Neural Reasoning for Chemical-Chemical Interaction

Trang Pham, Truyen Tran, and Svetha Venkatesh
Applied AI Institute, Deakin University, Geelong, Australia
{phtra,truyen.tran,svetha.venkatesh}@deakin.edu.au

Abstract

We present an effective solution for predicting chemical-chemical interaction (CCI) between compounds. We observed that reasoning about CCI is akin to question-answering (QA) about a set of graph-structured objects. That is, the problem admits the form `(query, {set of graphs}, ?)`, where the query can include environmental factors, the graphs are basic molecular structures, and the answer is the degree of interaction between molecules. To solve the CCI problem, we transform it into an instance of learning to reason through a new end-to-end neural architecture dubbed Relational Dynamic Memory Networks (RDMN). This is a generic differentiable neural computer capable of learning to answering queries about a set of graph-structured objects without explicit programming. Experiments are run on a large public CCI database known as STITCH, demonstrating the state-of-the-art performance of the proposed solution.

1 Introduction

We address the problem of learning to predict interaction between chemical compounds, given only the basic molecular information. This fundamental task paves the way for predicting a variety of chemical and biological phenomena, including drugs binding, toxicity, effects of combination therapies, and biological functions [12]. Our main insight is that the reasoning about chemical-chemical interaction (CCI) is akin to question-answering (QA) about a set of graph-structured objects. For example, a query can be environmental conditions, the objects are compounds represented as molecular graphs, and the answer is the degree of interaction between compounds.

Inspired by recent successes of neural networks in textual and visual QA (e.g., see [22]) and in predicting molecular bioactivities (e.g., see [15]), we transform the CCI problem into an instance of neural reasoning. Unlike classical symbolic reasoning which involves discrete symbol manipulation through logical inference [13], neural reasoning involves continuous tensor manipulation through real transformation [7]. In particular, we leverage our recent neural architecture known as Graph Memory Networks (GMN) [15] for the task. A GMN is a feedforward neural network augmented with a graph-structured working memory[1] module. For a given query and a graph, GMN loads the graph into memory, then iteratively refines the answers by consulting/updating the memory until the final answer is found. The entire reasoning process is learned from data rather than designed by hand, making it suitable for domains where there is rich training data of the form `(query, graph, answer)`. For example, in the previous application of GMN to predict bioactivity of a compound, the query can include a particular drug target (e.g., a protein or a disease) and other environmental factors, the graph is the compound molecular graph, and the answer is whether an activity occurs (e.g., a protein binding, or a response).

While GMN can answer queries about a graph, it is not readily suitable for modeling chemical-chemical interaction, which involves multiple molecular graphs. For that we generalize GMN to handle interaction between multiple graphs. The interaction is realized through an iterative

[1] Working memory is a cognitive faculty for temporarily making information available for reasoning [1].

NeurIPS 2018 Workshop on Machine Learning for Molecules and Materials, Montréal, Canada.

process of exchanging messages between graphs. The generalized architecture thus has multiple memory components coordinated by a controller to solve problems of the form $(\texttt{query}, \{\texttt{graph}_1, \texttt{graph}_2, \ldots, \texttt{graph}_n\}, \texttt{?})$. We rename the model *Relational Dynamic Memory Networks* (RDMN)[2].

In what follows, we describe the RDMN and empirically demonstrate its efficacy on chemical-chemical interaction data. In particular, our experiments are run on the STITCH database [10] (Search Tool for InTeractions of CHemicals) consisting of approximately 1M chemical interactions between over 68K chemicals. The molecules are extracted from the PubChem database using RDKit. Comparing against the latest CCI model known as DeepCCI, RDMN achieves higher accuracy.

2 Methods

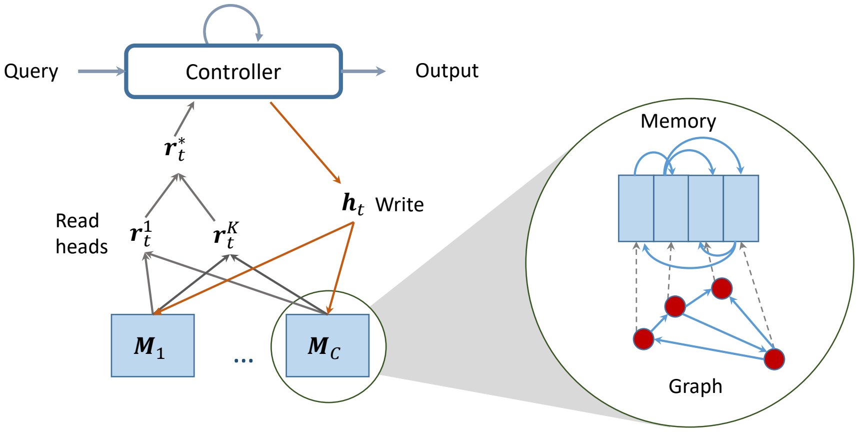


Figure 1: Relational Dynamic Memory Networks (RDMN). There are multiple structured memory components $\{M_c\}_{c=1}^{C}$, each of which is associated with an input graph. When a query is presented, the controller reads the query and intializes the memory. For each graph, each node is embedded into a memory cell. Then during the reasoning process, the controller iteratively reads from and writes to the memory. Finally, the controller emits the output.

We now present RDMN and how it can be used for reasoning about chemical-chemical interaction. Abstractly, RDMN is designed to solve problems of the form $(\texttt{query}, \{\texttt{graph}_1, \texttt{graph}_2, \ldots, \texttt{graph}_C\}, \texttt{?})$, that is, answering an arbitrary query about a set of graphs.

This is a natural fit for CCI between two or more compounds, each of which is associated with a molecular graph whose nodes are atoms and edges are bond types. Nodes have associated information, including the atom ID and the number of linked hydrogen. Here the queries can be environmental conditions (e.g., temperature, pressure, catalysts) or other externally extracted information from the set of compounds. The answer for CCI is the degree of interaction between the compounds.

RDMN is a neural computer, similar to Neural Turing Machines (NTM) [5, 6], thus consisting of a CPU-like controller and a RAM-like memory. The controller is a recurrent neural network responsible for memory manipulation. Both NTMs and RDMN are fully differentiable allowing them to be trained end-to-end using gradient-based techniques without explicit programming. However, unlike NTMs which have flat memory and thus make no assumption about the data structure other than the sequential arrival, RDMN has explicitly structured memory suitable for graph data. In RDMN, the memory module can have one or multiple components (Fig. 1), each of which is a matrix, whose columns are memory cells, each of which storing one piece of information. Cells are inter-linked, where the links are dynamically defined by the structure of the data.

[2] See [16] for a complete description.

Presented with graphs input and a query, RDMN initiates a reasoning episode, during which the machine dynamically allocates memory to hold the graphs, with one graph per memory component. The memory structure of a component reflects the corresponding graph. Then the controller takes the query as the input and repeatedly reads from the memory, processes and sends the signals back to the memory cells. Then at each reasoning step, the cell content is updated by the signals from the controller and its neighbor memory cells in the previous step. Throughout the reasoning episode, the memory cells evolve from the original input to a refined stage, preparing the controller for generating the output. The interaction between memory components is mediated through message exchanging with the controller. For C memory components, the system, when rolled out during reasoning, consists of $C + 1$ recurrent *matrix* neural networks interacting with each other [4].

In what follows we sketch the overall operations of RDMN, leaving the details in Appendix A.

2.1 Encoder

The encoder prepares data for the reasoning process, i.e., encoding queries into the controller and data into the memory. The query q is first encoded into a vector as:

$$\boldsymbol{q} = \text{q\_encode}\left(q\right) \tag{1}$$

Let X_c be a graph-structured representation of object c, for $c = 1, 2, ..., C$. The representation X_c is loaded into a memory matrix $\boldsymbol{M}_c$:

$$\left(\boldsymbol{M}_c, \boldsymbol{A}_c\right) = \text{m\_load}\left(X_c, q\right) \tag{2}$$

where $\boldsymbol{A}_c$ is relational structure between memory slots. The dependency on the query q offers flexibility to define query-specific memory, which could be more economical in cases where the query provides constraints on the relevant parts of the object.

Assume that the query and object c together impose a set $\mathcal{R}_c$ of pairwise relations between memory cells. More precisely, any pair of cells will have zero or more relations draw from the relation set. For example, for c is a molecule, then $\boldsymbol{M}_c[i]$ can be the (learnable) embedding of the atom at node i of the molecular graph, and the relations can be bonding types (e.g., ion or valence bonds). The relations can be modeled as a collection of adjacency matrices, that is, $\boldsymbol{A}_c = \left\{A_{c,r}\right\}_{r \in \mathcal{R}_c}$.

Given $\boldsymbol{q}$ and $\boldsymbol{M} = \left\{\boldsymbol{M}_1, \boldsymbol{M}_2, ..., \boldsymbol{M}_C\right\}$, the controller initiates a reasoning episode to compute the output probability $P_\theta\left(\boldsymbol{y} \mid \boldsymbol{M}, \boldsymbol{q}\right)$. In what follows, we briefly present how the computation occurs.

2.2 Reasoning processes

The controller manages the reasoning process by updating its own state and the working memory. Let $\boldsymbol{h}_t$ be the state of the controller at time t $(t = 0, ..., T)$. At time $t = 0$, the state is initialized as $\boldsymbol{h}_0 = \boldsymbol{q}$. At $t = 1, 2, ..., T$, the controller retrieves a content $\boldsymbol{r}_t^*$ from the memory at time t as follows:

$$\boldsymbol{r}_{c,t}^k = \text{m\_read}_k\left(\boldsymbol{M}_{c,t-1}, \boldsymbol{h}_{t-1}\right) \tag{3}$$

$$\bar{\boldsymbol{r}}_t^k = \text{r\_aggregate}\left\{\boldsymbol{r}_{c,t}^k\right\}_{c=1}^C \tag{4}$$

$$\boldsymbol{r}_t^* = \text{r\_combine}\left\{\bar{\boldsymbol{r}}_t^k\right\}_{k=1}^K \tag{5}$$

where $k = 1, 2, ..., K$ denotes the index of read head. The use of multiple read heads accounts for multiple pieces of information that may be relevant at any step t. The r_aggregate$\{\cdot\}$ function operates on the variable-size set of retrieved memory vectors and combines them into one vector. The r_combine$\{\cdot\}$ function operates on the fixed-size set of vectors returned by the read-heads.

Once the content $\boldsymbol{r}_t^*$ has been read, the controller state is updated as:

$$\boldsymbol{h}_t = \text{s\_update}\left(\boldsymbol{h}_{t-1}, \boldsymbol{r}_t^*\right) \tag{6}$$

This initiates an update of the memory:

$$\boldsymbol{M}_{c,t} = \text{m\_update}\left(\boldsymbol{M}_{c,t-1}, \boldsymbol{h}_t, \boldsymbol{A}_c\right) \tag{7}$$

for $c = 1, 2, ..., C$.

2.3 Decoder

At the end of the reasoning process, the controller predicts an output, that is

$$\boldsymbol{y} = \text{decode}\left(\boldsymbol{h}_T, \boldsymbol{q}\right) \tag{8}$$

The decoder is a task-specific model, which could be a deep feedforward net for vector output, or a RNN for sequence output.

2.4 Training

Given a training set, in which each data instance has the form `(query,` $\{$`graph`$_1$`,graph`$_2$`,...,graph`$_C\}$`, answer)`, RDMN is trained in a supervised fashion from end-to-end, i.e., by minimizing a loss function. The typical loss is $-\log P\left(\boldsymbol{y} \mid q, \{X_c\}_{c=1}^{C}\right)$. As the entire system is differentiable, gradient-based techniques can be applied for parameter optimization, e.g., [9].

3 Experiments

3.1 Datasets

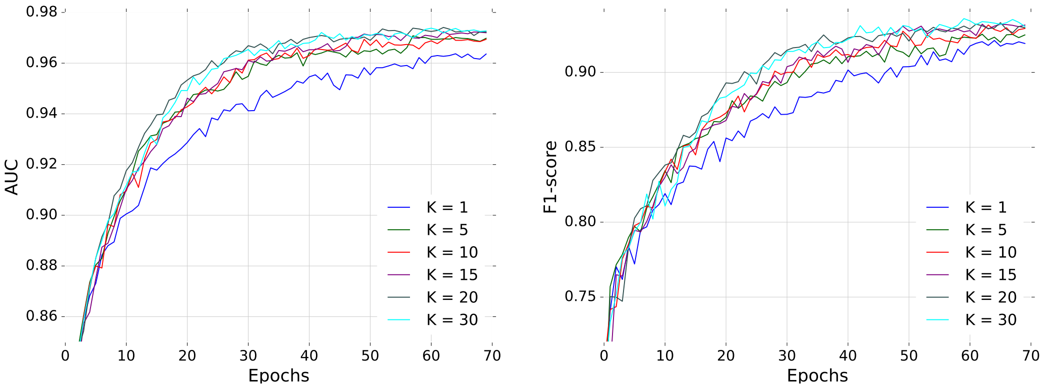
We conducted experiments on chemical-chemical interaction data downloaded from the STITCH database [10] (Search Tool for InTeractions of CHemicals), a network of nearly 1M chemical interactions for over 68K different chemicals. Each interaction between two chemicals has confidence score from 0 to 999. Following [11, 12], we extracted *positive-900* (11,764 examples) and *positive-800* (92,998) from interactions with confidence scores more than 900 and 800, respectively and extract *negative-0* (425,482 samples) from interactions with confidence scores equal to zero. We then created the CCI900 dataset from all the positive samples in positive-900 and the same number of negative samples randomly drawn from negative-0. CCI800 was also created similarly. Therefore, two datasets used for the experiments - CCI900 and CCI800 have 23,528 and 185,990 samples, respectively. The molecules were downloaded from the PubChem database using CID (Compound ID) shown in STITCH. The tool RDKit[3] was used to extract graph structures, atom and bond features, fingerprints and SMILES of the molecules.

3.2 Experiment settings

We designed experiments on three different types of representations of molecules: (i) fingerprint features, (ii) SMILES (Simplified Molecular-Input Line-Entry System) - a string format to describe the structures of molecules, and (iii) the graph structure information of the molecules.

Baselines Fingerprint feature vectors are processed by Random Forests and Highway Networks [18], which are two strong classifiers in many applications [14]. Each data point consists of two fingerprint vectors representing two molecules. We average the two vectors as the input vector for the two baselines. SMILES strings are modeled by DeepCCI [12], a recent deep neural network model for CCI. Each SMILES is represented by a matrix where each row is a one-hot vector of a character. The matrix is then passed through a convolution layer to learn the hidden representation for each SMILE string. The two hidden vectors are then summed and passed through a deep feedforward net to learn the final representation of the interaction between the two molecules. We tune the hyperparameters for DeepCCI as suggested by the authors [12].

[3] http://www.rdkit.org/



(a)

Figure 2: The performance of RDMN with different number of reading heads K during training, reported in (a) AUC and (b) F1-score. Best viewed in color.

Model setting We also conducted two other experiments to examine the effectiveness of *using side information as the query*. The attention read, which is a weighted sum of all memory cells at the controller, might not properly capture the global information of the graph while fingerprint feature vectors or SMILES strings attain this information. Here, we set two types of vectors as the queries: (i) the mean of two fingerprint vectors and (ii) the hidden representation generated by DeepCCI. For the latter setting, DeepCCI parameters are randomly initialized and jointly learned with our model's parameters.

3.3 Results

The effect of multiple attentions In CCI, substructures from different molecules interact, leading to multiple substructure interactions. In our model, we expect that each attention reading head is able to capture a specific interaction of substructures. Hence, an appropriate number of attention heads can capture all substructure interactions and provide more abundant information about the graph interaction, which may improve the prediction performance. Here, we evaluated the improvement in prediction brought by the number of attention heads K. Taking RDMN without the side information query, we varied K by 1, 5, 10, 15, 20 and 30, and trained the resulting six models independently on the CCI900 dataset. Fig. 2 reports the performance changes during training for different K. We can see that when $K > 1$, increasing K only slightly improves the performance while there is a bigger gap between the performance of $K = 1$ and $K > 1$. There is not much difference when K is increased from 20 to 30. It is possible that when the number of attention heads is large, they collect similar information from the graphs, leading to saturation in performance.

The effect of the side information as query While the neighborhood aggregation operation in our model can effectively capture the substructure information, the weighted sum of all memory cells in the attention mechanism might not properly collect the global information of the whole graph. Hence, using the side information containing the global information such as Fingerprints or SMILES strings might help the training. We have shown in Table 1 that using fingerprint vectors or the hidden state generated by DeepCCI as the query for our model can improve the performance on the both datasets. We also found that even though using side information query increases the number of training parameters, it can prevent overfitting. Fig. 3 shows the loss curves on the training and the validation set of RDMN with a single attention head when the query is set as constant and when the query is the hidden stated produced by DeepCCI model (RDMN+SMILES). We can see from the figure that the validation loss of RDMN starts to rise quickly after 40 epochs while the validation loss of RDMN+SMILES still remains after 100 epochs.

Comparative results Table 1 reports the performance of the baselines and our proposed model in different settings on the two dataset CCI900 and CCI800 reported in AUC and F1-score. SMILES features with DeepCCI outperformed the fingerprint features with Highway Networks by 3.8% F1-

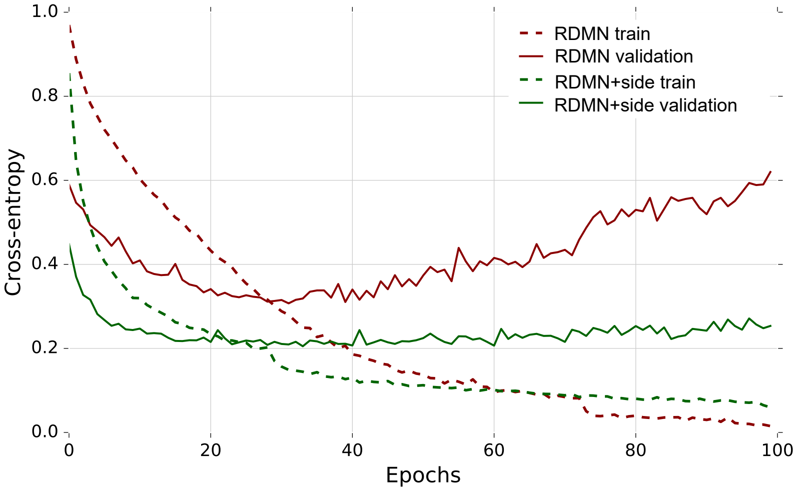


Figure 3: Training and validation losses of RDMN with and without side information during training. Best viewed in color.

score on CCI900 and by 3.6% F1-score on CCI800. Our model with a single attention head is slightly better than DeepCCI on both datasets. Interestingly, using multiple attention heads and side information (fingerprint and SMILES features) as the query both help to improve the performance. For CCI900, RDMN with SMILES query improves 1.7% F1-score on single attention setting and 1.2% on multiple attention settings.

	CCI900		CCI800	
	AUC	F1-score	AUC	F1-score
Random Forests	94.3	86.4	98.2	94.1
Highway Networks	94.7	88.4	98.5	94.7
DeepCCI [12]	96.5	92.2	99.1	97.3
RDMN	96.6	92.6	99.1	97.4
RDMN+multiAtt	97.3	93.4	99.1	97.8
RDMN+FP	97.8	93.3	99.4	98.0
RDMN+multiAtt+FP	98.0	94.1	99.5	98.1
RDMN+SMILES	98.1	94.3	99.7	97.8
RDMN+multiAtt+SMILES	**98.1**	**94.6**	**99.8**	**98.3**

Table 1: The performance on the CCI datasets reported in AUC and F1-score. *FP* stands for fingerprint and *multiAtt* stands for multiple attentions.

4 Related Work

Most existing work in chemical-chemical interaction requires manual feature engineering before a classifier is applied to detect interaction between two compounds (see [12] for references therein). To the best of our knowledge, DeepCCI [12] is the most recent work leveraging the end-to-end learning capability of neural networks for CCI. DeepCCI employs string representation of molecules, then uses CNN coupled with a pairwise loss defined by a Siamese network. In contrast, our approach using RDMN models the molecular graphs and their complex interaction directly. RDMN is theoretically powerful as it can model interaction of multiple compounds.

Related but distinct from CCI is the problem of predicting chemical reaction, where the goal is to produce reaction products. Typically heavy rules are applied to generate and filter candidate products, which are then ranked by machine learning algorithms [2, 8, 20]. The most recent work of [3] eliminates the intermediate step by generating the products in one go using a specialized graph neural network.

5 Conclusions

We have derived a new method to solve the problem of predicting chemical-chemical interaction given only basic molecular information. The method, named RDMN (which stands for Relational Dynamic Memory Networks), is a generalization of a recent Turing machine-like neural network called Graph Memory Networks [15]. RDMN supports arbitrary querying a set of graph-structured objects of the form `(query, {set of graphs}, ?)`, e.g., asking whether two or more chemical compounds will interact given specific conditions. Validated on a public dataset known as STITCH [10], RDMN was shown to achieve state-of-the-art accuracy.

There are rooms for future work. RDMN can be applied to learning any interaction between graphs, e.g., graph matching [23]. The experiments were run on pairwise interactions, but RDMN does not have such intrinsic restriction. It would be interesting to see how RDMN performs on high-order compound interactions – a setting of paramount importance in estimating treatments effects [21]. Furthermore, RDMN can be tailored for compound-protein prediction, e.g., by representing protein as a query through an LSTM (e.g., see [19]). An ambitious solution can be using the 3D structure of protein to build a graph. As rich chemical-chemical and chemical-protein interaction networks are being collected [10], it will be beneficial to leverage this prior knowledge, together with known chemical mechanisms, to bias the model in a knowledge-driven way. An exciting venue of exploration is the interaction of infinitely many molecules, such as water molecules and other dissolved substances. Finally, natural language queries over an arbitrary set of graphs will be an important AI problem.

A Model Specifications

We now present a specific realization of the generic framework proposed in Section 2 and validated in Section 3.

- The **q_encode()** operator in Eq. (1) translates the query q into a vector $\boldsymbol{q}$. This could be simply a look-up table when q is discrete (e.g., task or word). For textual query, we can use an RNN to represent q, and $\boldsymbol{q}$ is the last (or averaged) state of the RNN.
- The **m_load()** operator in Eq. (2), $(\boldsymbol{M}_c, \boldsymbol{A}_c) = \text{m\_load}\left(X_c, q\right)$, returns both the memory matrix $\boldsymbol{M}_c$ and the relational structure $\boldsymbol{A}_c$ given the query q. This is problem dependent. One example is when X_c represents a molecule, and q specifies a sub-structure of the molecule. Each atom i in the substructure is embedded into a vector $\boldsymbol{m}_i$, which is first assigned to the memory cell $\boldsymbol{M}_c[i]$. The relational structure $\boldsymbol{A}_c$ reflects the bonding between the atoms in the substructure.
- For the **m_read()** operator in Eq. (3), a content-based addressing scheme, also known as soft attention, is employed. At each time step t $(t = 1, 2, ..., T)$, the read vector $\boldsymbol{r}_{c,t}^{k}$ returned by the reading head k $(k = 1, 2, ..., K)$ over memory component c $(c = 1, 2, ..., C)$ is a sum of all memory cells, weighted by the attention probability vector $\boldsymbol{a}$:

$$\begin{aligned}\boldsymbol{a} &= \text{attention}_k\left(\boldsymbol{M}_{c,t-1}, \boldsymbol{h}_{t-1}\right)\\ \boldsymbol{r}_t &= \boldsymbol{M}_{c,t-1}\boldsymbol{a}\end{aligned}$$

  where attention$_k()$ is implemented as follows:

$$\boldsymbol{a}[i] = \text{softmax}\left(\boldsymbol{v}_k^{\top}\tanh\left(W_{a,k}\boldsymbol{M}_{c,t-1}[i] + U_{a,k}\boldsymbol{h}_{t-1}\right)\right).$$

- The controller is implemented as a recurrent net, where **s_update()** operator in Eq. (6) reads:

$$\begin{aligned}\tilde{\boldsymbol{h}}_t &= g\left(W_h\boldsymbol{h}_{t-1} + U_h\boldsymbol{r}_t\right)\\ \boldsymbol{h}_t &= \boldsymbol{\alpha}_h * \boldsymbol{h}_{t-1} + \left(1 - \boldsymbol{\alpha}_h\right) * \tilde{\boldsymbol{h}}_t \qquad (9)\end{aligned}$$

  where $(*)$ is element-wise multiplication; and $\boldsymbol{\alpha}_h \in (\boldsymbol{0}, \boldsymbol{1})$ is trainable forgetting gate that moderates the amount of information flowing from the previous step, e.g., $\boldsymbol{\alpha}_h = \sigma\left(\text{nnet}\left[\boldsymbol{h}_{t-1}, \boldsymbol{r}_t\right]\right)$ for sigmoid function σ.

- For the **m_update()** operator in Eq. (7), the memory is updated as follows:

$$\begin{aligned} \tilde{\boldsymbol{M}}_{c,t} &= g\left(U_m \boldsymbol{h}_t \boldsymbol{1}^\top + W_m \boldsymbol{M}_{c,t-1} + \sum_r V_{c,r} \boldsymbol{M}_{c,t-1} \hat{\boldsymbol{A}}_{c,r}\right) \\ \boldsymbol{M}_{c,t}\left[i\right] &= \boldsymbol{\alpha}_{c,i} * \boldsymbol{M}_{c,t-1}\left[i\right] + \left(1 - \boldsymbol{\alpha}_{c,i}\right) * \tilde{\boldsymbol{M}}_{c,t}\left[i\right] \end{aligned} \tag{10}$$

  for $c = 1, 2, ..., C$; where $\hat{\boldsymbol{A}}_{c,r}$ is the normalized adjacency matrix, i.e., $\hat{\boldsymbol{A}}_{c,r}\left[i,j\right] = \frac{\boldsymbol{A}_{c,r}[i,j]}{\sum_i \boldsymbol{A}_{c,r}[i,j]}$; $(*)$ is element-wise multiplication; and $\boldsymbol{\alpha}_{c,i} \in (\boldsymbol{0}, \boldsymbol{1})$ is a trainable forgetting gate.

- The **r_aggregate()** and the **r_combine()** operators in Eqs. (4,5) are simply averaging, i.e.,

$$\bar{\boldsymbol{r}}_t^k = \frac{1}{C}\sum_{c=1}^{C} \boldsymbol{r}_{c,t}^k; \qquad \boldsymbol{r}_t^* = \frac{1}{K}\sum_{k=1}^{K} \bar{\boldsymbol{r}}_t^k.$$

We use ReLU units for all steps and Dropout [17] is applied at the first and the last steps of the controller and the memory cells. Except for varying the number of attention heads to visualize the effect of multiple attentions, other hyper-parameters are tuned to maximize the performance on the validation dataset.

References

[1] Alan Baddeley. Working memory. *Science*, 255(5044):556–559, 1992.

[2] Connor W Coley, Regina Barzilay, Tommi S Jaakkola, William H Green, and Klavs F Jensen. Prediction of organic reaction outcomes using machine learning. *ACS central science*, 3(5):434–443, 2017.

[3] Kien Do, Truyen Tran, , and Svetha Venkatesh. Graph transformation policy network for chemical reaction prediction. *ICLR'19 submission. URL: https://openreview.net/forum?id=r1f78iAcFm*, 2019.

[4] Kien Do, Truyen Tran, and Svetha Venkatesh. Learning deep matrix representations. *arXiv preprint arXiv:1703.01454*, 2018.

[5] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[6] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[7] Herbert Jaeger. Artificial intelligence: Deep neural reasoning. *Nature*, 538(7626):467, 2016.

[8] Wengong Jin, Connor Coley, Regina Barzilay, and Tommi Jaakkola. Predicting Organic Reaction Outcomes with Weisfeiler-Lehman Network. In *Advances in Neural Information Processing Systems*, pages 2604–2613, 2017.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Michael Kuhn, Christian von Mering, Monica Campillos, Lars Juhl Jensen, and Peer Bork. STITCH: Interaction networks of chemicals and proteins. *Nucleic acids research*, 36(suppl_1):D684–D688, 2007.

[11] Sunyoung Kwon and Sungroh Yoon. DeepCCI: End-to-end Deep Learning for Chemical-Chemical Interaction Prediction. *arXiv preprint arXiv:1704.08432*, 2017.

[12] Sunyoung Kwon and Sungroh Yoon. End-to-end representation learning for chemical-chemical interaction prediction. *IEEE/ACM transactions on computational biology and bioinformatics*, 2018.

[13] Allen Newell. *Unified theories of cognition*. Harvard University Press, 1994.

[14] Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Faster training of very deep networks via p-norm gates. *ICPR*, 2016.

[15] Trang Pham, Truyen Tran, and Svetha Venkatesh. Graph memory networks for molecular activity prediction. *ICPR*, 2018.

[16] Trang Pham, Truyen Tran, and Svetha Venkatesh. Relational dynamic memory networks. *arXiv preprint arXiv:1808.04247*, 2018.

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[18] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[19] Masashi Tsubaki, Kentaro Tomii, and Jun Sese. Compound-protein interaction prediction with end-to-end learning of neural networks for graphs and sequences. *Bioinformatics*, 2018.

[20] Jennifer N Wei, David Duvenaud, and Alán Aspuru-Guzik. Neural networks for the prediction of organic chemistry reactions. *ACS Central Science*, 2(10):725–732, 2016.

[21] Zohar B Weinstein, Andreas Bender, and Murat Cokol. Prediction of synergistic drug combinations. *Current Opinion in Systems Biology*, 4:24–28, 2017.

[22] Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In *International Conference on Machine Learning*, pages 2397–2406, 2016.

[23] Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2684–2693, 2018.