Graph Informer Networks for Molecules

Jaak Simm[*]
ESAT-STADIUS
KU Leuven
Leuven, 3001, Belgium
jaak.simm@esat.kuleuven.be

Adam Arany[*]
ESAT-STADIUS
KU Leuven
Leuven, 3001, Belgium
adam.arany@esat.kuleuven.be

Edward De Brouwer
ESAT-STADIUS
KU Leuven
Leuven, 3001, Belgium
edward.debrouwer@esat.kuleuven.be

Yves Moreau
ESAT-STADIUS
KU LEUVEN
Leuven, 3001, Belgium
moreau@esat.kuleuven.be

Abstract

In machine learning, chemical molecules are often represented by sparse high-dimensional vectorial fingerprints. However, a more natural mathematical object for molecule representation is a graph, which is much more challenging to handle from a machine learning perspective. In recent years, several deep learning architectures have been proposed to directly learn from the graph structure of chemical molecules, including graph convolution (Duvenaud et al., 2015) and graph gating networks (Li et al., 2015). Here, we introduce *Graph Informer*, a route-based multi-head attention mechanism inspired by transformer networks (Vaswani et al., 2017), which incorporates features for node pairs. We show empirically that the proposed method gives significant improvements over existing approaches in prediction tasks for 13C nuclear magnetic resonance spectra and for drug bioactivity. These results indicate that our method is well suited for both node-level and graph-level prediction tasks.

1   Introduction

Graphs are used as a natural representation for objects in many important domains; for example, for compounds in computational chemistry and for protein–protein interaction networks in systems biology. Machine learning approaches for graphs fall into two main lines of research: the *spectral* and the *spatial* approach. The spectral approach relies on the eigenvalue decomposition of the Laplacian of the graph and is well suited for problems involving a single fixed graph structure.

By contrast, *spatial* graph methods work directly on the nodes and edges of the graph. Convolutional neural networks (CNN) have inspired many spatial methods. Similarly to a local convolution filter running on the 2D grid of an image, spatial approaches update the hidden vectors of a graph node by aggregating the hidden vectors of its neighbors. Several methods have been proposed, which differ on exactly how to carry out this update. The most straightforward approach is to sum (or average) the hidden vectors of neighbors, and then transform the results with a linear layer and a nonlinearity (*e.g.*, as proposed in Neural Fingerprints (Duvenaud et al., 2015) and Graph Convolutional Networks (Kipf & Welling, 2016)). Instead of a simple dense feedforward layer, some methods use GRU-based gating (Li et al., 2015), edge hidden vectors (Kearnes et al., 2016), and
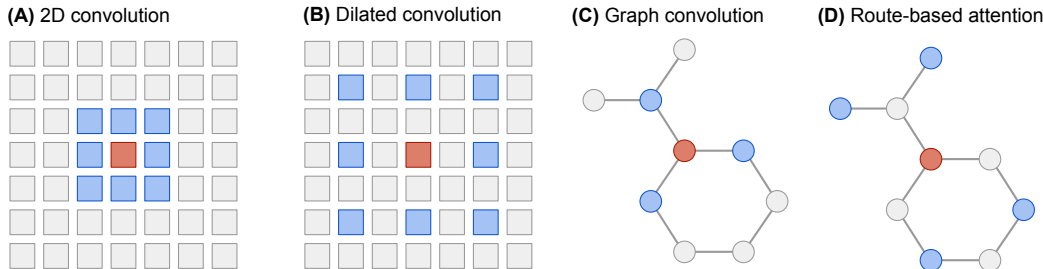
---

[*]Both authors contributed equally

Figure 1: Illustration of how route-based graph attention can mimic dilated filters in graphs. *Red* colored nodes are updated based on the vectors of the *blue* color nodes. (A) 2D convolutional filter, (B) 2D dilated filter, (C) graph convolution, (D) route-based graph attention.

attention to the neighbors (Veličković et al., 2017). One of the advantages of spatial methods over spectral methods is that they can be straightforwardly applied to problems involving multiple graphs.

However, the update step of a node in spatial methods only has access to its own neighborhood, hence limiting the information flow throughout the whole graph. Increasing the accessible neighborhood of each node can only be achieved at the expense of stacking several layers. The dilated filters in CNNs (Yu & Koltun, 2015) are an example of solving the neighborhood limitation in images, providing an efficient way to gather information over longer distances (see Figure 1, (A) and (B)).

In this work, we propose a method that flexibly aggregates information over longer distances in one step, analogously to dilated filters. Our *Graph Informer* approach is inspired by the sequence-to-sequence transformer model (Vaswani et al., 2017). The core contribution of our work is the introduction of *route-based multi-head self-attention* (RouteMHSA), which allows the attention mechanism to also access route information and, thus, base its attention scores both on the features of the nodes and the route between them. This enables Graph Informer to gather information from nodes that are not just direct neighbors. Furthermore, the proposed approach can straightforwardly use edge features, as they can be incorporated into the route features. In the case of chemical compounds, this allows using the annotation of the bonds between the atoms (single, double, aromatic, etc). The central idea is illustrated in Figure 1.

Our Graph Informer compares favorably with state-of-the-art graph-based neural networks on 13C NMR spectrum and drug bioactivity prediction tasks, which makes it suitable for both *node-level* and *graph-level* prediction tasks.

2 Proposed approach

2.1 Setup

In this work, we consider learning problems on graphs where both nodes and edges are labeled. For a graph G_i, we denote its node features by $V_i \in \mathbb{R}^{N_i \times F_{\text{nodes}}}$ and the node route information $P_i \in \mathbb{R}^{N_i \times N_i \times F_{\text{route}}}$, where the route information is computed using the adjacency matrix of the graph and the edge labels. For example, for chemical compounds we can include the distance information between the two nodes and type of connection (shortest path) between them (*e.g.*, whether it consists of only single bonds or aromatic bonds).

We thus propose a method that works for both (1) *graph-level* tasks, such as graph classification where the whole graph G_i is assigned a label and (2) *node-level* tasks where one should predict a value or a label for nodes (or subset of nodes) of the graph.

2.2 Route-based self-attention

Our approach is inspired by the encoder used in the transformer network (Vaswani et al., 2017), which proposed dot-product attention for sequence models formulated as

$$\text{Attn}(Q, K, V) = \text{Softmax}\left(\frac{1}{\sqrt{d_k}} Q K^{\top}\right) V,$$

where $Q, K \in \mathbb{R}^{N_i \times d_k}$ and $V \in \mathbb{R}^{N_i \times d_v}$ are the query, key, and value matrices. This attention can be used as *self-attention* by projecting the hidden vectors H into queries, keys, values (*i.e.*, $\text{Attn}(W_Q H, W_K H, W_V H)$). To encode sequences with dot-product attention, the transformer network adds position encodings to the hidden vectors H. In particular, sine waves with different frequencies and phases were used in the original paper to encode the position of the sequence elements. This allows the network to change attention probabilities based on the positions.

However, in the case of graphs, the global position of a node is not available. Even though one could use eigenvectors of the Laplacian of the graph in the case of a single given graph, this does not extend to multiple graphs. By contrast, we propose a stable addressing mechanism when working with multiple heterogeneous graphs. Our powerful but straightforward scheme allows the model not only to query based on distance, but also according to route features. Firstly, we map the hidden vectors H into a *route query* $Q_R = W_Q^{\text{route}} H$, where $Q_R \in \mathbb{R}^{N_i \times d_r}$ with d_r being the route query dimension. Similarly, we map the route features P into a *route key* $K_R = W_K^{\text{route}} P$, where $K_R \in \mathbb{R}^{N_i \times N_i \times d_r}$. The route information allows the attention mechanism to access the topology of the graph.

The route query and key can be now contracted into a $N_i \times N_i$ matrix of attention scores and added to the scores, after which Softmax is computed. Using the *einsum* convention[2], let

$$Q_R \otimes K_R = \text{einsum}_{\text{if,ijf}\rightarrow\text{ij}}(Q_R, K_R). \tag{1}$$

Now we can express the attention probabilities between the graph nodes as

$$A = \text{Softmax}\left(\frac{1}{\sqrt{d_k + d_r}}(QK^\top + Q_R \otimes K_R)\right), \tag{2}$$

where we have added also the size of the route key d_r to the normalization for keeping the values within trainable (non-plateau) ranges of the Softmax function. This enables the network to use both the information on the routes, as well as the node hidden vectors, to decide the attention probabilities.

Analogously, we project the route information into the value side. Firstly, we map the route data P into $V_R = W_V^{\text{route}} P$, where $V_R \in \mathbb{R}^{N_i \times N_i \times d_v}$. Finally, the route-based graph self-attention is given by

$$\text{RouteAttn}(Q, K, V, Q_R, K_R, V_R) = AV + \text{einsum}_{\text{ij,ijv}\rightarrow\text{iv}}(A, V_R), \tag{3}$$

where the attention probabilities $A \in \mathbb{R}^{N_i \times N_i}$ are defined as in Eq. 2.

2.3 Locality-constrained attention

The route-based attention defined in Eq. 3 is general and allows unconstrained access throughout the graph. In some applications, localized access is preferred and this can be easily forced by adding a masking matrix to the attention scores:

$$A_{\text{LC}} = \text{Softmax}\left(\frac{1}{\sqrt{d_k + d_r}}(QK^\top + Q_R \otimes K_R) + M_{\text{route}}\right), \tag{4}$$

where $M_{\text{route}} \in \mathbb{R}^{N_i \times N_i}$ is 0 for unmasked routes and minus infinity for masked connections (in practice, we use a large negative value). This allows us, for example, to mask all nodes whose shortest distance from the current node is larger than 3 or some other number, thereby creating an attention ball around the current node. Similarly, it is possible to use the mask M_{route} to create an attention shell (*i.e.*, attending nodes within a given range of shortest distances).

In our experiments, we only used the attention ball with different values for the radius.

2.4 Multi-head and mini-batching

Similarly to the transformer architecture, we use multiple heads. This means several route-based self-attentions are executed in parallel and their results are concatenated. This allows the network to attend to several neighbors at the same times.

---

[2]We follow the convention introduced by Numpy and used also in Pytorch packages.

Pool node   Additionally, we introduce a *pool* node that has a different embedding vector than the graph nodes. The pool node has no edges to the graph nodes, but is unmasked by M_{route}, so that the attention mechanism can always attend and read, if required. This idea has two motivations. First, it allows the information to be easily shared across the graph. Second, the pool node can be used as a "not found" answer when the attention mechanism does not find the queried node within the graph nodes.

Mini-batching   The proposed algorithm supports mini-batching via a few simple modifications. To introduce the batch dimension, the input tensors H and P must first be padded with zeros to have the same size, after which they are stacked. The linear projections Q, K, V, Q_R, K_R, V_R are now indexed by both batch (sample) and head. We replace all matrix multiplications by batched matrix products and in einsum we introduce leading batch and head dimensions. For the attention mechanism, we add an additional mask over the nodes, M_{node}, with large negative values for non-existing nodes (padded) for each sample:

$$A_{\text{LCB}} = \text{Softmax}\left(\frac{1}{\sqrt{d_k + d_r}}(QK^\top + Q_R \otimes K_R) + M_{\text{route}} + M_{\text{node}}\right). \tag{5}$$

Learnable module   Finally, we use the mini-batched attention probabilities A_{LCB} for RouteAttn from Eq. 3, which takes in (1) hidden vectors of the nodes H, with size $(B, N_{\text{nodes}}, N_{\text{hidden}})$, (2) route information P, with size $(B, N_{\text{nodes}}, N_{\text{nodes}}, F_{\text{route}})$, and (3) masks M_{node} and M_{route}. We use multiple heads and concatenate their results, giving us a route-based multi-head self-attention module (RouteMHSA):

$$\text{RouteMHSA} = \text{Concat}(X_1, \ldots, X_{N_{\text{head}}}) \tag{6}$$

$$\text{with } X_i = \text{RouteAttn}(W_{i,Q}H, W_{i,K}H, W_{i,V}H, W_{i,Q}^{\text{route}}H, W_{i,K}^{\text{route}}P, W_{i,V}^{\text{route}}P), \tag{7}$$

where W are the learnable parameters of the module.

The values for hidden size N_{hidden}, key size d_k, value size d_v, route key size d_r, and the number of heads N_{heads} can be chosen independently of each other. However, in our experiments we set N_{hidden} equal to $d_k N_{\text{heads}}$, with $N_{\text{heads}} \in \{6, 8\}$. We also use $d_v = d_k = d_r$.

2.5   Computational complexity

The computational complexity of the RouteMHSA is quadratic with respect to the number of nodes in the graph, because the attention mechanism computes all pairwise attention probabilities. For our drug-like compound application, the quadratic complexity is not an issue because such compounds have fewer than 100 atoms.

However, the complexity can be improved if the graph is sparsely connected. Because our method masks out connections that are too long, we can use sparse computations of the attention probabilities A_{LCB} instead computing a dense $N_{\text{node}} \times N_{\text{node}}$ matrix. This will lower the complexity from $O(N_{\text{node}}^2)$ to $O(N_{\text{node}}k)$, where k is the average number of nodes within each node's field of attention. Also, the masking tensor M_{route} needs to be replaced by a sparse counterpart. This can be done by creating a list of accessible nodes for each node. This makes it possible for Graph Informer to work on large graphs with sparse connectivity.

3   Architecture of the network

We explored two types of network architectures for the proposed route-based graph attention. In the first type, we simply combine a RouteMHSA with a node-wise dense linear layer of size N_{hidden} and add a ReLU nonlinearity (Nair & Hinton, 2010). This is the main building block for the network that we call *Feedforward Graph Informer*. We found that this network works well for node-level prediction tasks, as illustrated in Section 5.3.

The second type of network we explored was inspired by the architecture of transformer networks (Vaswani et al., 2017), where the output of the multi-head attention was fed through a linear layer and added to its input and then layer normalization (Ba et al., 2016) was applied. Additionally, a feedforward network (FFN) was applied to each hidden vector separately. However, in our graph
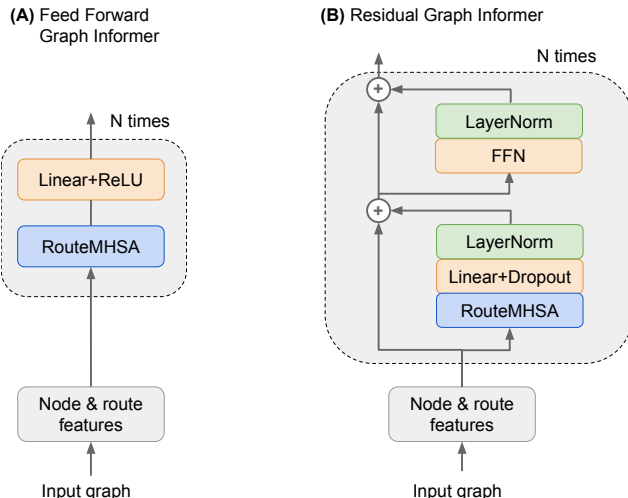
Figure 2: Layer setup for two architectures of Graph Informer. (A) Feedforward Graph Informer; (B) Residual Graph Informer.

experiments, both node-level and graph-level tasks, we found this architecture to be quite difficult to train (for more details, see Appendix D).

Instead, we found that creating a residual-style network (He et al., 2016) with LayerNorm solved the gradient flow issue and was easy to train. The layer for this architecture can be expressed as

$$T = H + \text{LayerNorm}(\text{Linear}(\text{RouteMHSA}(H))) \tag{8}$$

$$H' = T + \text{LayerNorm}(\text{FFN}(T)), \tag{9}$$

and H' is the output of the block (*i.e.*, updated hidden vectors). Empirically, we observed that in node-level tasks, the simple Feedforward Graph Informer outperformed the Residual Graph Informer. By contrast, the situation was reversed for graph-level tasks. Therefore, we use the feedforward architecture for the node-level experiments and the residual architecture for the graph-level experiments. The two proposed architectures are depicted in Figure 2.

4 Related research

There are several neural network approaches that can directly work on the graph inputs. The key component in many of the works is to propagate hidden vectors of the nodes H using the adjacency matrix[3] A, either by summing the neighbor vectors, AH, or by averaging them $\tilde{A}H$, where $\tilde{A} = A(D)^{-1}$ is the normalized adjacency matrix with D being the diagonal matrix of degrees. Neural Fingerprint (Duvenaud et al., 2015) proposed to use the propagation AH, combined with a node-wise linear layer and a nonlinearity to compute fingerprints for chemical compounds. Based on the degree of the node, Neural Fingerprint uses a different set of linear weights. Similarly, Graph Convolutional Networks (GCN, Kipf & Welling (2016)) propose to combine AH with a linear layer and a nonlinearity for semi-supervised learning, but by contrast with Neural Fingerprint, they use the same linear layer for all nodes, irrespective of their degree. Kearnes et al. (2016) proposed the Weave architecture, where they use embedding vectors also for the edges of the graph and the propagation step updates nodes and edges in a coupled way.

In a separate line of research, Scarselli et al. (2008) proposed Graph Neural Networks (GNN), which also use AH to propagate the hidden vectors between the nodes, but instead of passing the output to the next layer, the same layer is executed until convergence, which is known as the Almeida-Pineda algorithm (Almeida, 1987). Recently, (Li et al., 2015) proposed a GNN-based approach called Gated Graph Neural Networks (GGNN), where at each iteration the neighborhood information AH is fed into a GRU (Cho et al., 2014). An iterative graph network approach, motivated by the Message Passing algorithm in graphical model inference (Pearl, 2014) was proposed by Dai et al. (2016).

---

[3]The adjacency matrix here is assumed to include the self-connection (*i.e.*, $A_{ii} = 1$).

Hamilton et al. (2017) propose a representation learning method, GraphSAGE, that does not require supervised signals and inductively embeds nodes in large graphs.

The method closest to ours is Graph Attention Networks (GAT) proposed by Veličković et al. (2017). Similarly to us, they proposed to use multi-head attention to transform the hidden vectors of the graph nodes. However, in contrast with our approach, the integration of route features is not possible, which limits the attention to the neighbors of every node.

In a recent communication, Li et al. (2019) proposed a method different from previous works by learning a transformation from a source graph to a target graph. To this end, they used GCN-like convolutional layers without attention to propagate the hidden vectors within the graph. To propagate between graphs, they used the attention mechanism proposed for transformer networks (Vaswani et al., 2017), which solely relies on node features.

In contrast to the spatial methods reviewed above, spectral methods use the graph Fourier basis. One of the first spectral approaches to introduce this concept in the context of neural networks was SCNN (Bruna et al., 2013). To avoid the costly eigenvalue decomposition, Defferrard et al. (2016) used Chebyshev polynomials to define localized filters. Based on this work, Li et al. (2018) proposed a neural network that combines spectral methods with metric learning for graph classification.

5 Evaluation

We compare our method against several baselines on two tasks. We first consider a **node-level** task where, given the graph of the compound, the goal is to predict 13C NMR peaks for all of the carbon atoms, see Section 5.3. For the second task, we consider the drug–protein activity data set ChEMBL (Bento et al., 2014), which is a **graph-level** task. This task is covered in Section 5.4. In both tasks and for all methods we used the same feature set (See Appendix A). For methods that could not use route features but supported edge features, we used the bond type (single, double, triple, aromatic).

Each method was executed using one NVIDIA P100 with 16GB RAM with the data fed using 9 Intel(R) Xeon(R) 6140 Gold cores. For the implementation of our algorithm we used Pytorch. [4]

5.1 Baselines

GGNN We used the Pytorch implementation of Gated Graph Neural Networks[5], which was designed for node-level prediction on graphs. We also modified it to work on graph-level tasks by adding the same graph pooling layer used in our network (global average pooling). For both tasks, we used 5 propagation steps.

NeuralFP We implemented the Neural Fingerprint in Pytorch. The implementation follows the original method using softmax graph pooling (Duvenaud et al., 2015). We also extended the method to work on node-level tasks by removing the pooling and feeding the last layer hidden vectors to the regression output layer (see Section 5.3).

GCN For the Graph Convolutional Networks, we adapted the Pytorch implementation[6] to support mini-batched training, which we label **GCN-sum**. We also adapted the code to support graph classification by incorporating the same graph pooling layer used in our network (global average pooling). Additionally, we implemented averaging GCN (**GCN-avg**), where the adjacency matrix A is replaced by its normalized version.

Weave For the Weave network, we used the Deepchem implementation[7]. The implementation only supports graph-level predictions.

GAT We used the Tensorflow implementation of Graph Attention Networks[8]. The implementation only supports node-level prediction.

---

[4]The code is available here: `https://github.com/jaak-s/graphinformer`.

[5]Available from `https://github.com/JamesChuanggg/ggnn.pytorch/`.

[6]Available from: `https://github.com/tkipf/pygcn.git`.

[7]Available here: `https://github.com/deepchem/deepchem/`.

[8]Available here: `https://github.com/PetarV-/GAT`

Table 1: Results for NMR 13C spectrum prediction for models with different number of layers. We report the mean and standard deviation for the test MAE and the median size (#params column) of the best model over the repeats. The bolded entries are best or statistically not different from the best.

Method	#layers	Test MAE	#params
Mean model		46.894	1
GCN-avg	2	6.284 ± 0.053	0.2M
GCN-avg	3	8.195 ± 0.385	0.3M
GCN-sum	2	5.015 ± 0.082	0.2M
GCN-sum	3	6.275 ± 0.306	0.1M
GAT	1	2.889 ± 0.021	0.2M
GAT	2	3.148 ± 0.067	0.2M
GAT	3	5.193 ± 0.196	0.2M
GGNN	na	1.726 ± 0.055	0.2M
NeuralFP	1	1.450 ± 0.067	0.3M
NeuralFP	2	0.851 ± 0.038	1.3M
NeuralFP	3	0.742 ± 0.045	2.3M
NeuralFP	4	0.728 ± 0.028	0.5M
FF Graph Informer	1	0.701 ± 0.035	0.4M
FF Graph Informer	2	0.541 ± 0.027	0.3M
FF Graph Informer	3	$\mathbf{0.506} \pm 0.024$	1.9M
FF Graph Informer	4	$\mathbf{0.514} \pm 0.022$	0.7M

5.2 Model selection

All methods used the validation set to choose the optimal hidden size from $\{96, 144, 192, 384\}$ and dropout from $\{0.0, 0.1\}$ (Srivastava et al., 2014). For our method attention radius form $\{1, 2, 3, 4\}$ and head count from $\{6, 8\}$ were also selected. All methods trained the methods with the Adam optimizer (Kingma & Ba, 2014). The validation set was used for early stopping (*i.e.*, we execute all methods for 100 epochs and picked the best model). At epochs 40 and 70, the learning rate was decreased by a factor of 0.3. Finally, the best early stopped model was used to measure the performance on the test set.

5.3 Node-level task: NMR 13C spectrum prediction

13C NMR is a cost-efficient spectroscopic technique in organic chemistry for molecular structure identification and structure elucidation. If 13C NMR measurements can be combined with a highly accurate prediction, based on the graph (structure) of the molecule, it is possible to validate the structure of the sample. It is crucial to have highly accurate predictions, because in many cases several atoms have peaks that are very close to each other. If such predictions are available, the chemist can avoid running additional expensive and time consuming experiments for structure validation.

To create the data set, we used 13C NMR spectra from NMRShiftDB2[9]. We kept only molecules where all carbon atoms where assigned with a 13C peak position and removed any compounds that failed RDKit (Landrum et al., 2019) sanitization or Gasteiger partial charge calculation. This resulted in 25,645 molecules. The peak positions are measured in parts per million (ppm) and ranges from approximately -10 to 250 ppm[10]. In our data set, the mean and standard deviation are 95 and 52 ppm.

In accordance to the most common Lorentzian peak shape assumption in NMR spectroscopy, we minimize the mean absolute error (MAE) (Gunther, 1980). For evaluation, we split the data into train-validation-test sets containing 20,000, 2,500, and 3,145 molecules, respectively, and report results over 3 repeats with different splits. All methods use the same node-level output head, consisting of a `tanh` layer followed by a linear output layer. The MAE loss is minimized only for the atoms that have measurements for the 13C peaks (*i.e.*, the carbon atoms). For the results, see Table 1.

---

[9]Available here: `https://nmrshiftdb.nmr.uni-koeln.de/`

[10]The ppm value measures the shift of the peak relative to a specific reference carbon atom.

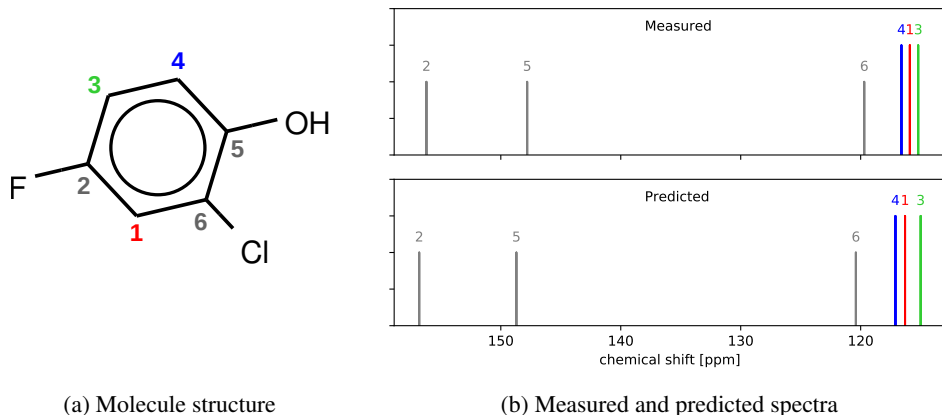(a) Molecule structure   (b) Measured and predicted spectra

Figure 3: 13C NMR spectrum for 2-chloro-4-fluorophenol. The predicted NMR peaks are in the same order, even in the densely packed region (carbons 4, 1, 3).

Table 2: Results for ChEMBL drug–protein activity prediction. We report the mean and standard deviation for the test AUC-ROC and the median size (#params) of the best model over the repetitions. The bolded entries are best or statistically not different from the best.

Method	#Layers	AUC-ROC (test)	#params
GCN-sum	2	0.816 ± 0.004	0.1M
GCN-sum	3	0.807 ± 0.005	0.8M
GGNN	*na*	0.756 ± 0.014	0.2M
NeuralFP	2	0.819 ± 0.003	0.4M
NeuralFP	3	0.826 ± 0.005	0.4M
Weave	2	0.830 ± 0.006	0.7M
Weave	3	0.791 ± 0.007	0.7M
Resid. Graph Informer	2	$\mathbf{0.840} \pm 0.003$	0.5M
Resid. Graph Informer	3	$\mathbf{0.839} \pm 0.003$	0.7M

Our proposed Graph Informer reaches 0.5 MAE, which is at the limit where even molecules with densely packed 13C NMR spectra can be resolved. Densely packed spectra are common for aromatic ring systems, which are prevalent in drug-like compounds. As an example, see Figure 3 depicting 2-chloro-4-fluorophenol, which has three carbons, labelled 1, 3, 4 with peaks within the 115.2 to 116.6 ppm range. Figure 3b displays the predictions from the molecule when it was in the test set. The predictions match closely the true spectrum and, critically, predict the true ordering, which allows us to perfectly match peaks. For more details and the table with exact prediction values, see Appendix C.

5.4 Results for graph-level task: ChEMBL

Prediction of drug–target interaction is central to modern drug discovery, because it allows prioritizing candidate compounds and avoiding toxicity. We extracted molecule–protein interaction data from the ChEMBL v23 database (Bento et al., 2014), see Appendix B. The extracted data set contains 94 binary classification tasks on 87,415 molecules, for a total of 349,192 activity points. Because of the well-known series effect in drug design, allocating molecules randomly to the training and test sets results in overly optimistic and inconsistent performance estimates (Simm et al., 2018) because highly similar molecules end up in both training and test sets. Therefore, we apply *clustered* hold-out validation (See Appendix B.3). We allocate the clusters (rather than individual molecules) to the training (60%), validation (20%), and test sets (20%). We repeat this split 5 times.

Our and all adapted methods (GCN-sum and GGNN) use a graph pooling layer consisting of a linear layer, ReLU, average pooling, and a linear layer. We minimize the cross-entropy loss and Table 2 reports the AUC-ROC. Our method outperformed all baselines in both the 2 and 3 layer settings.

6   Conclusion

In this work, we proposed a route-based multi-head attention mechanism, inspired by the transformer network, which can leverage features of node pairs. Thanks to the route features our approach can incorporate information from non-direct neighbors efficiently—similarly to the use of dilated filters in CNNs. Our empirical evaluation demonstrates that the proposed method is suitable both for node-level and graph-level prediction tasks, and delivers significant improvements over existing approaches in 13C NMR spectrum and drug bioactivity prediction.

References

Almeida, L. B. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proceedings, 1st First International Conference on Neural Networks*, volume 2, pp. 609–618. IEEE, 1987.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016. URL `http://arxiv.org/abs/1607.06450`. arXiv: 1607.06450.

Bento, A. P., Gaulton, A., Hersey, A., Bellis, L. J., Chambers, J., Davies, M., Krüger, F. A., Light, Y., Mak, L., McGlinchey, S., et al. The chembl bioactivity database: an update. *Nucleic acids research*, 42(D1): D1083–D1090, 2014.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL `http://arxiv.org/abs/1406.1078`.

Dai, H., Dai, B., and Song, L. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pp. 2702–2711, 2016.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016.

Doddrell, D., Pegg, D., and Bendall, M. R. Distortionless enhancement of nmr signals by polarization transfer. *Journal of Magnetic Resonance (1969)*, 48(2):323–327, 1982.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Gunther, H. *NMR spectroscopy: an introduction*, volume 81. Wiley Chichester, 1980.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Hartigan, J. A. *Clustering Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 99th edition, 1975. ISBN 047135645X.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

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

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Landrum, G., Kelley, B., Tosco, P., sriniker, gedeck, Schneider, N., Vianello, R., Dalke, A., Savelyev, A., Turk, S., Swain, M., Cole, B., Vaucher, A., Wójcikowski, M., Pahl, A., JP, strets123, JLVarjo, Fuller, P., Gavid, D., O'Boyle, N., Cosgrove, D., Sforna, G., Nowotka, M., pzc, van Santen, J., Jensen, J. H., Hall, D., N, D., and Avery, P. RDKit: Open-source cheminformatics, 2019. URL `http://www.rdkit.org`.

Li, R., Wang, S., Zhu, F., and Huang, J. Adaptive graph convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

Li, Y., Liang, X., Hu, Z., Chen, Y., and Xing, E. P. Graph transformer, 2019. URL `https://openreview.net/forum?id=HJei-2RcK7`.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Pearl, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Simm, J., Klambauer, G., Arany, A., Steijaert, M., Wegner, J. K., Gustin, E., Chupakhin, V., Chong, Y. T., Vialard, J., Buijnsters, P., et al. Repurposing high-throughput image assays enables biological activity prediction for drug discovery. *Cell chemical biology*, 25(5):611–618, 2018.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is All you Need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Yu, F. and Koltun, V. Multi-Scale Context Aggregation by Dilated Convolutions. *arXiv:1511.07122 [cs]*, November 2015. URL `http://arxiv.org/abs/1511.07122`. arXiv: 1511.07122.

A   Node and route features

In addition to the following features, we fed the one-hot encoding of the atom types by concatenating it to the node feature vector, see Table 3. The atom types occurred in the two data sets are {C, N, O, Cl, F, S, I, Br, P, B, Zn, Si, Li, Na, Mg, K}. As the route features (Table 4) contain all edge labels (single, double, triple and aromatic bond type), all information about the graph topology is retained.

Table 3: Node features

Position	Description
0-2	Formal charge, one-hot encoded {-1, 0, +1}
3-7	Hybridization state, one-hot encoded {s, sp, sp2, sp3}
8-13	Explicit valence, one-hot encoded integer, between 0 and 5
14	Aromaticity, binary
15	Whether it is in a ring size 3, binary
16	Whether it is in a ring size 4, binary
17	Whether it is in a ring size 5, binary
18	Whether it is in a ring size 6, binary
19	Whether it is in any ring, binary
20	Partial charge, computed by Gasteiger method, real number
21	Whether it is a H-acceptor, binary
22	Whether it is a H-donor, binary
23	Whether it is an R stereo center, binary
22	Whether it is an S stereo center, binary

Table 4: Route features

Position	Description
0-8	Bond distance, binned, $[0, 1, 2, 3, 4, 5 \leq d \leq 6, 7 \leq d \leq 8, 9 \leq d \leq 12, 13 \leq d\,]$
9	Whether the shortest pure conjugated path containing at most 4 bonds, binary
10	Whether the shortest pure conjugated path containing at least 5 bonds, binary
11	Whether there is a route containing only up to 13 single bonds, binary
12	Whether there is a route containing only up to 13 double bonds, binary
13	Triple bond, binary
14	Whether there is a shortest path which is a pure conjugated path, binary
15	Whether the endpoints are in the same ring, binary
16	Single bond, binary
17	Double bond, binary
18	Aromatic bond, binary

B   Preprocessing of the ChEMBL data set

B.1   Extracting human drug–protein interaction data

We extracted IC50 values for *Homo sapiens* protein targets from ChEMBL version 23 SQLite release (Bento et al., 2014) using the following SQL query, and computed the negative log10 of IC50 in nM, which we refer to as pIC50. Extreme values that cannot correspond to meaningful real measurement results are treated as database errors and filtered out ($IC50 < 10^{-5}$ or $10^9 \leq IC50$).

```
SELECT molecule_dictionary.chembl_id as cmpd_id,
    target_dictionary.chembl_id as target_id,
CASE activities.standard_type
    WHEN 'IC50' THEN activities.standard_value
    WHEN 'ug.mL-1' THEN
        activities.standard_value / compound_properties.full_mwt * 1E6
    END ic50,
CASE activities.standard_relation
```

```
      WHEN '<'  THEN '<'
      WHEN '<=' THEN '<'
      WHEN '='  THEN '='
      WHEN '>'  THEN '>'
      WHEN '>='  THEN '>'
      ELSE 'drop' END relation
FROM molecule_dictionary
JOIN activities on activities.molregno == molecule_dictionary.molregno
JOIN assays on assays.assay_id == activities.assay_id
JOIN target_dictionary on target_dictionary.tid == assays.tid
JOIN compound_properties on
   compound_properties.molregno = molecule_dictionary.molregno
WHERE target_dictionary.organism='Homo sapiens' AND
   target_dictionary.target_type='SINGLE PROTEIN' AND
   activities.standard_type = 'IC50' AND
   activities.standard_units IN  ('nM','ug.mL-1') and
   relation != 'drop' and
   ic50 < 10e9 AND
   ic50 >= 10e-5
```

B.2   Filtering

First we sanitized the molecules by removing inactive ions (Cl-, Na+, K+, Br-, I-) from the structure. Then, we removed all non-drug-like molecules containing heavy metals or more than 80 non-hydrogen atoms. We thresholded the activity for each protein with four pIC50 thresholds (5.5, 6.5, 7.5, and 8.5) taking into account the *relation* column. We then kept protein–threshold pairs that had measurements at least on 2,500 compounds containing at least 200 positive and 200 negative samples. This filtering process resulted in 94 protein–threshold pairs as binary classification tasks on 87,415 molecules. The matrix contains 349,192 activity points.

B.3   Clustered hold-out validation

We clustered the molecules with similarity threshold 0.6 Tanimoto on ECFP6 features computed using RDKit (Landrum et al., 2019). For clustering, we used the leader-follower method (Hartigan, 1975). We then assigned each cluster to the training, test, or validation sets randomly. We adapted the clustered validation pipeline from Simm et al. (2018).

C   Description of the NMR application

In molecules, the chemical environment of the atomic nuclei has an influence on their measured resonance frequency, the NMR peak position. This influence is called the chemical shift, and measured in parts per million (ppm). The nominal value of this shift is independent from the measurement equipment and can be used to compare chemical environments between molecules measured in different laboratories.

We are specifically interested in the following scenario. An organic chemist intends to synthesize *compound A*. After the synthesis and purification steps, the identity of the product needs to be validated. The chemist can measure the 13C NMR spectrum of the product, and compare it to the predicted spectrum from the structure of *compound A*. To compare the two spectra, the correct peaks corresponding to the same atom need to be aligned. There is no way to tell from a 13C NMR spectrum alone which peak corresponds to which atom. Therefore, wrongly ordered peaks in the prediction would result in an erroneous assignment.

As in the example in the main text, we use the 13C NMR spectrum of the compound 2-chloro-4-fluorophenol to illustrate the assignment problem. The measured and predicted chemical shifts are reported in Table 5. As can be seen in Figure 3, the three unsubstituted aromatic carbons (1,3,4) have very close peaks in the 115.2 to 116.6 ppm region. For the sake of completeness note that the intensity of the peaks (value of the y-axis) in the plot are directly related to the number of direct hydrogen neighbors of the carbon, and therefore does not need to be predicted (Doddrell et al., 1982).

Table 5: 13C NMR spectrum assignment of 2-chloro-4-fluorophenol.

Carbon	2	5	6	4	1	3
Measured	156.2	147.8	119.7	116.6	115.9	115.2
Predicted	156.8	148.7	120.4	117.1	116.3	115.0
Abs. Error	0.6	0.9	0.7	0.5	0.4	0.2

D   Improving the convergence of Residual Graph Informer

Precisely following the strategy used for transformer networks (Vaswani et al., 2017), we get the following architecture for graphs:

$$T = \mathrm{LayerNorm}(H + \mathrm{Linear}(\mathrm{RouteMHSA}(H))) \tag{10}$$

$$H' = \mathrm{LayerNorm}(T + \mathrm{FFN}(T)), \tag{11}$$

$$\text{where } \mathrm{FFN}(x) = W_2 \mathrm{ReLU}(W_1 x + b_1) + b_2, \tag{12}$$

and H' is the output of the block (*i.e.,* updated hidden vectors).

In our experiments, we observed that the output of $\mathrm{RouteMHSA}(H)$ dominates the input H in Eq. 10. This causes convergence issues as normalizing this sum by the $\mathrm{LayerNorm}$ causes the gradient through the input H branch to vanish. This was the main motivation for introducing the pure-residual style network described in Section 3.