Message Passing Neural Networks

10

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl

Abstract

Supervised learning on molecules has incredible potential to be useful in chemistry, drug discovery, and materials science. Luckily, several promising and closely related neural network models invariant to molecular symmetries have already been described in the literature. These models learn a message passing algorithm and aggregation procedure to compute a function of their entire input graph. In this chapter, we describe a general common framework for learning representations on graph data called message passing neural networks (MPNNs) and show how several prior neural network models for graph data fit into this framework. This chapter contains large overlap with Gilmer et al. (International Conference on Machine Learning, pp. 1263–1272, 2017), and has been modified to highlight more recent extensions to the MPNN framework.

10.1 Introduction

The past decade has seen remarkable success in the use of deep neural networks to understand and translate natural language [1], generate and decode complex audio

J. Gilmer (✉) · S. S. Schoenholz · G. E. Dahl
Google Brain, Mountain View, CA, USA
e-mail: gilmer@google.com; schsam@google.com; gdahl@google.com

P. F. Riley
Google, Mountain View, CA, USA
e-mail: pfr@google.com

O. Vinyals
DeepMind, London, UK
e-mail: vinyals@google.com

K. T. Schütt et al. (eds.), *Machine Learning Meets Quantum Physics*,
Lecture Notes in Physics 968, https://doi.org/10.1007/978-3-030-40245-7_10

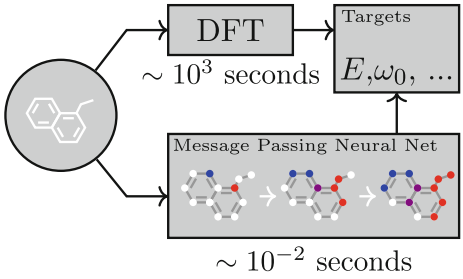

Fig. 10.1 A message passing neural network predicts quantum properties of an organic molecule by modeling a computationally expensive DFT calculation

signals [2], and infer features from real-world images and videos [3]. Although chemists have applied machine learning to many problems over the years, predicting the properties of molecules and materials using machine learning (and especially deep learning) is still in its infancy. A classic approach to applying machine learning to chemistry tasks [4–10] revolves around feature engineering. Recently, large scale quantum chemistry calculation and molecular dynamics simulations coupled with advances in high throughput experiments have begun to generate data at an unprecedented rate. Most classical techniques do not make effective use of the larger amounts of data that are now available. The time is ripe to apply more powerful and flexible machine learning methods to these problems, assuming we can find models with suitable inductive biases. The symmetries of atomic systems suggest neural networks that are invariant to the symmetries of graph structured data might also be appropriate for molecules. Sufficiently successful models could someday help automate challenging chemical search problems in drug discovery or materials science.

In this chapter, we describe a general framework for supervised learning on graphs called message passing neural networks (MPNNs) that simply abstracts the commonalities between several of the most promising prior neural models for graph structured data, in order to make it easier to understand the relationships between them and come up with novel variations. MPNNs have proven to have a strong inductive bias for graph data, with applications ranging from program synthesis [11], modeling citation networks [12], reinforcement learning [13], modeling physical systems, and predicting properties of molecules [14–17]. In this chapter, we describe the general MPNN framework before discussing specific applications of this framework in predicting the quantum mechanical properties of small organic molecules (see task schematic in Fig. 10.1).

In general, the search for practically effective machine learning (ML) models in a given domain proceeds through a sequence of increasingly realistic and interesting benchmarks. Here, we focus on the QM9 dataset as such a benchmark [18]. QM9 consists of 130k molecules with 13 properties for each molecule which are approximated by an expensive[1] quantum mechanical simulation method (DFT), to

[1]By comparison, the inference time of the neural networks discussed in this work is 300k times faster.

yield 13 corresponding regression tasks. These tasks serve as a useful benchmark for developing architectures with a strong inductive bias in the chemical domain. Additionally, QM9 also includes complete spatial information for the single low energy conformation of the atoms in the molecule that was used in calculating the chemical properties. QM9 therefore lets us consider both the setting where the complete molecular geometry is known (atomic distances, bond angles, etc.) and the setting where we need to compute properties that might still be *defined* in terms of the spatial positions of atoms, but where only the atom and bond information (i.e., graph) is available as input. In the latter case, the model must implicitly fit something about the computation used to determine a low energy 3D conformation and hopefully would still work on problems where it is not clear how to compute a reasonable 3D conformation.

When measuring the performance of our models on QM9, there are two important benchmark error levels. The first is the estimated average error of the DFT approximation to nature, which we refer to as "DFT error." The second, known as "chemical accuracy," is a target error that has been established by the chemistry community. Estimates of DFT error and chemical accuracy are provided for each of the 13 targets in Faber et al. [16]. One important goal of this line of research is to produce a model which can achieve chemical accuracy with respect to the *true* targets as measured by an extremely precise experiment. The ability to fit the DFT approximation to within chemical accuracy would be an encouraging step in this direction. In the rest of this paper, when we talk about chemical accuracy we generally mean with respect to our available ground truth labels.

In this chapter, we discuss our original application of the MPNN framework [17] to the QM9 dataset as well as related work [15, 19, 20] which have improved upon our initial results. To date MPNNs have been shown to predict DFT to within chemical accuracy on 11 out of 13 targets in the QM9 benchmark, and are less than DFT error on all 13 targets. We also show that MPNNs can predict DFT to within chemical accuracy on 5 out of 13 targets while operating on the topology of the molecule alone (with no spatial information as input). In this sparse setting, we explore some simple graph preprocessing techniques which can improve performance. Finally, we discuss a general method to train MPNNs with larger node representations without a corresponding increase in computation time or memory, yielding a substantial savings for high dimensional node representations.

10.2 Message Passing Neural Networks

There are many notable examples of models from the literature that we can describe using the message passing neural networks (MPNN) framework; in this section, we discuss a few specific instances. For simplicity, we describe MPNNs which operate on undirected graphs G with node features x_v and edge features e_{vw}. It is trivial to extend the formalism to directed multigraphs. The forward pass has two phases, a message passing phase and a readout phase. The message passing phase runs for T time steps and is defined in terms of message functions M_t and vertex update

functions U_t. During the message passing phase, hidden states h_v^t at each node in the graph are updated based on aggregated messages m_v^{t+1} according to

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{10.1}$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{10.2}$$

where in the sum, $N(v)$ denotes the neighbors of v in graph G. Note the sum in Eq. 10.1 can be replaced with any permutation invariant function α operating on the set of neighbors $N(v)$. Some instantiations of MPNNs use self-attention in this aggregation step [21]. It has also been shown [22] that for more general aggregation functions α, other model families such as *non local neural networks* [23] can be described in this framework. The readout phase computes a feature vector for the whole graph using some readout function R according to

$$\hat{y} = R(\{h_v^T \mid v \in G\}). \tag{10.3}$$

The message functions M_t, vertex update functions U_t, and readout function R are all learned differentiable functions. R operates on the set of node states and must be invariant to permutations of the node states in order for the MPNN to be invariant to graph isomorphism. Some MPNNs [19, 24] learn edge features by introducing hidden states for all edges in the graph $h_{e_{vw}}^t$ and updating them using a learned differentiable edge function E according to Eq. 10.4.

$$e_{vw}^{t+1} = E(h_v^t, h_w^t, e_{vw}^t) \tag{10.4}$$

In what follows, we describe several models in the literature as they fit into the MPNN framework by specifying the specific message, update, and readout functions used.

10.2.1 Convolutional Networks for Learning Molecular Fingerprints [14]

The message function used is

$$M(h_v, h_w, e_{vw}) = (h_w, e_{vw}), \tag{10.5}$$

where $(\cdot\,,\,\cdot)$ denotes concatenation. The vertex update function used is

$$U_t(h_v^t, m_v^{t+1}) = \sigma(H_t^{\deg(v)} m_v^{t+1}), \tag{10.6}$$

where σ is the sigmoid function, $\deg(v)$ is the degree of vertex v, and H_t^N is a learned matrix for each time step t and vertex degree N. The readout function has skip connections to all previous hidden states h_v^t and is

$$R = f\left(\sum_{v,t} \text{softmax}(W_t h_v^t)\right), \tag{10.7}$$

where f is a neural network and W_t are learned readout matrices, one for each time step t. This message passing scheme may be problematic since the resulting message vector is

$$m_v^{t+1} = \left(\sum h_w^t, \sum e_{vw}\right), \tag{10.8}$$

which separately sums over connected nodes and connected edges. It follows that the message passing implemented in Duvenaud et al. [14] is unable to identify correlations between edge states and node states.

10.2.2 Gated Graph Neural Networks (GG-NN) [25]

The message function used is

$$M_t(h_v^t, h_w^t, e_{vw}) = A_{e_{vw}} h_w^t, \tag{10.9}$$

where $A_{e_{vw}}$ is a learned matrix, one for each edge label e (the model assumes discrete edge types). The update function is

$$U_t = \text{GRU}(h_v^t, m_v^{t+1}), \tag{10.10}$$

where GRU is the gated recurrent unit introduced by Cho et al. [26]. This work used weight tying, so the same update function is used at each time step t. Finally, the readout function is

$$R = \sum_{v \in V} \sigma\left(i(h_v^T, h_v^0)\right) \odot \left(j(h_v^T)\right), \tag{10.11}$$

where i and j are neural networks, and $\odot$ denotes element-wise multiplication.

10.2.3 Interaction Networks [27]

This work considered both the case where there is a target at each node in the graph and where there is a graph level target. It also considered the case where there are node level effects applied at each time step, in such a case the update function takes

as input the concatenation (h_v, x_v, m_v) where x_v is an external vector representing some outside influence on the vertex v. The message function

$$M(h_v, h_w, e_{vw}) \tag{10.12}$$

is a neural network which takes the concatenation (h_v, h_w, e_{vw}). The vertex update function

$$U(h_v, x_v, m_v) \tag{10.13}$$

is a neural network which takes as input the concatenation (h_v, x_v, m_v). Finally, in the case where there is a graph level output, the readout function is

$$R = f\left(\sum_{v \in G} h_v^T\right), \tag{10.14}$$

where f is a neural network which takes the sum of the final hidden states h_v^T. Note the original work only defined the model for $T = 1$.

10.2.4 Molecular Graph Convolutions [24]

To the best of our knowledge, this work was the first to consider learned edge representations e_{vw}^t which are updated during the message passing phase. The message function used for node messages is

$$M(h_v^t, h_w^t, e_{vw}^t) = e_{vw}^t. \tag{10.15}$$

The vertex update function is

$$U_t(h_v^t, m_v^{t+1}) = \alpha\left(W_1\left(\alpha\left(W_0 h_v^t\right),\, m_v^{t+1}\right)\right), \tag{10.16}$$

where $(\cdot\,,\,\cdot)$ denotes concatenation, α is the ReLU activation, and W_1, W_0 are learned weight matrices. The edge state update is defined by

$$e_{vw}^{t+1} = E(e_{vw}^t, h_v^t, h_w^t) = \alpha\left(W_4\left(\alpha\left(W_2, e_{vw}^t\right), \alpha\left(W_3\left(h_v^t, h_w^t\right)\right)\right)\right), \tag{10.17}$$

where the W_i are also learned weight matrices.

10.2.5 Deep Tensor Neural Networks [15]

The message from w to v is computed by

$$M_t(h_v^t, h_w^t, e_{vw}) = \tanh\left(W^{fc}((W^{cf}h_w^t + b_1) \odot (W^{df}e_{vw} + b_2))\right), \tag{10.18}$$

where W^{fc}, W^{cf}, W^{df} are matrices and b_1, b_2 are bias vectors. The update function used is

$$U_t(h_v^t, m_v^{t+1}) = h_v^t + m_v^{t+1}. \tag{10.19}$$

The readout function passes each node independently through a single hidden layer neural network and sums the outputs, in particular

$$R = \sum_v \mathrm{NN}(h_v^T). \tag{10.20}$$

10.2.6 SchNet with Edge Updates [19]

This work extends SchNet [20] to include an edge update function. The message function used is

$$M_t(h_v^t, h_w^t, e_{vw}^t) = (W_1^t h_w^t) \odot g(W_3^t g(W_2^t e_{vw}^t)) \tag{10.21}$$

where g is the softplus function, and $\odot$ denotes the element-wise multiplication operation. The update function used is

$$U_t(h_v^t, m_v^{t+1}) = h_v^t + W_5^t g(W_4^t m_v^{t+1}) \tag{10.22}$$

while the edge update is

$$E(h_v^t, h_w^t, e_{vw}^t) = g(W_{E2}^{t+1} g(W_{E1}^{t+1}(h_v^t; h_w^t; e_{vw}^t))) \tag{10.23}$$

and the readout is

$$R = \sum_{v \in G} W_7 g(W_6 h_v^T). \tag{10.24}$$

10.2.7 Laplacian-Based Methods [12, 28, 29]

These methods generalize the notion of the convolution operation typically applied to image datasets to an operation that operates on an arbitrary graph G with a real

valued adjacency matrix A. The operations defined in Bruna et al. [28], Defferrard et al. [29] result in message functions of the form

$$M_t(h_v^t, h_w^t) = C_{vw}^t h_w^t, \tag{10.25}$$

where the matrices C_{vw}^t are parameterized by the eigenvectors of the graph Laplacian L, and the learned parameters of the model. The vertex update function used is

$$U_t(h_v^t, m_v^{t+1}) = \sigma(m_v^{t+1}), \tag{10.26}$$

where σ is some pointwise non-linearity (such as ReLU). One model [12] results in a message function

$$M_t(h_v^t, h_w^t) = c_{vw} h_w^t \tag{10.27}$$

where $c_{vw} = (\deg(v)\deg(w))^{-1/2} A_{vw}$. The vertex update function is

$$U_v^t(h_v^t, m_v^{t+1}) = \text{ReLU}(W^t m_v^{t+1}). \tag{10.28}$$

The exact expressions for the C_{vw}^t and the derivation of the reformulation of these models as MPNNs can be found in [17].

10.3 MPNNs for Modeling Molecules

We began our exploration of MPNNs around the GG-NN model which we believe to be a strong baseline. We focused on trying different message functions, output functions, finding the appropriate input representation, and properly tuning hyper-parameters.

For the rest of the chapter, we use d to denote the dimension of the internal hidden representation of each node in the graph, and n to denote the number of nodes in the graph. Our implementation of MPNNs in general operates on directed graphs with a separate message channel for incoming and outgoing edges, in which case the incoming message m_v is the concatenation of m_v^{in} and m_v^{out}, this was also used in Li et al. [25]. When we apply this to undirected chemical graphs, we treat the graph as directed, where each original edge becomes both an incoming and outgoing edge with the same label. Note there is nothing special about the direction of the edge, it is only relevant for parameter tying. Treating undirected graphs as directed means that the size of the message channel is $2d$ instead of d.

The input to our MPNN model is a set of feature vectors for the nodes of the graph, x_v, and an adjacency matrix A with vector valued entries to indicate different bonds in the molecule as well as pairwise spatial distance between two atoms. We experimented as well with the message function used in the GG-NN family, which assumes discrete edge labels, in which case the matrix A has entries in a discrete

alphabet of size k. The initial hidden states h_v^0 are set to be the atom input feature vectors x_v and are padded up to some larger dimension d. All of our experiments used weight tying at each time step t, and a GRU [26] for the update function as in the GG-NN family.

10.3.1 Message Functions

Matrix Multiplication We started with the message function used in GG-NN which is defined by the equation

$$M(h_v, h_w, e_{vw}) = A_{e_{vw}} h_w.$$

Edge Network To allow vector valued edge features, we propose the message function

$$M(h_v, h_w, e_{vw}) = A(e_{vw}) h_w,$$

where $A(e_{vw})$ is a neural network which maps the edge vector e_{vw} to a $d \times d$ matrix.

Pair Message One property that the matrix multiplication rule has is that the message from node w to node v is a function only of the hidden state h_w and the edge e_{vw}. In particular, it does not depend on the hidden state h_v^t. In theory, a network may be able to use the message channel more efficiently if the node messages are allowed to depend on both the source and destination node. Thus, we also tried using a variant on the message function as described in [27]. Here, the message from w to v along edge e is

$$m_{wv} = f\left(h_w^t, h_v^t, e_{vw}\right),$$

where f is a neural network.

When we apply the above message functions to directed graphs, there are two separate functions used, M^{in} and an M^{out}. Which function is applied to a particular edge e_{vw} depends on the direction of that edge.

10.3.2 Virtual Graph Elements

We explored two different ways to change how the messages are passed throughout the model. The simplest modification involves adding a separate "virtual" edge type for pairs of nodes that are not connected. This can be implemented as a data preprocessing step and allows information to travel long distances during the propagation phase.

We also experimented with using a latent "master" node, which is connected to every input node in the graph with a special edge type. The master node serves as a global scratch space that each node both reads from and writes to in every step of message passing. We allow the master node to have a separate node dimension d_{master}, as well as separate weights for the internal update function (in our case a GRU). This allows information to travel long distances during the propagation phase. It also, in theory, allows additional model capacity (e.g., large values of d_{master}) without a substantial hit in performance, as the complexity of the master node model is $O(|E|d^2 + nd_{master}^2)$.

10.3.3 Readout Functions

We experimented with two readout functions. First is the readout function used in GG-NN, which is defined by Eq. 10.11. Second is a set2set model from Vinyals et al. [30]. The set2set model is specifically designed to operate on sets and should have more expressive power than simply summing the final node states. This model first applies a linear projection to each tuple (h_v^T, x_v) and then takes as input the set of projected tuples $T = \{(h_v^T, x_v)\}$. Then, after M steps of computation, the set2set model produces a graph level embedding q_t^* which is invariant to the order of the tuples T. We feed this embedding q_t^* through a neural network to produce the output.

10.3.4 Multiple Towers

One issue with MPNNs is scalability. In particular, a single step of the message passing phase for a dense graph requires $O(n^2d^2)$ floating point multiplications. As n or d get large this can be computationally expensive. To address this issue, we break the d dimensional node embeddings h_v^t into k different d/k dimensional embeddings $h_v^{t,k}$ and run a propagation step on each of the k copies separately to get temporary embeddings $\{\tilde{h}_v^{t+1,k}, v \in G\}$, using separate message and update functions for each copy. The k temporary embeddings of each node are then mixed together according to the equation

$$\left(h_v^{t,1}, h_v^{t,2}, \ldots, h_v^{t,k}\right) = g\left(\tilde{h}_v^{t,1}, \tilde{h}_v^{t,2}, \ldots, \tilde{h}_v^{t,k}\right) \tag{10.29}$$

where g denotes a neural network and $(x, y, \ldots)$ denotes concatenation, with g shared across all nodes in the graph. This mixing preserves the invariance to permutations of the nodes, while allowing the different copies of the graph to communicate with each other during the propagation phase. This can be advantageous in that it allows larger hidden states for the same number of parameters, which yields a computational speedup in practice. On dense graphs, when the message function is matrix multiplication (as in GG-NN) a propagation step of a single copy takes

$O\left(n^2(d/k)^2\right)$ time, and there are k copies, therefore the overall time complexity is $O\left(n^2d^2/k\right)$, with some additional overhead due to the mixing network. For $k = 8$, $n = 9$, and $d = 200$, we see a factor of 2 speedup in inference time over a $k = 1$, $n = 9$, and $d = 200$ architecture. This variation would be most useful for larger molecules, for instance, molecules from GDB-17 [31].

10.4 Input Representation

There are a number of features available for each atom in a molecule which capture both properties of the electrons in the atom and the bonds that the atom participates in. For a list of all of the features, see Table 10.1. We experimented with making the hydrogen atoms explicit nodes in the graph (as opposed to simply including the count as a node feature), in which case graphs have up to 29 nodes. Note that having larger graphs significantly slows training time, in this case by a factor of roughly 10. For the adjacency matrix, there are three edge representations used depending on the model.

Chemical Graph In the absence of distance information, adjacency matrix entries are discrete bond types: single, double, triple, or aromatic.

Distance Bins The matrix multiply message function assumes discrete edge types, so to include distance information we bin bond distances into 10 bins, the bins are obtained by uniformly partitioning the interval [2, 6] into 8 bins, followed by adding a bin [0, 2] and [6, ∞]. These bins were hand chosen by looking at a histogram of all distances. The adjacency matrix then has entries in an alphabet of size 14, indicating bond type for bonded atoms and distance bin for atoms that are not bonded. We found the distance for bonded atoms to be almost completely determined by bond type.

Raw Distance Feature When using a message function which operates on vector valued edges, the entries of the adjacency matrix are then 5 dimensional, where the first dimension indicates the Euclidean distance between the pair of atoms, and the remaining four are a one-hot encoding of the bond type.

Table 10.1 Atom features

Feature	Description
Atom type	H, C, N, O, F (one-hot)
Atomic number	Number of protons (integer)
Acceptor	Accepts electrons (binary)
Donor	Donates electrons (binary)
Aromatic	In an aromatic system (binary)
Hybridization	sp, sp2, sp3 (one-hot or null)
Number of hydrogens	(integer)

10.5 Training

Each model and target combination was trained using a uniform random hyperparameter search with 50 trials. T was constrained to be in the range $3 \leq T \leq 8$ (in practice, any $T \geq 3$ works). The number of set2set computations M was chosen from the range $1 \leq M \leq 12$. All models were trained using SGD with the ADAM optimizer (Kingma and Ba [32]), with batch size 20 for 3 million steps (540 epochs). The initial learning rate was chosen uniformly between $1e^{-5}$ and $5e^{-4}$. We used a linear learning rate decay that began between 10 and 90% of the way through training and the initial learning rate l decayed to a final learning rate $l \cdot F$, using a decay factor F in the range [0.01, 1].

The QM9 dataset has 130,462 molecules in it. We randomly chose 10,000 samples for validation, 10,000 samples for testing, and used the rest for training. We use the validation set to do early stopping and model selection and we report scores on the test set. All targets were normalized to have mean 0 and variance 1. We minimize the mean squared error between the model output and the target, although we evaluate mean absolute error.

10.6 Results

In all of our tables, we report the ratio of the mean absolute error (MAE) of our models with the provided estimate of chemical accuracy for that target. Thus, any model with error ratio less than 1 has achieved chemical accuracy for that target. A list of chemical accuracy estimates for each target can be found in Faber et al. [16]. In this way, the MAE of our models can be calculated as (Error Ratio) $\times$ (Chemical Accuracy). Note, unless otherwise indicated, all tables display result of models trained individually on each target (as opposed to training one model to predict all 13).

We performed numerous experiments in order to find the best possible MPNN on this dataset as well as the proper input representation. In our experiments, we found that including the complete edge feature vector (bond type, spatial distance) and treating hydrogen atoms as explicit nodes in the graph to be very important for a number of targets. We also found that training one model per target consistently outperformed jointly training on all 13 targets. In some cases, the improvement was up to 40%. Our best MPNN variant used the edge network message function, set2set output, and operated on graphs with explicit hydrogens.

In Table 10.2, we compare the performance of our best MPNN variant (denoted with **enn-s2s**) with several baselines which use feature engineering as reported in Faber et al. [16]. We also show the performance of several other MPNNs, the GG-NN architecture [25], graph convolutions [24], and two more recent MPNNs which have improved upon our initial results reported in [17]. These include the SchNet MPNN [20], which developed message functions better suited for chemical systems, as well as [19] which added edge updates to the SchNet architectures. As of writing, the current state of the art on this benchmark is the MPNN described in [19]. For the exact functions used in these two models, see Sect. 10.2. For clarity, the error

Table 10.2 Comparison of hand engineered approaches (left) with different MPNNs (right)

Target	BAML	BOB	CM	ECFP4	HDAD	GC	GG-NN	enn-s2s	SchNet	SchNetE
mu	4.34	4.23	4.49	4.82	3.34	0.70	1.22	**0.30**	0.33	**0.29**
alpha	3.01	2.98	4.33	34.54	1.75	2.27	1.55	0.92	2.35	**0.77**
HOMO	2.20	2.20	3.09	2.89	1.54	1.18	1.17	0.99	0.95	**0.85**
LUMO	2.76	2.74	4.26	3.10	1.96	1.10	1.08	0.87	0.79	**0.72**
gap	3.28	3.41	5.32	3.86	2.49	1.78	1.70	1.60	1.47	**1.35**
R2	3.25	0.80	2.83	90.68	1.35	4.73	3.99	0.15	**0.06**	**0.06**
ZPVE	3.31	3.40	4.80	241.58	1.91	9.75	2.52	1.44	**1.37**	**1.35**
U0	1.21	1.43	2.98	85.01	0.58	3.02	0.83	0.45	0.33	**0.24**
U	1.22	1.44	2.99	85.59	0.59	3.16	0.86	0.45	0.44	**0.25**
H	1.22	1.44	2.99	86.21	0.59	3.19	0.81	0.39	0.33	**0.26**
G	1.20	1.42	2.97	78.36	0.59	2.95	0.78	0.44	0.33	**0.28**
Cv	1.64	1.83	2.36	30.29	0.88	1.45	1.19	0.80	0.66	**0.64**
Omega	0.27	0.35	1.32	1.47	0.34	0.32	0.53	0.19	N/A	N/A

Table 10.3 Models trained without spatial information

Model	Average error ratio
GG-NN	3.47
GG-NN + virtual edge	2.90
GG-NN + master node	2.62
GG-NN + set2set	**2.57**

Table 10.4 Towers vs vanilla GG-NN (no explicit hydrogen)

Model	Average error ratio
GG-NN + joint training	1.92
Towers8 + joint training	**1.75**
GG-NN + individual training	1.53
Towers8 + individual training	**1.37**

ratios of the best non-ensemble models are shown in bold. Overall, the best MPNNs achieve chemical accuracy on 11 out of 13 targets.

Training Without Spatial Information We also experimented in the setting where spatial information is not included in the input. In general, we find that augmenting the MPNN with some means of capturing long range interactions between nodes in the graph greatly improves performance in this setting. To demonstrate this, we performed 4 experiments, one where we train the GG-NN model on the sparse graph, one where we add virtual edges, one where we add a master node, and one where we change the graph level output to a set2set output. The error ratios averaged across the 13 targets are shown in Table 10.3. Overall, these three modifications help on all 13 targets, and the Set2Set output achieves chemical accuracy on 5 out of 13 targets. The experiments shown in Tables 10.3 and 10.4 were run with a partial charge feature as a node input. This feature is an output of the DFT calculation and thus could not be used in an applied setting. The numbers we report in Table 10.2 do not use this feature.

Table 10.5 Results from training the edge network + set2set model on different sized training sets (N denotes the number of training samples)

Target	$N = 11$k	$N = 35$k	$N = 58$k	$N = 82$k	$N = 110$k
mu	1.28	0.55	0.44	0.32	0.30
alpha	2.76	1.59	1.26	1.09	0.92
HOMO	2.33	1.50	1.34	1.19	0.99
LUMO	2.18	1.47	1.19	1.10	0.87
gap	3.53	2.34	2.07	1.84	1.60
R2	0.28	0.22	0.21	0.21	0.15
ZPVE	2.52	1.78	1.69	1.68	1.27
U0	1.24	0.69	0.58	0.62	0.45
U	1.05	0.69	0.60	0.52	0.45
H	1.14	0.64	0.65	0.53	0.39
G	1.23	0.62	0.64	0.49	0.44
Cv	1.99	1.24	0.93	0.87	0.80
Omega	0.28	0.25	0.24	0.15	0.19

Towers Our original intent in developing the towers variant was to improve training time, as well as to allow the model to be trained on larger graphs. However, we also found some evidence that the multi-tower structure improves generalization performance. In Table 10.4, we compare GG-NN + towers + set2set output vs a baseline GG-NN + set2set output when distance bins are used. We do this comparison in both the joint training regime and when training one model per target. The towers model outperforms the baseline model on 12 out of 13 targets in both individual and joint target training. We believe the benefit of towers is that it resembles training an ensemble of models. Unfortunately, our attempts so far at combining the towers and edge network message function have failed to further improve performance, possibly because the combination makes training more difficult.[2]

Additional Experiments In preliminary experiments, we tried disabling weight tying across different time steps. However, we found that the most effective way to increase performance was to tie the weights and use a larger hidden dimension d. We also early on found the pair message function to perform worse than the edge network function. This included a toy pathfinding problem which was originally designed to benefit from using pair messages. Also, when trained jointly on the 13 targets the edge network function outperforms pair message on 11 out of 13 targets, and has an average error ratio of 1.53 compared to 3.98 for pair message. Given the difficulties with training this function we did not pursue it further. For performance on smaller sized training sets, see Table 10.5.

[2]As reported in Schütt et al. [15]. The model was trained on a different train/test split with 100k training samples vs 110k used in our experiments.

10.7 Conclusions and Future Work

Our results show that MPNNs with the appropriate message, update, and output functions have a useful inductive bias for predicting molecular properties, outperforming several strong baselines, and eliminating the need for complicated feature engineering. Moreover, our results also reveal the importance of allowing long range interactions between nodes in the graph with either the master node or the set2set output. The towers variation makes these models more scalable, but additional improvements will be needed to scale to much larger graphs.

An important future direction is to design MPNNs that can generalize effectively to larger graphs than those appearing in the training set or at least work with benchmarks designed to expose issues with generalization across graph sizes. Generalizing to larger molecule sizes seems particularly challenging when using spatial information. First of all, the pairwise distance distribution depends heavily on the number of atoms. Second, our most successful ways of using spatial information create a fully connected graph where the number of incoming messages also depends on the number of nodes. To address the second issue, we believe that adding an attention mechanism over the incoming message vectors could be an interesting direction to explore.

Acknowledgments We would like to thank Lukasz Kaiser, Geoffrey Irving, Alex Graves, and Yujia Li for helpful discussions. Thank you to Adrian Roitberg for pointing out an issue with the use of partial charges in an earlier version of this work.

References

1. Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., (2016, preprint). arXiv:1609.08144
2. G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, et al., IEEE Signal. Proc. Mag. **29**(6), 82 (2012)
3. A. Krizhevsky, I. Sutskever, G.E. Hinton, *Advances in Neural Information Processing Systems* (The MIT Press, Cambridge, 2012), pp. 1097–1105
4. K. Hansen, F. Biegler, R. Ramakrishnan, W. Pronobis, O.A. von Lilienfeld, K.-R. Müller, A. Tkatchenko, J. Phys. Chem. Lett. **6**(12), 2326 (2015). https://doi.org/10.1021/acs.jpclett.5b00831
5. B. Huang, O.A. von Lilienfeld, J. Chem. Phys. **145**(16), 161102 (2016). https://doi.org/10.1063/1.4964627
6. M. Rupp, A. Tkatchenko, K.-R. Müller, O.A. von Lilienfeld, Phys. Rev. Lett. **108**(5), 058301 (2012)
7. D. Rogers, M. Hahn, J. Chem. Inf. Model. **50**(5), 742 (2010)
8. G. Montavon, K. Hansen, S. Fazli, M. Rupp, F. Biegler, A. Ziehe, A. Tkatchenko, O.A. von Lilienfeld, K.-R. Müller, *Advances in Neural Information Processing Systems* (Curran Associates, Red Hook, 2012), pp. 440–448
9. J. Behler, M. Parrinello, Phys. Rev. Lett. **98**, 146401 (2007). https://doi.org/10.1103/PhysRevLett.98.146401
10. S.S. Schoenholz, E.D. Cubuk, D.M. Sussman, E. Kaxiras, A.J. Liu, A structural approach to relaxation in glassy liquids. Nat. Phys. **12**(5), 469–471 (2016)
11. M. Allamanis, M. Brockschmidt, M. Khademi, (2017, preprint). arXiv:1711.00740

12. T.N. Kipf, M. Welling, ArXiv e-prints (2016)
13. V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, et al. (2018, preprint). arXiv:1806.01830
14. D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, *Advances in Neural Information Processing Systems* (2015), pp. 2224–2232
15. K.T. Schütt, F. Arbabzadah, S. Chmiela, K.R. Müller, A. Tkatchenko, Quantum-chemical insights from deep tensor neural networks. Nat. Commun. **8**(1), 1–8 (2017)
16. F. Faber, L. Hutchison, B. Huang, J. Gilmer, S.S. Schoenholz, G.E. Dahl, O. Vinyals, S. Kearnes, P.F. Riley, O.A. von Lilienfeld, (2017). https://arxiv.org/abs/1702.05532
17. J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, *International Conference on Machine Learning* (2017), pp. 1263–1272
18. R. Ramakrishnan, P.O. Dral, M. Rupp, O.A. Von Lilienfeld, Quantum chemistry structures and properties of 134 kilo molecules. Sci. Data **1**, 140022 (2014)
19. P.B. Jørgensen, K.W. Jacobsen, M.N. Schmidt (2018, preprint). arXiv:1806.03146
20. K.T. Schütt, P.-J. Kindermans, H.E.S. Felix, S. Chmiela, A. Tkatchenko, K.-R. Müller, *Advances in Neural Information Processing Systems* (Curran Associates, Red Hook, 2017), pp. 991–1001
21. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, (2017, preprint). arXiv:1710.10903
22. P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., (2018, preprint). arXiv:1806.01261
23. X. Wang, R. Girshick, A. Gupta, K. He, (2017, preprint). arXiv:1711.07971
24. S. Kearnes, K. McCloskey, M. Berndl, V. Pande, P. Riley, J. Comput.-Aided Mol. Des. **30**(8), 595 (2016)
25. Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, *International Conference on Learning Representations, ICLR* (2016)
26. K. Cho, B. Van Merriënboer, D. Bahdanau, Y. Bengio, (2014, preprint). arXiv:1409.1259
27. P. Battaglia, R. Pascanu, M. Lai, D.J. Rezende, K. Kavukcuoglu, *Advances in Neural Information Processing Systems* (Curran Associates, Red Hook, 2016), pp. 4502–4510
28. J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, (2013, preprint). arXiv:1312.6203
29. M. Defferrard, X. Bresson, P. Vandergheynst, *Advances in Neural Information Processing Systems* (Curran Associates, Red Hook, 2016), pp. 3837–3845
30. O. Vinyals, S. Bengio, M. Kudlur, (2015, preprint). arXiv:1511.06391
31. L. Ruddigkeit, R. Van Deursen, L.C. Blum, J.-L. Reymond, J. Chem. Inf. Model. **52**(11), 2864 (2012)
32. D. Kingma, J. Ba, (2014, preprint). arXiv:1412.6980