomization energies for organic molecules using Kernel Ridge Regression (KRR) with Coulomb matrices, which are adjacency matrices with Coulomb potential entries [16]. Hansen et al. calculated atomization energies and total energies also using KRR with a vectorized representation called "bag of bonds [17]. Numerous other papers also use KRR, with quirky names and different representations such as SOAP (smooth overlap of atomic positions) [18], BAML (bonds, angles, and machine learning) [19], HDAD (histogram of distances, angles, and dihedrals) [2], and MBTR (many-body tensor representations)[20]. There are also many artificial neural network-based methods such as DTNN (deep tensor neural network) [21] and DCNN (deep convolutional neural network) [22].

On the front of predicting properties of materials, Behler and Parrinello used a neural network with Cartesian coordinates of atoms as input [23]. Ward et al. used random forests with a large input set of system information such as fractions of elements present, average atomic number, and range of atomic radii [24]. And most recently, Xie and Grossman published a convolution-based neural network for crystal graphs [25].

A detailed performance comparison of combinations of data representations and learning algorithms by Faber et al. [2] included various of the above studies such as bag of bonds, BAML, molecular graphs, Bayesian ridge regression, RF, KRR, graph convolutions [26], gated graph neural networks [27], and more. It was found that KRR and graph-based neural networks returned the highest accuracies.

In particular, the use of neural networks (NNs) with graph representations of systems is an interesting route. NNs are often able to approximate complex nonlinear models, and graph representations are desirable as they preserve spatial invariance: for example, a molecule rotated or translated should have the same properties as its untransformed state. Convolutional neural networks (CNNs) are common methods used in image and speech recognition today. In the case of image recognition, a CNN takes in pixels of an image and extracts some meaning from the pixels, which is analogous to taking in the graph structure of a molecule or crystal to find its properties.

Gilmer et al. demonstrated that CNNs with graph representation of data outperform other representations in [14], and abstracted common features of CNNs used for finding molecular properties into a "Message Passing Neural Network" (MPNN) framework, which includes CNN for molecular fingerprints [28], Gated Graph NN [27], Interaction networks [29], Molecular Graph Convolutions [26], DTNN [21], and two original variants of MPNNs.

The scope of this thesis project is to use the established MPNN formalism to create new variations of neural networks with varying graph-based representations of molecules as inputs, and explore the effects of various parameters on model accuracy.

Chapter 2

Theory

In this section, basic ideas of machine learning and neural networks are reviewed, and the MPNN framework is explained.

2.1 Machine Learning Fundamentals

An ML model is a generalized parameterized function that learns to map a collection of input data samples to their target outputs. The objective in the learning, or 'training' stage is to optimize the parameters of the model so as to minimize the difference between the predicted outputs and target outputs for each sample. This is also known as minimizing the loss, and is usually accomplished using gradient descent techniques. The ultimate goal is to be able to train on a known dataset of input-output pairs so that the model can be used to predict outputs of samples that have never been trained on. The typical methodology in building an ML model is to partition the available labelled data set into a training set, and a testing set. The model has an omniscient awareness of the training set, but is blind to the target outputs on the testing set.

The fundamental assumptions in these models are:

1. Every sample is independent of other samples.
2. The training set distribution is representative of the real distribution.

These assumptions ensure that every sample can be predicted exclusively from its own features, and that learning from the training set can allow a model to accurately predict targets on an unseen set.

A dataset is comprised of a collection of samples, with each sample having a collection of features. For instance, the QM9 dataset[7] contains 134k small organic molecules made of C, H, O, N, F and various properties of each molecule such as atomization energies, specific heat, polarizability, etc. One sample in the dataset would be a molecule; its features are the coordinates of each atom and their various atomic properties. One can process, transform, and select useful subsets of the features to use in the training phase. For each sample, the training dataset includes the desired output targets. In the case of a molecule, the targets are the molecular properties. In theory, a sample's features should encapsulate all information needed to predict the target, but the challenge is designing a function to accomplish that.

Once the parameters of an ML model have been optimized in some way (training), one would approximate the accuracy of the model through what is known as a validation set. The model training was blind to the validation set, so it serves as an unbiased measure of the predictive

accuracy on unseen data. The reason that the validation set is a separate entity from the training set, is to eliminate the optimization bias on the testing set. For this reason, the testing set accuracy is measured as few times as possible, and serves as a true and final measure of the predictive accuracy of the model on unseen data.

There are two extremes in an ML model's performance that is critical to understand. If the model is too complex and matches the training data too exactly, it is said to be "overfitting". This is characterized by a low training loss, and a relatively high testing loss. A model can be made arbitrarily complex to predict its training set's targets very well, but this is not useful unless predictions on an unseen testing set are accurate. On the other end of the spectrum, if the model is too simplistic and does not capture the complexities of the training data well, then the training loss and the testing loss should be very similar. In this case, the model has not reached its full potential for predicting targets. There is therefore a fundamental trade-off of model complexity that must be optimized to have the best results on unseen testing data.

2.2 Artificial Neural Networks

Among the generalized, parameterizable functions that are typically used in machine learning models, artificial neural networks are one of the most popular methods for nonlinear problems. Artificial neural networks, or NNs, are cascades of layers of connected weighted nodes that can pass information through their connections. This structure resembles biological neural network in how they transmit signals. Each layer of the network can be thought of as a vector of data, and the operation between layers as matrix multiplications. Therefore a neural network is parameterized by many matrix weights.

In the particular case of a feed-forward neural network, input data is inserted into the network through the input layer. The information is propagated forward through the hidden layers via linear transformations to the output layer. This is illustrated in Figure 2.1. As it stands, this is purely a linear transformation. In practice, a non-linear 'activation function' is applied to each node at every layer in order to introduce non-linearities. Typical activation functions include the sigmoid function, the hyperbolic tangent function tanh, and the ReLU function ($y = max(0, x)$).

Several hyperparameters can be modified to increase or decrease the complexity of the neural network. Hyperparameters are characteristics of the model that are defined prior to training and set by the designer (rather than trained directly). Some example hyperparameters that a designer can modify include the neural network depth (number of layers), the neural network breadth (width of layers), and the activation functions used. The assumption is that the designed structure can take on a particular set of weights such that the neural network can accurately approximate the desired function that maps the input samples to their respective targets. The weights are "trained", which means they are adjusted at each training step to achieve lower loss.

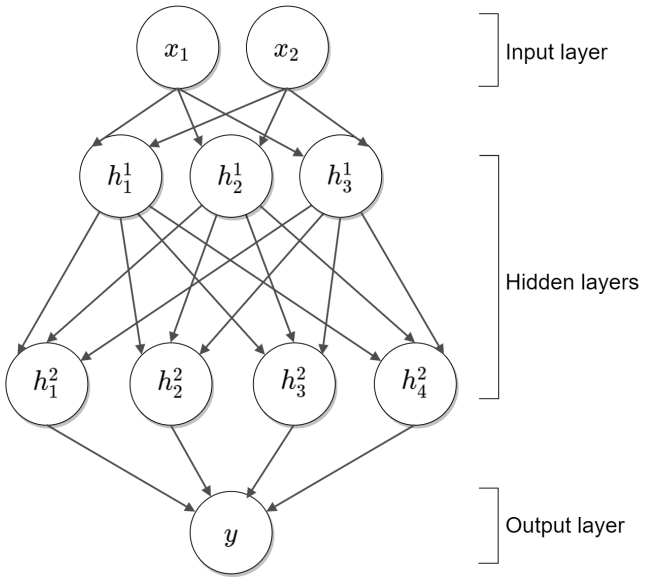

Figure 2.1: A feed-forward neural network with an input layer, an output layer, and two hidden layers. x_1, x_2 are inputs, y is the output, and h_i^l are the values of node i in layer l.

2.2.1 Input Representation

The representation of input data is one of the more important details to decide when training a neural network. The input data samples for this project are molecules, which can be represented as graphs with nodes and edges connecting the nodes. The graphs, in turn, can be represented with adjacency matrices indicating the existence of a bond between two nodes. An example with H_2O is shown in Figure 2.2, where the molecule is represented in a matrix with entries "1" representing a single bond, and entries "S" representing some connection of each atom to itself. Each node has its own set of features associated with the atom, and each edge also has its own set of features associated with the bond. Note the adjacency matrix is not limited to representations of bonds. It can encode Coulomb potentials, Euclidean distances between non-bonded atoms, and much more. Message passing neural networks (MPNN) serve as a way of parameterizing a function that can generate a single array of outputs from an input graph.

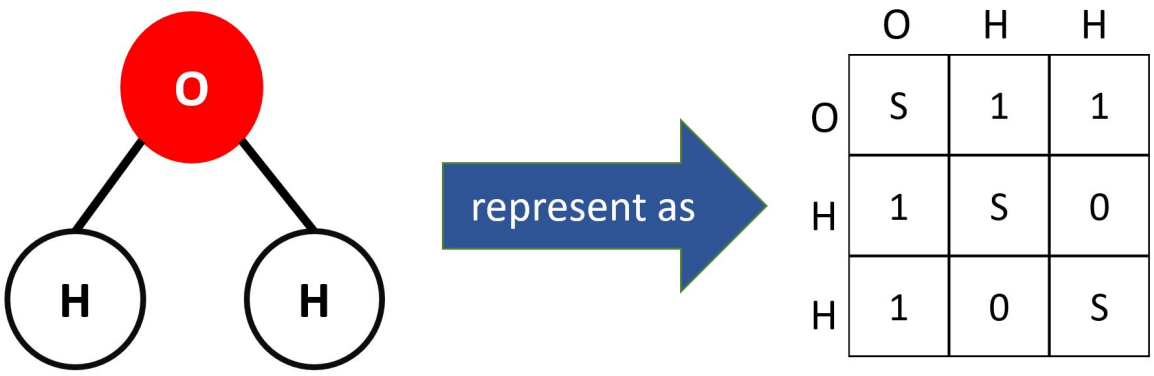

Figure 2.2: Molecular graph represented in an adjacency matrix. The "1" entries represent the single bond between oxygen and the two hydrogen atoms. The "S" entries represent the connection of each atom to itself. The zero entries indicate that no bond exist between the two hydrogen atoms.

2.2.2 Message Passing Neural Network

A message passing neural network (MPNN) is a generalized framework for neural networks whose inputs are graphs. A brief overview is provided here, but details can be found in [14]. In an MPNN, the collective layers of the neural network are organized into a message-passing phase, and a readout phase. In the message-passing phase, nodes are updated at each timestep t using information from neighbouring nodes in the graph, hence the name "message-passing". Specifically, for an input graph G with nodes v, node features x_v and edge features e_{vw}, the state of the v^{th} node at time t is h_v^t. The node states after each message-pass is determined by

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{2.1}$$

where U_t are the vertex update functions, and m_v^{t+1} is determined by

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{2.2}$$

which generates a message to pass from node v to node w. $N(v)$ are the neighbours of v in G, and M_t are the message functions. After applying T message-passing iterations to the original input G, the output $\hat{y}$ is computed in the readout phase as

$$\hat{y} = R\left(\{h_v^T | v \in G\}\right) \tag{2.3}$$

where R is the readout function that looks at the state of every node after the message passing. This outputs the final values that will be directly compared against the targets.

When applied to molecular data, the nodes v are constituent atoms of a molecule, the edges e are the bonds between atoms, node/edge features x_v and e_{vw} are information about atoms/bonds, and the readout $\hat{y}$ is one or more properties of the molecule.

The message functions M_t, the update functions U_t, and the readout function R are parametrized in such a way that standard gradient descent techniques can be used to optimize their learnable weights. The choice of these functions creates a unique variant of an MPNN.

2.2.3 Convolution Layers

The neural networks in this project are characterized with message-passing formalism, but the functions M_t, U_t, and R are fully dependent on the structure and function of hidden layers. The choice of these hidden layers is therefore one of the most important pieces to solving this puzzle. An ubiquitous network is the 2D Convolutional Neural Network (CNN), which is a type of feed forward network formally introduced by [30]. It has proved itself to be extremely effective in image classification domains, and is precisely the technology used in automated image labelling systems.[1] One of reasons for its success is the inherent encoding of spatial information from the data due to the structure of the network. A variant of the CNN that works on graph-based inputs instead of images is a natural choice for the molecule problem, as the spatial configuration of a molecule is a crucial part of its identity.

The defining characteristic of CNNs is the use of specific hidden layer types, namely convolution layers, pooling layers, and fully-connected (dense) layers. For our purposes, only the convolution layer is important.[2] A convolution layer is illustrated in Figure 2.3. In this layer, the input data can be thought of as a matrix M. This input matrix is "convolved" with a $k \times k$ weight window matrix w, where k is a hyperparameter. As per the definition of convolution, each $k \times k$ area (called receptive fields) of M is multiplied element-wise with w and summed to become one entry in the input array L of the next layer. The resulting output of this process has the same dimensions as M, but every element in the new matrix becomes a linear combination of its neighbouring elements in the previous layer. The values of the weight matrix is learned through training. Often, a nonlinear activation function is applied after convolution as similarly described in Section 2.2.

This can be repeated multiple times to obtain higher-level levels of abstraction that can allow one to convert data from very large 2D matrices (e.g. images) to simple higher level representations (e.g. classification labels).

[1] If you have a Google account and use its photos storage function, Google Photos, it will automatically create categories for your images such as "Cat", "Boat", and "Archery". This is most certainly done with CNNs.

[2] As the size of our inputs is relatively small (molecules of sizes < 100 atoms) compared to the large images with hundreds of thousands of pixels a CNN typically encounters, a pooling layer would cause too much loss of data. Instead of pooling layers, small random dropouts can be used to reduce overfitting if necessary. A fully-connected layer in our neural network is something to explore in the future, as discussed in Chapter 5.

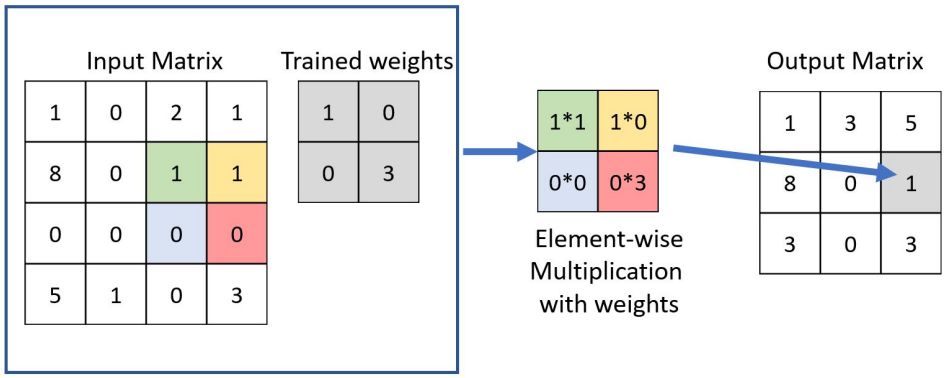

Figure 2.3: An example of a convolution operation. The gray window can be thought as a window matrix. It slides over every position in the input matrix, and is multiplied element-wise with it. The sum of all products of the element-wise multiplication becomes the value in the output matrix at that particular window position. Window matrix weights can be trained to identify specific features in the input matrix. This spatial processing is critical for modern image classification systems.

Chapter 3

Method

3.1 Computation Tools

All code in this project is written in a Python 2 environment and there are ongoing efforts to make it fully compatible with Python 3. We use TensorFlow[31], a machine learning library for Python.

3.2 Neural Network Architecture

The neural network is structured to be a Message Passing Neural Network (see Section 2.2.2). Thus data in the input layer propagates forward to hidden layers where a message-passing phase and a subsequent readout phase are defined. Output from the readout phase is the output of the neural network. The overall network is shown in Figure 3.1.

It is tricky to choose or optimize the structure of the network, as there are many hyperparameters to tune that occasionally influence each other. To experiment with each hyperparameter, a "default" setting of all parameters for a network is used, and a single hyperparameter is varied for a few trials. The default setting and trial ranges for the tested parameters are discussed below, and are listed in Table 3.1. The default settings for all other parameters are listed in Table 3.2.

3.2.1 Network Size

The number of hidden layers, as well as the size of each hidden layer are hyperparameters pertaining to the size of the neural network. We do not perform a systematic search of the optimal "number of nodes in each layer" in combination with "number of layers" to avoid optimization bias.[3] In addition, the physical layout of the network is not the focus of the project. Instead, the model accuracy as the number of layers and layer size increased is roughly benchmarked and a reasonable configuration is chosen as the default setting for all subsequent training.

Number of Layers (Depth)

We train 4 separate models with neural networks of similar widths, but different depths. The depths we choose are 3, 5, 7, and 9 graph convolution layers with one readout layer appended

[3]Optimization bias occurs when one finds a particularly favourable model by chance when testing out many models using the same dataset. The more models one tests out, the higher the likelihood that one of the models has the highest accuracy purely by chance, and is not actually the best model for the purpose.

Table 3.1: Parameters of the neural network that are tested. Separate models are trained for each experiment, which tests only one parameter. If a particular parameter is not being tested in a model, then the value of the parameter is set to the default setting.

Parameter	Default setting	Tested range
Readout function	Single-layer w. Activation	6 functions (See Sec. 3.2.3)
Training set size	10,000	1000-11000
Target	C_v	12 properties (See Table 3.3)
Batch normalization	False	True/False for using batch normalization in convolution layers
Hydrogen inclusion	False	True/false for including H as explicit input nodes
Features	Full feature set	Various combinations of features selected from the full set
Adjacency matrix	Bond order elements	Bond order, geodesic distance, aromaticity

Table 3.2: Parameters of the neural network kept constant. Note the number of graph convolution layers and the number of nodes in each convolution layer are selected after rough benchmarking.

Parameter	Default setting	Details
Random seed	12	Used for initialization of weight matrices. Included for reproducibility
Initial learning rate	N/A	Chosen carefully based on the training loss in the first few training steps for each model
Number of graph convolution layers	7 layers	This is the depth of the neural network
Number of nodes in each convolution layer	(f,100,80,80, 80,90,90)	(layer 1, layer 2, ...), where f = number of features per atom

after the convolution layers for every model. The specific configurations (width of layer 1, width of layer 2, ..., width of layer n) of the neural networks for input data with f features are:

3 convolution layers: (f, 100, 80)
5 convolution layers: (f, 100, 80, 80, 90)
7 convolution layers: (f, 100, 80, 80, 80, 90, 90)
9 convolution layers: (f, 100, 80, 80, 80, 80, 80, 90, 90)

Number of Nodes (Width)

For a neural network depth of 7 graph convolution layers + 1 readout layer, we train 4 separate models with different layer widths.
The configurations of the neural networks for input data with f features are:
model a: (f, 50, 30, 30, 40, 40, 40)
model b: (f, 80, 60, 60, 60, 70, 70)
model c: (f, 100, 80, 80, 80, 90, 90)
model d: (f, 130, 110, 110, 110, 120, 120)
where we've named the models a, b, c, d to reference them more easily.

3.2.2 Initialization

All learned weight matrices are initialized with the glorot scheme by Glorot and Bengio[32], which, in theory, leads to faster convergence.

3.2.3 Hidden Layers

We experimented with networks consisting of several graph convolution layers in the message-passing phase and end with a readout layer. The size and number of these layers are fixed hyperparameters. The overall architecture is visualized in Figure 3.1.

I. Message-Passing Phase: Graph Convolution Layer

In the message-passing phase, only graph convolution layers are used. Recall each node h_v^t in the graph is updated with update and message functions

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

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

In a graph convolution layer, each node becomes a linear combination of itself and its neighbours as it propagates to the next layer. In the MPNN formalism, the t^{th} graph convolution layer is equivalent to the message passing function

$$M_t(h_v^t, h_w^t) = c_{vw} W_{vw} h_w^t \tag{3.1}$$

and the vertex update function

$$U_t(h_v^t, m_v^{t+1}) = Act(W^t m_v^{t+1}) \tag{3.2}$$

where c_{vw} is a scalar dependent on the input representation chosen, W_{vw} and W^t are learned weight matrices, and $Act(\cdots)$ is an activation function of either leaky ReLu, sigmoid, or tanh (hyperbolic tan). The choice of the activation function in this phase is somewhat arbitrary, as switching between different functions produce no significant difference in the models. Leaky ReLu[33] is used in place of the classic ReLu function as the latter encountered issues with "dead" neurons stuck in a zero-gradient and cannot learn. As an added benefit, leaky ReLu may perform better on convolutional neural networks [34].

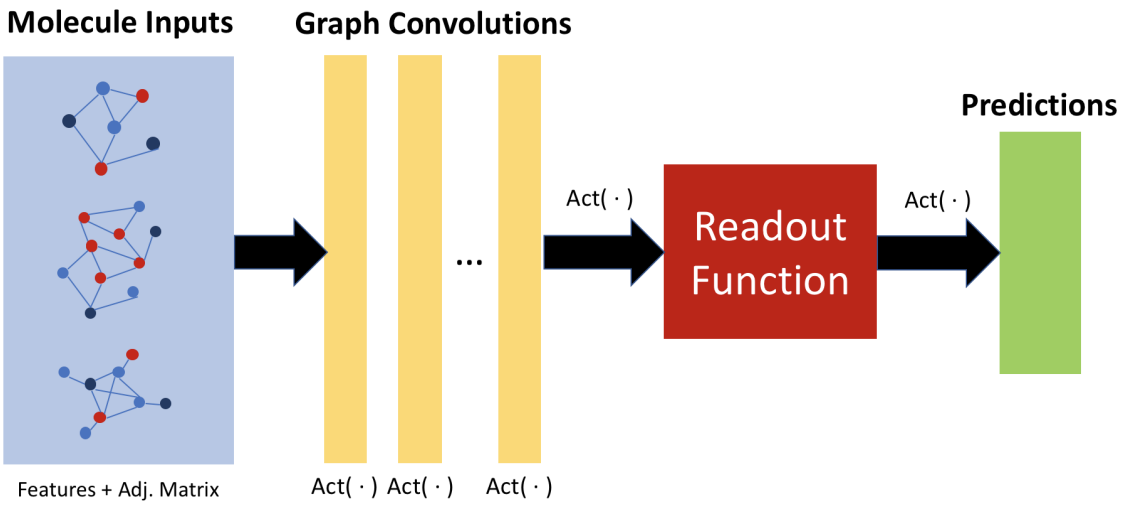

Figure 3.1: A representation of the end-to-end structure model structure used in this study. The input molecules each have an associated adjacency matrix and a per-atom feature list. The graphs are convolved several times so the atoms can pass information to each other. The readout function produces a single array of outputs for each convolved molecule. $Act(\cdot)$ represents a non-linear activation function applied to a layer output. In general, different activation functions are used throughout the model. This model is parameterized by the weights making up the graph convolution and readout layers, and can be optimized using gradient descent to minimize training loss between the predicted outputs and target outputs.

II. Readout Phase

Recall the readout layer is a function R defined in Section 2.2.2 of the form

$$R\left(\{h_v^T | v \in G\}\right) = \hat{y} \tag{3.3}$$

where the inputs h_v^T are node states after T layers of graph convolution. Several readout layers are defined to create models of varying complexities.

II.i. Single-layer Readout

The simplest readout layer after T convolution layers is defined as

$$R\left(\{h_v^T | v \in G\}\right) = \sum_{v \in G} i(h_v^T) \equiv \sum_{v \in G} W_v h_v^T \tag{3.4}$$

where i is a single-layer neural network, which is equivalent to multiplying the h_v^T vector with a learned weight matrix W. Or equivalently, this is a simple weighted sum of all nodes pertaining to a molecule outputted from the message-passing phase.

II.ii. Single-layer Readout With Activation

A small upgrade of the single-layer readout is to add an activation function in the neural network:

$$R\left(\{h_v^T | v \in G\}\right) = \sum_{v \in G} Act(i(h_v^T)) \equiv \sum_{v \in G} Act\left(W h_v^T\right) \tag{3.5}$$

where $Act(\cdots)$ is tanh. Note ReLu and sigmoid are not suitable activation functions as they force all values to be positive, and we want to predict values that are also negative. We could normalize the data differently so that all targets are positive, but the choice of activation function was roughly tested and found to make no noticeable difference.

II.iii. Multilayer Readout with Activation

The multilayer readout function is almost identical to the single-layer readout, with the exception that the neural network $i(\cdots)$ is a 3-layer neural network with weight matrices M_k, biases B_k and tanh activations Act_k, as below:

$$R\left(\{h_v^T | v \in G\}\right) = \sum_{v \in G} Act(i(h_v^T)) = \sum_{v \in G} Act_3\left(M_3 Act_2\left(M_2 Act_1(M_1(h_v^T) + B_1) + B_2\right) + B_3\right) \tag{3.6}$$

II.iv/v. Single-/Multilayer Readout with Activation and Initial Features

The inclusion of initial inputs h_v^0 in the readout from a previous study[26] is interesting enough to warrant some exploration. Thus we define

$$R\left(\{h_v^T | v \in G\}\right) = \sum_{v \in G} \sigma\left(i(h_v^T)\right) \odot Act\left(j(h_v^0)\right) \tag{3.7}$$

where σ is the sigmoid function, $\odot$ is element-wise multiplication, and $Act(\cdots)$ is tanh. The sigmoid function can be used here as tanh allows for negative values.

II.vi. Multilayer Readout with Activation and Initial Features II

As an extension, this readout function is a slightly more complex version of the previous readout:

$$R\left(\{h_v^T | v \in G\}\right) = \sum_{v \in G} \sigma\left(k\left(i(h_v^T t) \odot h_v^0\right)\right) \odot Act\left(j(h_v^0)\right) \tag{3.8}$$

where $\odot$ is element-wise multiplication, $Act(\cdots)$ is tanh, i, j, k are single-layered neural networks, and h_v^0 are initial features.

3.2.4 Batch Normalization

Batch normalization[35] is a common technique in neural network architectures said to lead to higher accuracy and faster convergence. Rather than normalizing the input data distributions so they have 0-mean and unit variance once at the beginning, batch normalization is done after every graph convolution layer. This corrects data that has its distribution shift as it propagates through the network due to the matrix multiplications and activation functions. A neural network with default settings is trained along with an identical neural network with batch normalization to deduce its impact on convergence time and identify potential effects on model accuracy. The batch normalization method is provided by TensorFlow under `tf.nn.batch_normalization()`.

3.3 Training Data

The training data used in this project is the QM9 dataset [7], which contains 133,885 organic molecules composed of elements C, H, O, N, F. It provides Cartesian coordinates of each constituent atom and their Mulliken partial charges, as well as thermochemical properties listed in Table 3.3. Density functional theory (DFT) is used to relax each molecule to a minimum-energy geometry, and to compute each property.

The dataset is processed to remove syntactical irregularities that cannot be parsed by Python, namely all instances of $x * 10^y$ written as "$x * \wedge y$" are replaced with "$x * 10 \wedge y$". Node and edge features for each molecule are generated using Python library RDKit [36]. The full set of features can be found in Table 3.4.

There are two SciPy sparse matrices in which input data are stored. The first is an $n \times f$ node features matrix, for n samples (i.e. n molecules) and f features. The second structure is the $n \times n$ edge features matrix called the adjacency matrix. The adjacency matrix represents the edges in the molecule graph. For instance, for a training set of 2 molecules C_2H_2 and CO, an adjacency matrix can be

$$\begin{bmatrix} & C & C & H & H & C & O \\ C & S & 2 & 1 & 0 & 0 & 0 \\ C & 2 & S & 0 & 1 & 0 & 0 \\ H & 1 & 0 & S & 0 & 0 & 0 \\ H & 0 & 1 & 0 & S & 0 & 0 \\ C & 0 & 0 & 0 & 0 & S & 2 \\ O & 0 & 0 & 0 & 0 & 2 & S \end{bmatrix}$$

where the value of "S" is some large number indicating the correlation between an atom and itself. In practice, setting S=0 did not affect model accuracy for our particular flavour of neural network. The edges do not necessarily have to be bonds while the nodes are always atoms in our models.

3.3.1 Normalization

Target Normalization

The distribution of each molecular property from the QM9 dataset is shown in Figure 3.2. The distributions look fairly Gaussian or Lorentzian, thus we can standardize them to a normal distribution with a mean of 0 and variance of 1. For practicality reasons, we approximate the Lorentzian peaks as very narrow Gaussians. This is effectively suppressing the values within one standard deviation of the mean to a range of [-1,1]. This accounts for approximately 68% of the data. As our activation functions from the readout suppresses values to a range of [-1,1], this means the model is limited to only be able to predict 68% of the data. Thus we divide by 2 to include two standard deviations, which accounts for 95% of the data.

The formula used for standardized targets X_i and original targets y_i of the i^{th} molecule is thus

$$X_i = \frac{y_i - \mu}{2\sigma} \tag{3.9}$$

where the mean μ and the standard deviation σ are

$$\mu = \sum_{i=1}^{m} \frac{y_i}{m}$$

$$\sigma = \sqrt{\sum_{i=1}^{m} \frac{(y_i - \mu)^2}{m}}$$

All targets and errors pertaining to target v.s. prediction are for normalized values unless otherwise specified.

Feature Normalization

All features except boolean values and atomic numbers are normalized in the same way targets are normalized to restrict values to the range[-1,1] so that different features are comparable in magnitude. Atomic numbers are normalized to be positive numbers smaller than 1. This is done with a division of all atomic numbers by 9, as the largest atom is Fluorine.

3.3.2 Experiments

To experiment with various moving parts in the training data, a "default" setting for training is defined as (1) hydrogen atoms are excluded from the molecular graph inputs to reduce memory usage, (2) the full set of available features is used for training, (3) training set size is 10,000 randomly chosen molecules, and (4) one adjacency matrix with bond order information is used. Tables 3.1, 3.2 show the default settings for all important parameters.

The ratios of training:validation:test set sizes are set to 3:1:1. The validation set can be used to compare to test set accuracy as a sanity-check, as the two sets should have similar errors that are higher than training error.

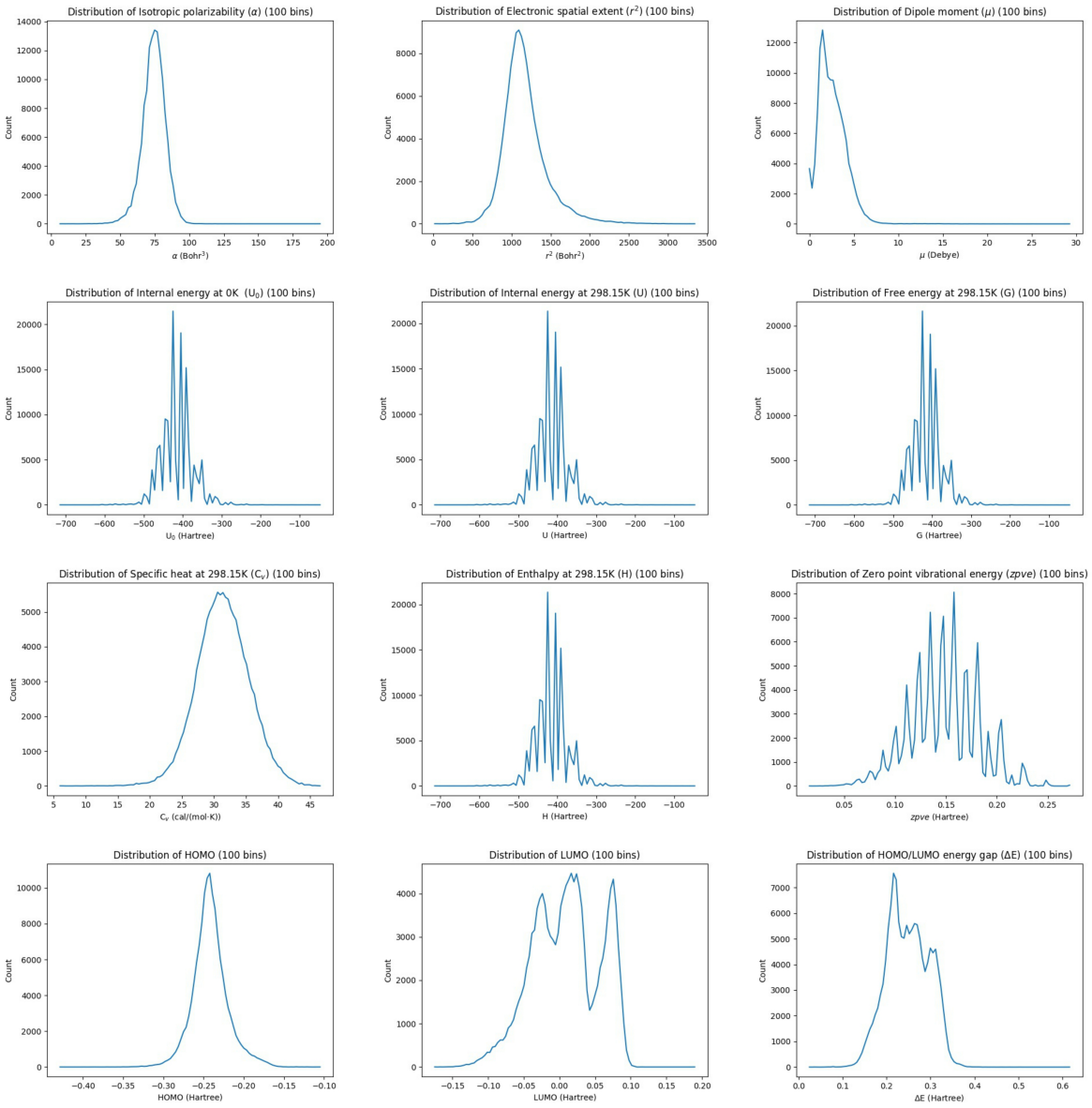

Figure 3.2: Distributions of values for all thermochemical properties provided in the QM9 dataset. Most distributions are fairly Gaussian and thus warrant normalization to a standard distribution of mean=0 and variance=1.

Hydrogen Inclusion

Hydrogen atoms on organic molecules are often left as implicit nodes as they are usually fillers to complete octets in the valence shells. Omitting hydrogen from our molecular graphs for the input data is done in many previous studies and certainly reduces memory consumption and runtime. However, Gilmer et al.[14] found that including hydrogen in the molecules for the input data was important for their MPNN, as it improved the predictions of several targets. Thus we also compare models with and without explicit hydrogen nodes.

Training Set Size

Models are trained on 1000, 3000, 5000, 7000, 9000, 10000, and 11000 molecules. Each model uses a set of molecules that are randomly drawn from the 130k molecule dataset. Marginal improvements at different molecule counts are explored in order to better understand the data efficiency of the system. Using more than 11k molecules per model was found to be impractical with the computational resources used in this study.

Features Matrix

To assess the effects on model accuracy of including each node feature, we start with a model trained on only the atomic numbers. For each subsequent model, we add another feature to the set of inputs. All features are listed in 3.4.

Adjacency Matrix

We experiment with including more than one adjacency matrix, or equivalently, a three-dimensional adjacency matrix, for which entry ijk is the k^{th} edge feature of the bond between atoms i, j. The edge features are bond order, geodesic distance, and bond aromaticity, as listed in Table 3.4. Geodesic distance is the number of edges in the shortest path between two atoms. This particular feature is computed for between all atoms, regardless of whether they are neighbours.[4] Lastly, bond aromaticity is simply a binary value indicating whether the bond is aromatic.

3.4 Training Algorithm

At least 200 epochs (training steps) are performed for all models. If at 200 epochs, the error is still actively decreasing with each step, indicating that the model has not converged, an additional 1000 steps with smaller learning rates are performed, with early stopping possible and expected. Early stopping occurs when the algorithm detects that training error will no longer decrease, or while training error is decreasing, the validation error increases steadily– which indicates potential overfitting.

We optimize the mean squared error on the training data. As the data is encoded in large sparse matrices, we use the Adam optimizer [37] as our gradient descent optimizer algorithm,

[4]There is no issue with unreachable nodes, as there are no atoms without any bonds in an organic molecule.

Table 3.3: Thermochemical properties of organic molecules provided by the QM9 dataset kept in their original units. 298.15K is room temperature.

Property	Unit	Description
μ	Debye	Dipole moment
α	Bohr^3	Isotropic polarizability
r^2	Bohr^2	Electronic spatial extent
zpve	Hartree	Zero point vibrational energy
U_0	Hartree	Internal energy at 0K
U	Hartree	Internal energy at 298.15K
H	Hartree	Enthalpy at 298.15K
G	Hartree	Free energy at 298.15K
C_v	cal/(mol K)	Heat capacity at 298.15K
HOMO	Hartree	Highest occupied molecular orbital
LUMO	Hartree	Lowest occupied molecular orbital
ΔE	Hartree	Energy gap between HOMO and LUMO

Table 3.4: The full set of features used as inputs for the feature matrix (node features) and the adjacency matrix (edge featues). Values in this table are not normalized.

Node Feature	Data type	Description
Atomic number	Integer	The atomic numbers of [H,C,N,O,F] are [1,6,7,8,9]
Atom type	Boolean	True if atom is (atom type); 5 fields in total
Aromaticity	Boolean	True if an atom in an aromatic ring
Hybridization	Integer	1: s, 2: sp, 3: sp2, 4: sp4, 5: sp3d, 6: sp3d2, 7: other hybridization
Degree	Double	The number of atoms linked to an atom
Valence	Double	The number of valence electrons for an atom
Partial charges	Double	Gasteiger partial charges of atom
Edge feature		
Bond order	Integer	Single bond=1, double bond = 2, aromatic = 1.5, ...
Aromaticity	Boolean	Is bond in an aromatic system
Geodesic distance	Integer	The number of nodes one has to traverse on a graph to reach node i from node j in the shortest path, and is computed for all i,j in the graph, regardless of whether they are neighbours.

since it works well with sparse matrices and automatically adjusts learning rate. We also try other similar optimizers, such as Adagrad [38] and AdaDelta [39]. As using different optimizers produce no obvious difference in the learning curve, we settle on Adam as a somewhat arbitrary choice. All optimizers mentioned are provided by TensorFlow under the `tf.train.Optimizer` class.

3.5 Model Evaluation

The loss function for which the model is optimized is the mean squared error

$$MSE = \sum_{i=1}^{m} \frac{|X_i - \hat{X}_i|^2}{m}$$

for m molecules in the training data, normalized targets X_i and predictions $\hat{X}_i$.

To compare our results to literature values, we also compute mean absolute error for all test data:

$$MAE = \sum_{i=1}^{m} \frac{|X_i - \hat{X}_i|}{m}$$

For a more intuitive evaluation, a relative accuracy A is also computed. We define relative accuracy as

$$R = \sum_{i=0}^{m} \frac{|y_i - \hat{y}_i|}{y_i}$$

and an absolute difference

$$A = \sum_{i=0}^{m} |y_i - \hat{y}_i|$$

where y_i are unnormalized labels, $\hat{y}_i$ are unnormalized predictions. Unnormalization of targets and labels is a simple multiplication by 2σ, then an addition by μ to reverse the normalization steps. σ and μ are respectively the standard deviation and mean of training data.

Chapter 4

Results

4.1 Benchmark of Network Size

As discussed in the methods, we chose a default network setting of 7 graph convolution layers with number of nodes (f, 100, 80, 80, 90, 90, 20) as a result of some rough benchmarking discussed below to quantify the convergence time and model accuracy. Convergence of a model, or errors, means the training error has reached some minimum zero-slope point (discounting random fluctuations).

4.1.1 Number of Layers

Figures 4.1 and 4.2 show the training loss of 4 different models trained with 3, 5, 7, and 9 graph convolution layers as a function of epochs and time. In general, convergence is slightly slower for models with more layers, but the loss values the models converge to are also lower.

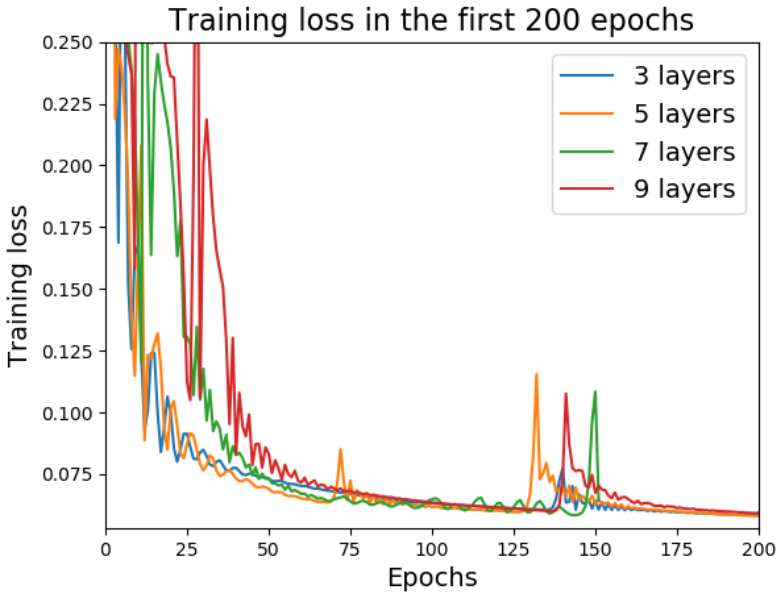

Figure 4.1: Training loss (mean square error) in the first 200 epochs for 4 different models trained on neural networks with 3, 5, 7, or 9 graph convolution layers. In general, convergence is slightly slower for models with more layers. Although it appears that all 4 models converge to the same training loss, the scale is simply too large to show the finer divide.

The magnitude of training error is unimportant in this context. Rather, we look at the test error. Mean absolute errors (MAEs) of the model on test data is plotted in Figure 4.3. As expected, networks with more graph convolution layers have lower errors. The mean absolute error appears to be reaching a minimum at some point close to, but beyond 9 layers. The difference in mean absolute error between 7 layers and 9 layers is 0.00423435. Although the mean absolute error is lower for a network with 9 convolution layers, given the slow convergence speed seen in Figure 4.1, we decide that a 0.004 difference in MAE is not significant enough to expend extra computation costs. Thus we choose to train all subsequent models with 7 layers of graph convolutions.

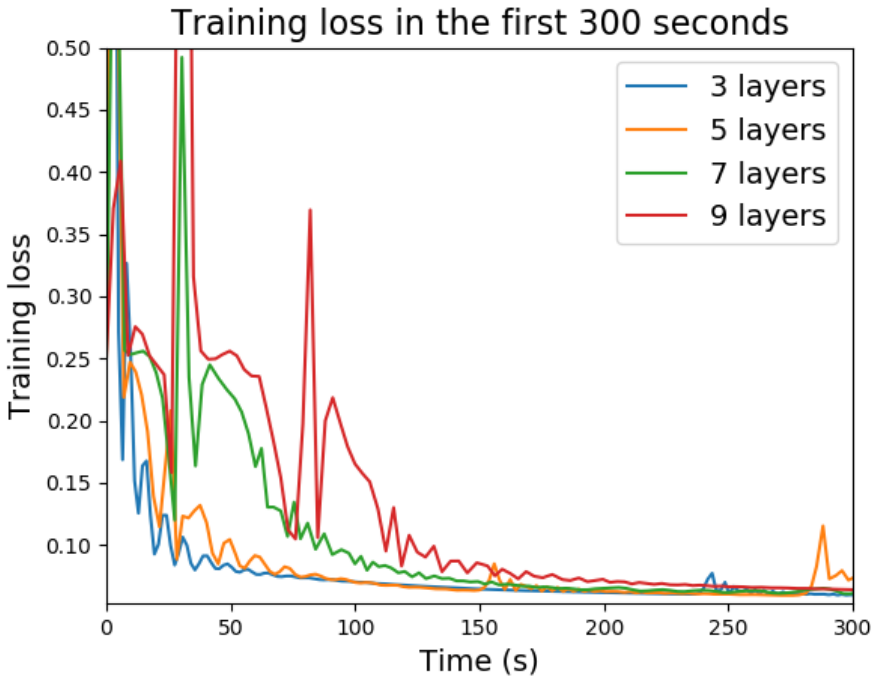

Figure 4.2: Training loss (mean square error) in the first 300 seconds for 4 different models trained on neural networks with 3, 5, 7, or 9 graph convolution layers. Models trained on more layers take longer to converge.

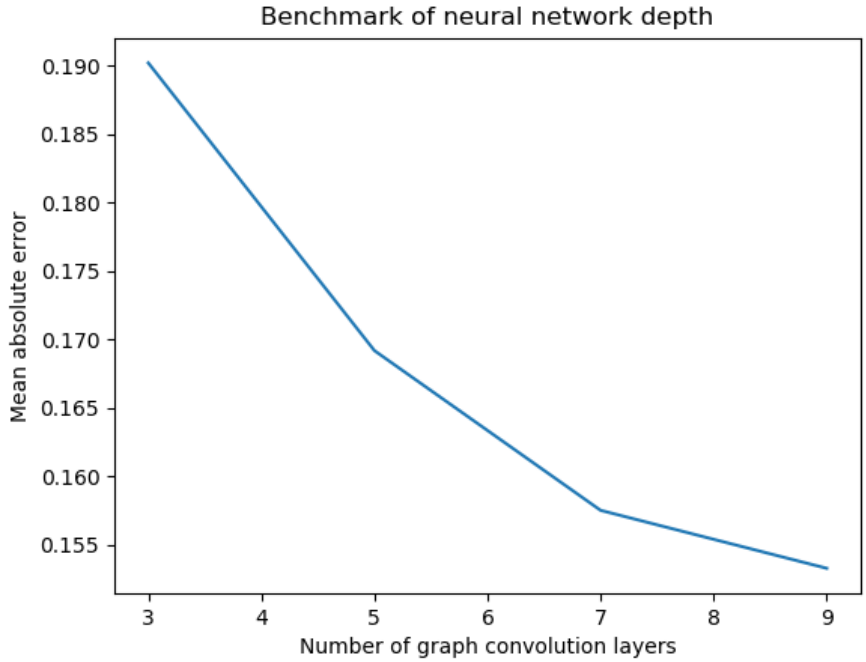

Figure 4.3: Test errors (mean absolute error) of models as the neural network depth increases. A network with more graph convolution layers is more complex and more likely to be able to predict with higher accuracy.

4.1.2 Number of Nodes in Each Layer

Figure 4.4 shows the training loss for the four models (a, b, c, d) defined in Section 3.2.1. The values to which the training losses converge follow the general trend of decreasing as the number of nodes increases. Model a, which has the least number of nodes, converged to the highest training loss as expected. Model b and c also follow the same trend. Model d appears to have higher training loss due to its slow convergence rate, but is converging to a lower loss in the long run. Figure 4.5 gives a closer look at the training loss of models b, c, d in terms of both epochs and time elapsed. It also provides a chance to see how number of epochs relate to the actual training time elapsed since the plots are just horizontally-stretched versions of each other. Comparing, for example, peaks of models b and c between the time plot and the epoch plot show that model b is faster to train as it has higher frequency oscillations in the time plot–more epochs trained in a smaller amount of time.

To determine model accuracy, we look at the mean absolute error plotted in Figure 4.6. Each model is characterized by the width of its widest layer, and its mean absolute error is shown in the figure. Again, if we disregard model d then there is a downwards trend for the test error as number of nodes increased. Model d, with 130 nodes in its widest layer, performed worse compared to simpler models due to the fact that it became harder to optimize a more complex model. It is very likely that the model could not converge properly in the time frame

allotted.

Considering convergence time and model accuracy, we chose the widths of model c to be the default setting.

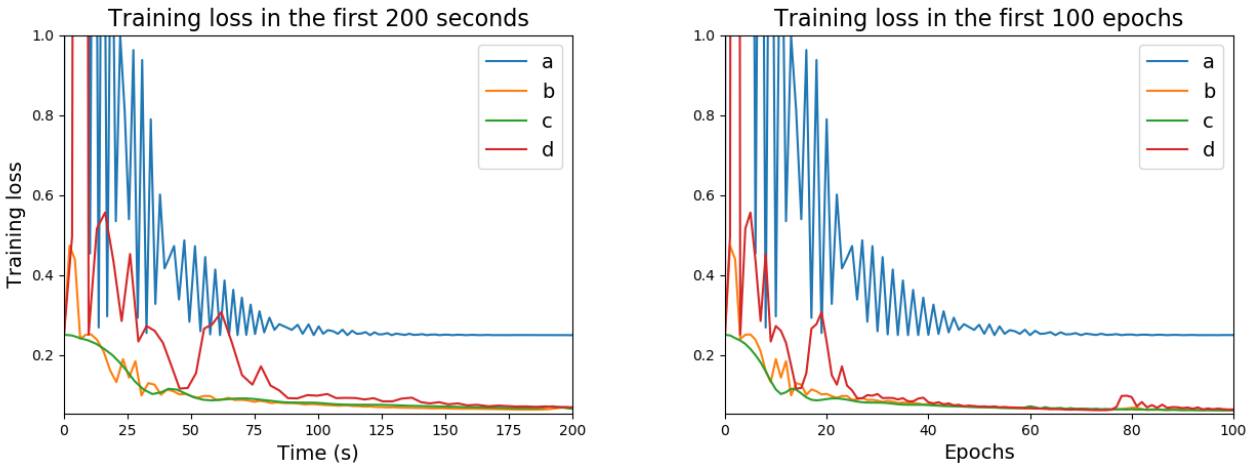

Figure 4.4: Training loss for models a, b, c, d with the same depth and different graph convolution layer widths. For input data with f features, a=(f, 50, 30, 30, 40, 40, 40), b=(f, 80, 60, 60, 60, 70, 70), c=(f, 100, 80, 80, 80, 90, 90), d=(f, 130, 110, 110, 110, 120, 120). The training losses follow the general trend of converging to lower numbers as number of nodes increases, except for model d, which has the highest number of nodes. This is because d converges slowly and thus appears to have higher training loss.

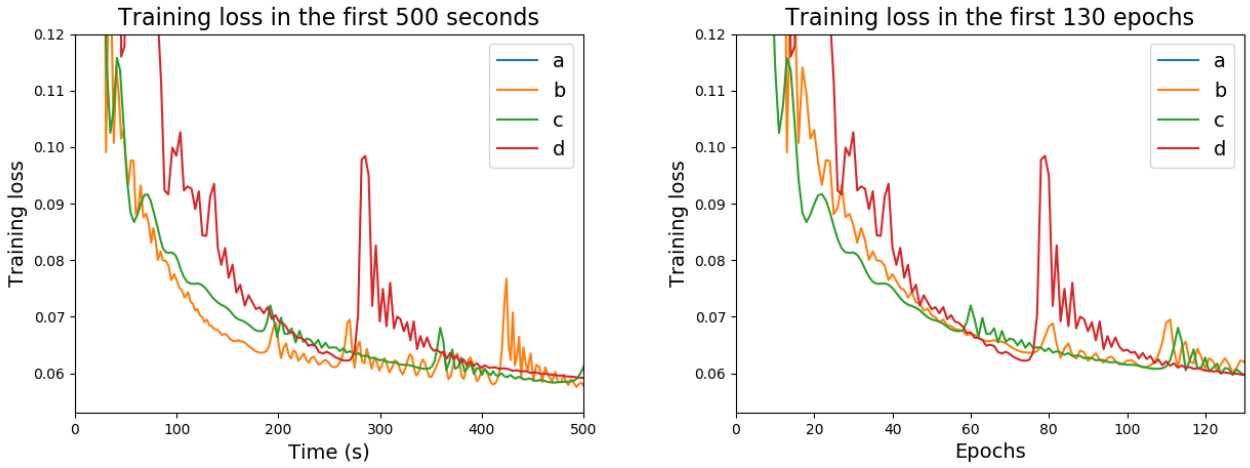

Figure 4.5: Training loss for models a, b, c, d as in Figure 4.4 zoomed in to show more detail.

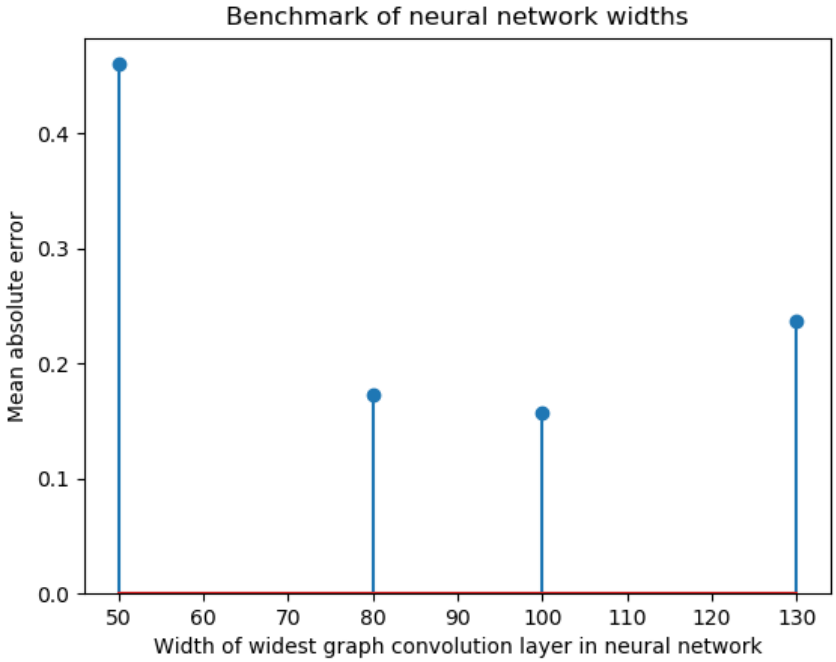

Figure 4.6: Test error (mean absolute error) for models a, b, c, d plotted from left to right. Model a has a maximum width of 50 nodes, model b has max. width 80, model c has max. width 100, and model d has max. width 130. Test error decreases as the widths increase as a general trend. Model d does not conform to this trend as it may not have fully converged after all allotted training steps.

4.2 Model Test Errors

4.2.1 Training set size

The test errors of models trained with data of sizes 1000, 3000, 5000, 7000, 9000, 10000, and 11000 molecules are plotted in Figure 4.7. The slope appears to slowly become zero as training set size increases, indicating diminishing returns of increased training data as the training set size increases.

4.2.2 Readout

Test errors of models trained on the six different readout functions are listed in Table 4.1. Increasing the number of layers from single-layer to multilayer within the readout function appears to have little effect on model accuracy. Including initial features also did not improve model accuracy.

A general problem with the readout test errors is that most of the models may not have been trained to full convergence before being tested, with the exception of the Single-layer Readout

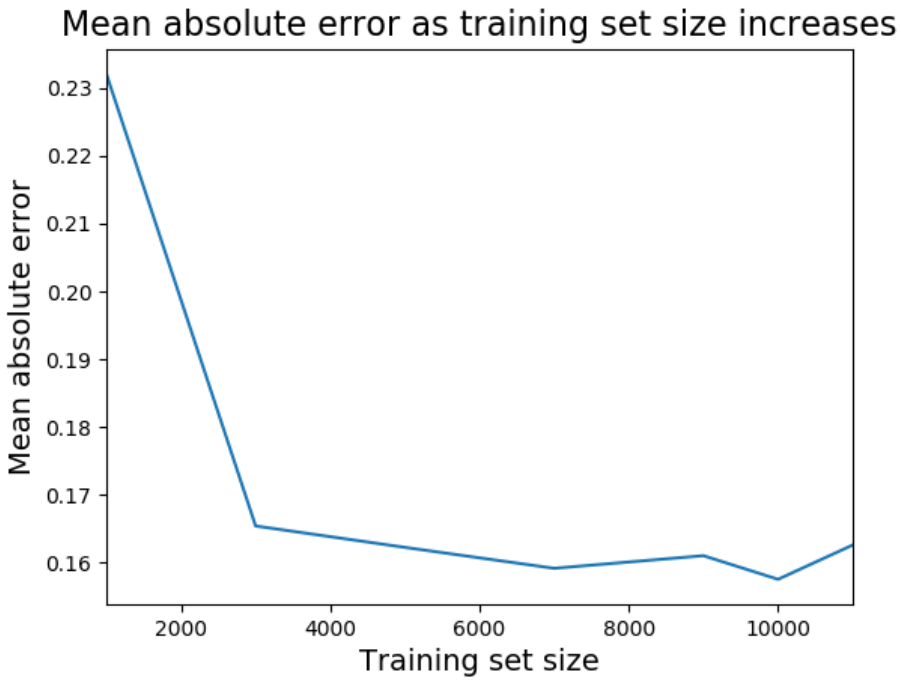

Figure 4.7: Test errors (mean absolute error) of models trained with data of sizes 1000, 3000, 5000, 7000, 9000, 10000, and 11000 molecules. The slope appears to slowly become zero as training set size increases.

with Activation. Further tests need to be carried out to fully determine the effectiveness of each readout function.

Readout function	Mean absolute error
Single-layer	0.1607217
Single-layer with Activation	0.15753435
Multilayer with Activation	0.20219968
Single-layer with Initial features	0.1678207
Multilayer with Initial features	0.16329241
Multilayer with Initial features II	0.16415943

Table 4.1: Test errors of models trained with different readout functions. See Section 3.2.3 for the full definition of each readout.

4.2.3 Batch Normalization

Including batch normalization in each graph convolution layer did not appear to have any effect on model accuracy. Batch normalization does not fulfill its original purpose of shortening the convergence time, as Figure 4.8 shows. The convergence times with and without batch

normalization are similar, and batch normalization appears to even slow down the training phase.

The MAE of the batch normalization model is 0.2057254, whereas the model without has an MAE of 0.15753435 for the same test set. The error appears higher for the model trained with batch normalization as it was not trained until the error converged.

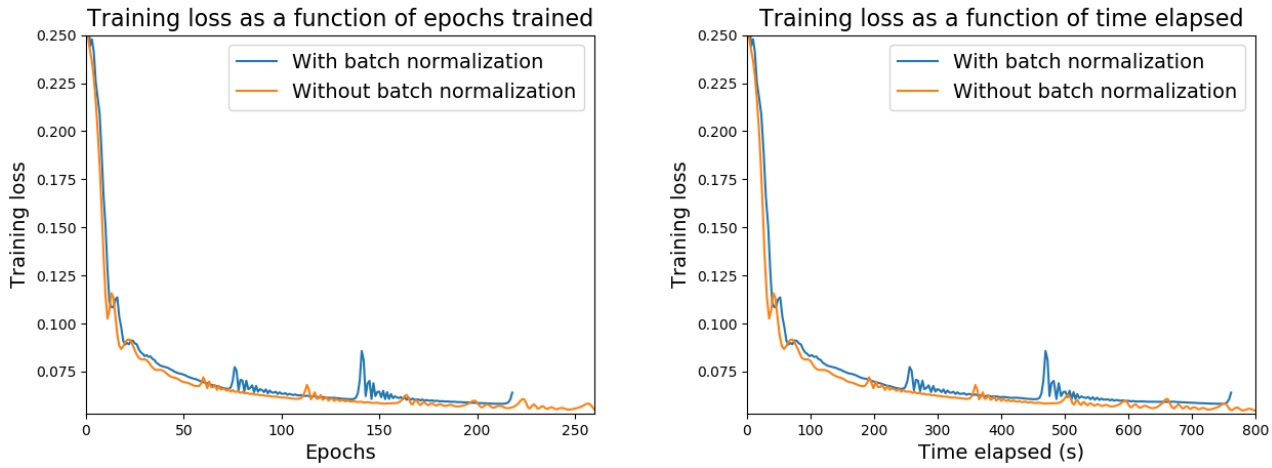

Figure 4.8: The number of epochs and the time it took for a model to converge when trained with neural networks including or excluding batch normalization. The convergence times between the two are extremely similar, thus batch normalization was not useful.

4.2.4 Hydrogen Inclusion

The inclusion of hydrogen atoms increased convergence time significantly, as shown in Figure 4.9. The MAE of a model including hydrogen atoms is 0.15630645 while a model trained in identical conditions, but excludes hydrogen, has MAE 0.15753435. Including hydrogen atoms as specific input graph nodes decreased MAE by 0.0012279, which is surprisingly low, considering the amount of extra information that is available in terms of bond order between hydrogen atoms and other heavier atoms.

4.2.5 Features Matrix

Table 4.2 lists the test errors for various models trained on different sets of features. The addition of each feature creates a better model with lower mean absolute error. Although aromaticity (AR) appears to break the trend when feature set used is {atomic number, atom type, aromaticity} with an MAE of 0.362391 compared the MAE of 0.362095 of the previous model with {atomic number, atom type}, it is likely due to the degree of convergence to which the model was trained. To be certain, we tested a model with the feature set {atomic number, aromaticity} and confirmed that it produces a test error lower than a model with only atomic number features.

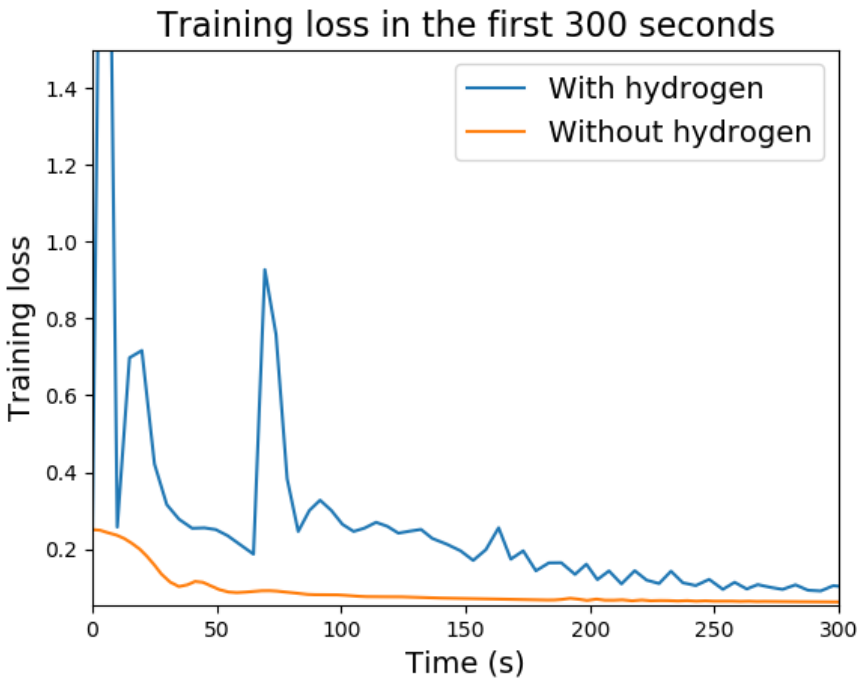

Figure 4.9: Training loss (mean square error) in the first 300 seconds for 2 models, one of which includes hydrogen atoms as input nodes while the other excludes hydrogen. Convergence is considerably slower for the model including hydrogen atoms.

A problem is that different features will likely affect the accuracies of each property differently. As all the MAEs are computed for the property C_v, we cannot truly know what features will be useful for other properties. To create the full picture, ideally one would test all features for all targets. However, it is a costly task with limited rewards, as 11 features * 11 properties = 121 models that need to be trained.

Table 4.2: Test errors for models trained with particular sets of features. "⊙" indicates the feature was used for a model, and "x" indicates the opposite. All of the features individually improved model accuracy. AN=atomic number, AT=atom type (binary encoding), AR=aromaticity, HB=hybridization, D=degree, VE=valence electrons, PC=partial charges.

AN	AT	AR	HB	D	VE	PC	Mean Absolute Error
⊙	x	x	x	x	x	x	0.39626253
⊙	x	⊙	x	x	x	x	0.36994627
⊙	⊙	x	x	x	x	x	0.36209515
⊙	⊙	⊙	x	x	x	x	0.36239088
⊙	⊙	⊙	⊙	x	x	x	0.33197278
⊙	⊙	⊙	⊙	⊙	x	x	0.18197903
⊙	⊙	⊙	⊙	⊙	⊙	x	0.16822363
⊙	⊙	⊙	⊙	⊙	⊙	⊙	0.17183454

4.2.6 Adjacency Matrix

The inclusion of the geodesic distance matrix and the aromaticity matrix did not have any notable effects on model accuracy, and caused slower convergence. The slower convergence can be directly attributed to the fact that with the addition of each matrix, there is at least one more weight matrix to train in each graph convolution layer. If a bias is added to a layer, then another extra weight matrix needs to be trained.

The geodesic distance matrix likely did not cause any improvements as the interaction between two atoms is more affected by their physical distance than their graph distance. The matrix does not provide Euclidean distance between atoms, but rather the degrees of separation, which is not proportional to the former quantity. Figure 4.10 shows that atoms can be separated by a large degree yet remain close in physical space.

The ineffectiveness of the aromaticity matrix is somewhat unsettling. Including aromaticity information as node features vastly improved errors, so presumptuously including this edge information should have an effect. Bond aromaticity is technically encoded in the bond order matrix, as aromatic bonds have order 1.5. However, we saw from earlier that the inclusion of binary-encoding of atom type was useful even though the atom type is technically already given in the atomic number. Clearly the aromaticity of bonds encoded as edge features need to be represented in a way that is more complicated than a simple binary matrix indicating whether an edge is aromatic. A better alternative could be to encode a binary matrix indicating whether atoms are in the same aromatic ring system, and is work for the future.

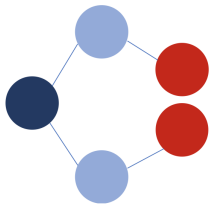

Figure 4.10: An example of an extreme case where the geodesic distance between the red atoms is large but they are physically close together.

4.2.7 Target

The relative accuracy and mean absolute error for each target can be found in Table 4.3. Due to some division by 0, the relative accuracy for dipole moment and LUMO exploded, however, their mean absolute errors and absolute differences still reflect how well the model fared.

Looking at the accuracies of predictions for each target, we see that the model is able to predict, on average, 3.4 to 9.3% within the true value. The absolute difference gives the average magnitude of the difference between predicted value and true value. Ideally the absolute difference should be within chemical accuracy listed in Table 4.4, which would indicate that the predictions are considered to be realistic by the chemistry community.

Comparing to other machine learning models predicting on the same data set, our models have test errors comparable to methods such as kernel ridge regression (KRR) or random forest (RF) with various input representations. Table 4.5 is the list of MAEs for various machine learning function-types with different input representations from Faber et al. [2]. Although our models are comparable to methods in the first four row-sections, GG and GC in the last two rows are part of the MPNN family and report test errors almost an entire magnitude lower.

The higher test errors of our models are very likely due to the fact that they were only trained on about 10k molecules. There is also always room for improvement in the neural network structure, as well as a more comprehensive feature set.

Property	MAE	Relative Accuracy	Absolute Difference
μ	0.2487	-	0.76761615 Debye
α	0.17488107	0.042068224	2.8819375 $Bohr^3$
r^2	0.17142713	0.09280105	95.79429 $Bohr^2$
zpve	0.09159453	0.04516752	0.006124639 Hartree
U_0	0.16959925	0.035733294	13.58376 Hartree
U	0.17136525	0.03550485	13.631208 Hartree
H	0.1705893	0.035686475	13.470697 Hartree
G	0.17115746	0.03559587	13.493001 Hartree
C_v	0.15753435	0.044081703	1.3183024 cal/(mol K)
HOMO	0.1790714	0.03386227	0.007918857 Hartree
LUMO	0.13892719	-	0.013056787 Hartree
ΔE	0.21738179	0.08842557	0.020741854 Hartree

Table 4.3: The mean absolute error (MAE), relative accuracy, and absolute difference for each thermochemical property of 2000 test molecules trained on a neural network with default settings.

Property	Unit	Chemical Accuracy
μ	Debye	0.1
α	$Bohr^3$	0.1
r^2	$Bohr^2$	1.2
zpve	Hartree	4.4e-5
U_0	Hartree	1.58e-3
U	Hartree	1.58e-3
H	Hartree	1.58e-3
G	Hartree	1.58e-3
C_v	cal/(mol K)	0.05
HOMO	Hartree	1.58e-3
LUMO	Hartree	1.58e-3
ΔE	Hartree	1.58e-3

Table 4.4: Chemical accuracies of targets used by Faber et al. [2], with units converted as appropriate. These are the accuracies required to make realistic chemical predictions.

		μ	α	$\varepsilon_{\text{HOMO}}$	$\varepsilon_{\text{LUMO}}$	$\Delta\varepsilon$	$\langle R^2 \rangle$	ZPVE	U_0	C_v
		Debye	Bohr^3	eV	eV	eV	Bohr^2	eV	eV	cal/molK
Mean		2.67	75.3	−6.54	0.322	6.86	1190	4.06	−76.6	31.6
MAD		1.17	6.29	0.439	1.05	1.07	203	0.717	8.19	3.21
Target[a]		0.10	0.10	0.043	0.043	0.043	1.2	0.0012	0.043	0.050
DFT[b]		0.10	0.4	2.0	2.6	1.2	-	0.0097	0.10	0.34
EN	CM	0.844	1.33	0.338	0.631	0.722	55.5	0.0265	0.911	0.906
	BOB	0.763	1.20	0.283	0.521	0.614	55.3	0.0232	0.602	0.700
	BAML	0.686	0.793	0.186	0.275	0.339	32.6	0.0129	0.212	0.439
	ECFP4	0.737	3.45	0.224	0.344	0.383	118	0.270	3.68	1.51
	HDAD	0.563	0.437	0.139	0.238	0.278	6.19	0.006 47	0.0983	0.0876
	HD	0.705	0.638	0.203	0.299	0.360	6.70	0.009 49	0.192	0.195
	MARAD	0.707	0.698	0.222	0.305	0.391	27.4	0.008 08	0.183	0.206
	Mean	0.715	1.22	0.228	0.373	0.441	43.1	0.0509	0.840	0.578
BR	CM	0.844	1.33	0.338	0.632	0.723	55.5	0.0265	0.911	0.907
	BOB	0.761	1.14	0.279	0.521	0.614	48.0	0.0222	0.586	0.684
	BAML	0.685	0.785	0.183	0.275	0.339	30.4	0.0129	0.202	0.444
	ECFP4	0.737	3.45	0.224	0.344	0.383	118	0.270	3.69	1.51
	HDAD	0.565	0.43	0.14	0.238	0.278	5.94	0.003 18	0.0614	0.0787
	HD	0.705	0.633	0.203	0.298	0.359	6.8	0.006 93	0.171	0.19
	MARAD	0.647	0.533	0.18	0.257	0.315	26.8	0.008 54	0.171	0.201
	Mean	0.706	1.19	0.221	0.367	0.430	41.7	0.0500	0.828	0.574
RF	CM	0.608	1.04	0.208	0.302	0.373	45.0	0.0199	0.431	0.777
	BOB	0.450	0.623	0.120	0.137	0.164	39.0	0.0111	0.202	0.443
	BAML	0.434	0.638	0.107	0.118	0.141	51.1	0.0132	0.200	0.451
	ECFP4	0.483	3.70	0.143	0.145	0.166	109	0.242	3.66	1.57
	HDAD	0.454	1.71	0.116	0.136	0.156	48.3	0.0525	1.44	0.895
	HD	0.457	1.66	0.126	0.139	0.150	46.8	0.0497	1.39	0.879
	MARAD	0.607	0.676	0.178	0.243	0.311	45.3	0.0102	0.21	0.311
	Mean	0.499	1.43	0.142	0.174	0.209	54.9	0.0569	1.08	0.761
KRR	CM	0.449	0.433	0.133	0.183	0.229	3.39	0.0048	0.128	0.118
	BOB	0.423	0.298	0.0948	0.122	0.148	**0.978**	0.003 64	0.0667	0.0917
	BAML	0.460	0.301	0.0946	0.121	0.152	3.9	0.003 31	0.0519	0.082
	ECFP4	0.490	4.17	0.124	0.133	0.174	128	0.248	4.25	1.84
	HDAD	0.334	0.175	0.0662	0.0842	0.107	1.62	**0.001 91**	**0.0251**	**0.0441**
	HD	0.364	0.299	0.0874	0.113	0.143	1.72	0.003 16	0.0644	0.0844
	MARAD	0.468	0.343	0.103	0.124	0.163	7.58	0.003 01	0.0529	0.0758
	Mean	0.427	0.859	0.101	0.126	0.159	21.1	0.0383	0.662	0.333
GG	MG	0.247	**0.161**	0.0567	0.0628	0.0877	6.30	0.004 31	0.0421	0.0837
GC	MG	**0.101**	0.232	**0.0549**	**0.0620**	**0.0869**	4.71	0.009 66	0.15	0.097

Table 4.5: The list of mean absolute errors (MAEs) for each of the targets. Taken from Faber et al. [2]. Original caption: "MAE on out-of-sample data of all representations for all regressors and properties at 117k (90%) training set size. Regressors include Bayesian ridge regression (BR), linear regression with elastic net regularization (EN), random forest (RF), kernel ridge regression (KRR) and molecular graphs based neural networks (GG/GC). The best combination for each property are highlighted in bold. [...]"

4.3 General Observations

While training all the neural networks, we found that although our optimizer, Adam, updates learning rates on its own, it is extremely important to provide a good initial learning rate.[5] If the initial learning rate is too large, the neural network will get stuck in a local minimum where training loss is high, and will never leave the local minimum unless the learning rate is manually set to be lower. This was especially true for complex models with more layers or more nodes per layer. If the initial learning rate is too small, the model will not converge fast enough. This is likely the root cause of the readout functions that did not fully converge even after thousands of training epochs.

Finally, training loss for each model was consistently less than validation loss and test losses but is often within a $\pm 30\%$ range of the validation/test losses. This suggests that the model can be made more complex.

[5]The learning rate is the rate at which the trainable weights change values.

Chapter 5

Future Work

As crucial code fixes were implemented late in the project timeline, many improvements are still in store for the project.

5.1 Neural Network Architecture

5.1.1 Implementation of a Dense Layer

Due to the data structure of our input data and the definition of the readout function, the idea of using a dense (fully-connected) layer in the style of a convolutional neural network (CNN) was initially discarded as redundant. The readout layer effectively acts as a fully-connected layer in that it performs some weighted sum of all nodes from the previous layer. However, there is the possibility of placing fully-connected layers in the message-passing phase (before the readout). Thus defining a new fully-connected layer type is an option to explore how including information from non-neighbour atoms during the message-passing phase may affect model performance.

5.1.2 Feature Set

Currently the interatomic distance between unbonded atoms is not available as an adjacency matrix element, due to complications with Python library RDKit. This is a coding challenge to be resolved in the future.

5.2 Memory, Runtime Improvements

In terms of input data representation, both the adjacency matrix and features matrix need to be reformulated to reduce memory usage. For m molecules in the input data and number of features f,the features matrix is dimension $N \times f$ and the adjacency matrix is $N \times N$ currently. N is total number of atoms defined by

$$N = \sum_{i=1}^{m} n_i$$

where n_i = number of atoms in molecule i.

If the average molecule size were 10 atoms, the full dataset will have an adjacency matrix of dimension $1,130,000 \times 1,130,000$. Even as sparse matrices, matrices containing the full 130k molecules will require significant memory to construct. It is currently limiting the number of training samples we can train on at a time.

Runtime can be optimized for many processes that were implemented as makeshift functions, and by delegating some tasks in self-defined functions to TensorFlow built-in functions.

5.3 Material Property Prediction

The eventual goal of this project is to extend the same framework to predict properties for crystalline solids similar to the crystal graph convolutions in the recent publication by Xie and Grossman[25].

Chapter 6

Conclusion

Our results show a graph convolution-based neural network under the MPNN framework training on only 10,000 samples are similar in performance to many non-MPNN machine learning methods, but do not perform as well as other MPNN variants.

For our particular flavour of MPNN, we found that batch normalization does not cause faster convergence in the training phase, and including hydrogen atoms as explicit input nodes gave more accurate models but significantly increased the training time. Including all features in the node feature set {atomic number, atomic type, aromaticity, hybridization, degree, valence, partial charges} is important as all of these features can increase model accuracy. One of the more interesting points is that the repeated encoding of features, i.e. the atom type represented in both the atomic number and a binary-encoding atom type features, still improved model accuracy. On the other hand, edge features in the adjacency matrix did not improve model accuracy, potentially also due to difficulties in convergence.

In general, the most complex models we tested did not necessarily fare better, as initial learning rate plays a big role in determining how well the model converges. Experiments where the convergence of the model is monitored need to be performed for the different readout functions and adjacency matrix representations, as there was not enough information to judge the effectiveness of any of the readout functions or edge features. Our models are not fully ready to predict molecular properties at chemical accuracy, but were not intentionally created to be. They are only trained on about 10% of the full database and were intended as explorations of model variations (in terms of parameters) while keeping the form of an MPNN. Regardless, more work needs to be done to eventually build a model that can make accurate predictions.

Bibliography

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.

[2] F. A. Faber, L. Hutchison, B. Huang, J. Gilmer, S. S. Schoenholz, G. E. Dahl, O. Vinyals, S. Kearnes, P. F. Riley, and O. A. Von Lilienfeld, "Fast machine learning models of electronic and energetic properties consistently reach approximation errors better than dft accuracy," *arXiv preprint arXiv:1702.05532*, 2017.

[3] D. Ceperley and B. Alder, "Quantum monte carlo," *Science*, vol. 231, no. 4738, pp. 555–560, 1986.

[4] L. Hedin, "L. hedin, phys. rev. 139, a796 (1965).," *Phys. Rev.*, vol. 139, p. A796, 1965.

[5] P. Hohenberg, "P. hohenberg and w. kohn, phys. rev. 136, b864 (1964).," *Phys. Rev.*, vol. 136, p. B864, 1964.

[6] W. Kohn, "W. kohn and lj sham, phys. rev. 140, a1133 (1965).," *Phys. Rev.*, vol. 140, p. A1133, 1965.

[7] 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, 2012.

[8] 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, 2014.

[9] J. E. Saal, S. Kirklin, M. Aykol, B. Meredig, and C. Wolverton, "Materials design and discovery with high-throughput density functional theory: the open quantum materials database (oqmd)," *Jom*, vol. 65, no. 11, pp. 1501–1509, 2013.

[10] S. Kirklin, J. E. Saal, B. Meredig, A. Thompson, J. W. Doak, M. Aykol, S. Rühl, and C. Wolverton, "The open quantum materials database (oqmd): assessing the accuracy of dft formation energies," *npj Computational Materials*, vol. 1, p. 15010, 2015.

[11] S. P. Ong, S. Cholia, A. Jain, M. Brafman, D. Gunter, G. Ceder, and K. A. Persson, "The materials application programming interface (API): A simple, flexible and efficient API for materials data based on REpresentational state transfer (REST) principles," *Computational Materials Science*, vol. 97, pp. 209–215, feb 2015.

[12] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical Science*, vol. 9, no. 2, pp. 513–530, 2018.

[13] A. Zakutayev, N. Wunder, M. Schwarting, J. D. Perkins, R. White, K. Munch, W. Tumas, and C. Phillips, "An open experimental database for exploring inorganic materials," *Scientific data*, vol. 5, p. 180053, 2018.

[14] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

[15] K. Gubaev, E. V. Podryabinkin, and A. V. Shapeev, "Machine learning of molecular properties: locality and active learning," *arXiv preprint arXiv:1709.07082*, 2017.

[16] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. Von Lilienfeld, "Fast and accurate modeling of molecular atomization energies with machine learning," *Physical review letters*, vol. 108, no. 5, p. 058301, 2012.

[17] K. Hansen, F. Biegler, R. Ramakrishnan, W. Pronobis, O. A. Von Lilienfeld, K.-R. Muller, and A. Tkatchenko, "Machine learning predictions of molecular properties: Accurate many-body potentials and nonlocality in chemical space," *The journal of physical chemistry letters*, vol. 6, no. 12, pp. 2326–2331, 2015.

[18] S. De, A. P. Bartók, G. Csányi, and M. Ceriotti, "Comparing molecules and solids across structural and alchemical space," *Physical Chemistry Chemical Physics*, vol. 18, no. 20, pp. 13754–13769, 2016.

[19] B. Huang and O. A. von Lilienfeld, "Communication: Understanding molecular representations in machine learning: The role of uniqueness and target similarity," 2016.

[20] H. Huo and M. Rupp, "Unified representation for machine learning of molecules and crystals," *arXiv preprint arXiv:1704.06439*, 2017.

[21] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nature communications*, vol. 8, p. 13890, 2017.

[22] K. Ryczko, K. Mills, I. Luchak, C. Homenick, and I. Tamblyn, "Convolutional neural networks for atomistic systems," *arXiv preprint arXiv:1706.09496*, 2017.

[23] J. Behler and M. Parrinello, "Generalized neural-network representation of high-dimensional potential-energy surfaces," *Physical review letters*, vol. 98, no. 14, p. 146401, 2007.

[24] L. Ward, A. Agrawal, A. Choudhary, and C. Wolverton, "A general-purpose machine learning framework for predicting properties of inorganic materials," *npj Computational Materials*, vol. 2, p. 16028, 2016.

[25] T. Xie and J. C. Grossman, "Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties," *Physical Review Letters*, vol. 120, no. 14, p. 145301, 2018.

[26] 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.

[27] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[28] D. K. 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," *CoRR*, vol. abs/1509.09292, 2015.

[29] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, *et al.*, "Interaction networks for learning about objects, relations and physics," in *Advances in neural information processing systems*, pp. 4502–4510, 2016.

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

[31] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[32] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010.

[33] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[34] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *CoRR*, vol. abs/1505.00853, 2015.

[35] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[36] "RDKit: Open-source cheminformatics." `http://www.rdkit.org`. [Online; accessed 11-April-2013].

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[38] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," Tech. Rep. UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.

[39] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.