Relational dynamic memory networks

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

Abstract

Neural networks excel in detecting regular patterns but are less successful in representing and manipulating complex data structures, possibly due to the lack of an external memory. This has led to the recent development of a new line of architectures known as Memory-Augmented Neural Networks (MANNs), each of which consists of a neural network that interacts with an external memory matrix. However, this RAM-like memory matrix is unstructured and thus does not naturally encode structured objects. Here we design a new MANN dubbed Relational Dynamic Memory Network (RDMN) to bridge the gap. Like existing MANNs, RDMN has a neural controller but its memory is structured as multi-relational graphs. RDMN uses the memory to represent and manipulate graph-structured data in response to query; and as a neural network, RDMN is trainable from labeled data. Thus RDMN learns to answer queries about a set of graph-structured objects without explicit programming. We evaluate the capability of RDMN on several important prediction problems, including software vulnerability, molecular bioactivity and chemical-chemical interaction. Results demonstrate the efficacy of the proposed model.

Keywords: Memory-augmented neural networks; graph neural networks; relational memory; graph-graph interaction

1   Introduction

To support reasoning – the process of forming answer to a new question by deliberately manipulating previously acquired knowledge [7] – intelligent systems need a working memory to load, hold, integrate and alter information needed for processing [14]. This has inspired a recent line of research collectively known as memory-augmented neural networks (MANNs), in which a neural network reads from and writes to an external memory matrix [22, 30, 50]. This results in a powerful differentiable machinery that can learn programs from data and answer complex queries. Technically, memory provides a short-cut for passing signal and gradient between query, input and output [33], making credit-assignment easier in a long chain of computation. However, these memory modules have been developed to be generic without considering the structural information that may be available in the objects being queried about. For structured objects such as graphs, encoding structural information into a flat memory matrix is not straightforward [22]. We conjecture that a memory architecture that is reflective of the structure of the data might be easier to train and generate a more focused answer.

Here we introduce a new MANN called Relational Dynamic Memory Networks (RDMN) capable of answering queries about structured data[1]. We consider the case where data are attributed graphs, e.g., molecular graphs or function call graphs in software. RDMN is composed of a controller and a memory dynamically organized as a set of networks of memory cells. The memory structure is not fixed but shaped by the structure within the input data conditioned on the query. This memory design is partly inspired by the current understanding of working memory as dynamic networks emerged from functional coordination between active regions in brain [8, 18, 49]. The query triggers the controller in RDMN to initiate a reasoning episode, during which the controller first prepares the memory structure and content based on the input data, then iteratively manipulates the memory states until a probable answer is reached. Neighboring memory cells interact directly during the reasoning process, i.e., cell states are updated by aggregating the write content from the controller as well as the messages sent by neighboring cells. Distant cells within a network or between networks interact indirectly through the exchanging messages with the controller.

With this architecture, the RDMN supports learning to answer queries about not just a single graph, but also several interacting graphs. In other words, RDMN learns o solve problems of the form (`query, {set of graphs}, ?`). This is a generalized form of *graph-graph interaction*, an under-explored machine learning area in its own right. An example of query over single graph is predicting whether a drug molecule (a graph) has any positive effect on a type of disease (a query). An example of querying over multiple graphs is chemical-chemical interaction prediction, where the query can include environmental factors, the graphs are basic molecular structures, and the answer is the interaction strength between molecules [19, 31].

In summary, we claim the following contributions:

- A novel differentiable architecture named Relational Dynamic Memory Networks (RDMN), which consists of a structured working memory module augmented to a neural network.

- A general solution for learning to answer queries about multiple graphs. In particular, it solves a relatively novel problem of predicting graph-graph interaction.

- Validation of RDMN on three distinct tasks: software source code vulnerability assessment, molecular bioactivity prediction, and chemical-chemical interaction prediction.

The rest of the paper is organized as follows. Section 2 briefly introduces preliminaries on memory-augmented neural networks and graph neural networks. The main contribution of the paper, the RDMN, is described in Section 3 with implementation detailed in Section 4. Experiments and results are reported in Section 5. Section 6 reviews related work, followed by conclusion.

---

[1]A preliminary version of this work was published in a conference [40].

2 Preliminaries

2.1 Memory-augmented neural nets

A MANN consists of a neural controller augmented with an external memory. The controller can be a feedforward net (memoryless) or a recurrent net (which has its own short-term memory). The external memory is often modeled as a matrix. At each time step, the controller reads an input, updates the memory, and optionally emits an output. With stationary update rules, the memory can be rolled out over time into a recurrent matrix net [16]. Let us denote the memory matrix as $\boldsymbol{M} \in \mathbb{R}^{d \times m}$ for d dimensions and m slots. Upon seeing a new evidence $\boldsymbol{x}$ (which could be empty), the controller generates a write vector $\boldsymbol{w} \in \mathbb{R}^p$ which causes a candidate memory update. As an example, the update may take the following form:

$$\tilde{\boldsymbol{M}} \leftarrow \phi \left(W \boldsymbol{w} \boldsymbol{1}^\top + U \boldsymbol{M} R + B \right) \tag{1}$$

where $W \in \mathbb{R}^{d \times p}$ is data encoding matrix, $U \in \mathbb{R}^{d \times d}$ is transition matrix , $R \in \mathbb{R}^{m \times m}$ is the graph of relation between memory slots, and $B \in \mathbb{R}^{d \times m}$ is bias. The memory is then updated as: $\boldsymbol{M} \leftarrow f \left(\boldsymbol{M}, \tilde{\boldsymbol{M}} \right)$ for some function f. For example, a simple but effective update is linear forgetting: $f \left(\boldsymbol{M}, \tilde{\boldsymbol{M}} \right) = \boldsymbol{\alpha} * \boldsymbol{M} + (1 - \boldsymbol{\alpha}) * \tilde{\boldsymbol{M}}$, where $*$ is element-wise multiplication, and $\boldsymbol{\alpha} \in (0, \boldsymbol{1})$ is learnable forget-gate.

The controller then reads the memory to determine what to do next. For example, the End-to-End Memory Network [50] maintains a dynamic state of an answer $\boldsymbol{u}$ by reading the memory as follows:

$$\boldsymbol{a} \leftarrow \text{softmax} \left(\boldsymbol{M}^\top A \boldsymbol{u} \right)$$

$$\boldsymbol{u} \leftarrow U \boldsymbol{u} + C M \boldsymbol{a}$$

where A, U, C are trainable parameters. Here $\boldsymbol{k} = A \boldsymbol{u}$ plays the role of a key in the content-addressing scheme, and $\boldsymbol{a}$ assigns an attention weight to each memory slot.

2.2 Graph neural networks

A graph is a tuple $\mathbf{G} = \{\mathbf{A}, \mathbf{R}, \mathbf{X}\}$, where $\mathbf{A} = \left\{ a^1, ..., a^M \right\}$ are M nodes; $\mathbf{X} = \left\{ \boldsymbol{x}^1, ..., \boldsymbol{x}^M \right\}$ is the set of node features; and $\mathbf{R}$ is the set of relations in the graph. Each tuple $\left\{ a^i, a^j, r, \boldsymbol{b}^{ij} \right\} \in \mathbf{R}$ describes a relation of type r $(r = 1...R)$ between two nodes a^i and a^j. The relations can be unidirectional or bi-directional. The vector $\boldsymbol{b}^{ij}$ represents the link features. Node a^j is a neighbor of a^i if there is a connection between the two nodes. Let $\mathcal{N}(i)$ be the set of all neighbors of a^i and $\mathcal{N}_r(i)$ be the set of neighbors connected to a^i through type r. This implies $\mathcal{N}(i) = \cup_r \mathcal{N}_r(i)$.

Graph neural networks are a class of neural nets that model the graph structure directly. The most common type is message passing graph neural networks [20, 39, 46], which update note states using messages sent from the neighborhood. In the *message aggregation* step, we combine multiple messages sent to node i into a single message vector $\boldsymbol{m}_i$:

$$\boldsymbol{m}_i^t = g^{\mathrm{a}} \left(\boldsymbol{x}_i^{t-1}, \left\{ \left(\boldsymbol{x}_j^{t-1}, \boldsymbol{e}_{ij} \right) \right\}_{j \in \mathcal{N}(i)} \right) \tag{2}$$

where $g^{\mathrm{a}}(\cdot)$ can be an attention [4] or a pooling architecture. During the *state update* step, the node state is updated as follows:

$$\boldsymbol{x}_i^t \leftarrow g^{\mathrm{u}} \left(\boldsymbol{x}_i^{t-1}, \boldsymbol{m}_i^t \right) \tag{3}$$

where $g^{\mathrm{u}}(\cdot)$ can be any type of deep neural networks such as MLP [28], RNN [46], GRU [34] or Highway Net [39].

3   Relational dynamic memory

(a) Uni-component memory   (b) Multi-component memory
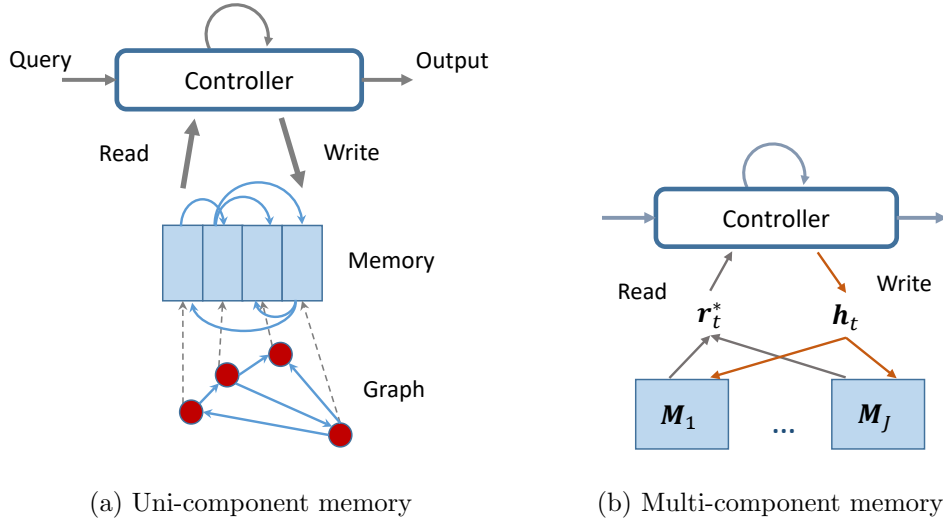
Figure 1: Relational Dynamic Memory Networks (RDMN). (a) With single component memory showing how the graph is mapped into structured memory. At the first step, the controller reads the query; the memory is initialized by the input graph, one node embedding per memory cell. Then during the reasoning process, the controller iteratively reads from and writes to the memory. Finally, the controller emits the output. (b) RDMN with multi-component memory.

Here we present our main contribution, the Relational Dynamic Memory Network (RDMN), which is built upon two independent concepts, the Memory-Augmented Neural Net (see Section 2.1) and Graph Neural Net (see Section 2.2). RDMN is a neural computer, similar to Neural Turing Machines (NTM) [21, 22], thus consisting of a CPU-like controller and a RAM-like memory. The controller is a recurrent neural network responsible for memory manipulation. Like NTMs, RDMN is fully differentiable allowing it 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 (Fig. 1a) or multiple components (Fig. 1b), each of which is a graph of cells. Links between cells are dynamically defined by the structure of the input data and the query.

Presented with input graphs and a query, RDMN initiates a reasoning episode, during which the machine dynamically allocates memory to represent the input graphs, with one memory component per graph. The memory structure of a component reflects the corresponding input graph with respect to the query. Then the controller takes the query as the input and repeatedly reads from the memory, processes and writes back to the memory cells. At each reasoning step, the cell content is updated by the signals from the controller and its neighbor memory cells. 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 J memory components, the system, when rolled out during reasoning, consists of $J+1$ recurrent *matrix* neural networks interacting with each other [16].

3.1 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:

$$\boldsymbol{q} = \text{q\_encode}\,(q) \tag{4}$$

Let X_c be a graph-structured representation of object c, for $c = 1, 2, .., J$. X_c is loaded into a memory component $\boldsymbol{M}_c$ consisting of K_c memory cells:

$$\boldsymbol{M}_c = \text{m\_load}\,(X_c, \boldsymbol{q}) \tag{5}$$

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 $\boldsymbol{A}_c = \{A_{c,r}\}_{r \in \mathcal{R}_c}$.

Given $\boldsymbol{q}$ and $\boldsymbol{M} = \{\boldsymbol{M}_1, \boldsymbol{M}_2, ..., \boldsymbol{M}_J\}$, the controller initiates a reasoning episode to compute the output probability $P_\theta\,(\boldsymbol{y} \mid \boldsymbol{M}, \boldsymbol{q})$. In what follows, we briefly present how the computation occurs, leaving the detailed implementation in Section 4.

3.2 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 and updates its state and memory as follows:

$$\boldsymbol{r}_{c,t} = \text{m\_read}\,(\boldsymbol{M}_{c,t-1}, \boldsymbol{h}_{t-1}) \tag{6}$$

$$\boldsymbol{r}_t^* = \text{r\_aggregate}\,\{\boldsymbol{r}_{tc}\}_{c=1}^{J} \tag{7}$$

$$\boldsymbol{h}_t = \text{s\_update}\,(\boldsymbol{h}_{t-1}, \boldsymbol{r}_t^*) \tag{8}$$

$$\boldsymbol{M}_{c,t} = \text{m\_update}\,(\boldsymbol{M}_{c,t-1}, \boldsymbol{h}_t, \boldsymbol{A}_c) \tag{9}$$

The read operator can have multiple read heads, and the output is aggregated (e.g., averaged). The r_aggregate{·} function operates on the set of retrieved memory vectors and combines them into one vector.

3.3 Decoder

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

$$\boldsymbol{y} = \text{decode}\left(\boldsymbol{h}_T, q\right) \tag{10}$$

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

4 An implementation

We now present a specific realization of the generic framework proposed in Section 3 and validated in Section 5. All operators are parameterized and differentiable. We use ReLU units for all steps and Dropout is applied at the first and the last steps of the controller and the memory cells.

4.1 Operators

- For the **m_read** operator in Eq. (6), a content-based addressing scheme, also known as soft attention, is employed. At each time step t $(t = 1, ..., T)$, $\boldsymbol{r}_t$ is a sum of all memory cells, weighted by the attention probability a_t^i, for each memory cell $i = 1, ..., K$:

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

  where attention is implemented as follows:

$$
a_t^i = \text{softmax}\left(\boldsymbol{v}^\top \tanh\left(W_a \boldsymbol{M}_{t-1}[i] + U_a \boldsymbol{h}_{t-1}\right)\right).
$$

  This single read-head can be extended straightforwardly to multiple read-heads to produce multiple read vectors $\left\{\boldsymbol{r}_t^1, \boldsymbol{r}_t^2, ..., \boldsymbol{r}_t^F\right\}$. This is followed by a pooling operator, e.g., mean pooling: $\boldsymbol{r}_t = \frac{1}{F}\sum_f \boldsymbol{r}_t^f$.

- The controller is implemented as a recurrent net, where **s_update** operator in Eq. (8) reads:

$$
\begin{aligned}
\tilde{\boldsymbol{h}}_t &= g\left(W_h \boldsymbol{h}_{t-1} + U_h \boldsymbol{r}_t\right) \\
\boldsymbol{h}_t &= \boldsymbol{\alpha}_h * \boldsymbol{h}_{t-1} + (1 - \boldsymbol{\alpha}_h) * \tilde{\boldsymbol{h}}_t
\end{aligned} \tag{11}
$$

  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. (9), the memory is updated as follows:

$$\tilde{\boldsymbol{M}}_t = g\left(U_m \boldsymbol{h}_t \boldsymbol{1}^\top + W_m \boldsymbol{M}_{t-1} + \sum_r V_r \boldsymbol{M} \hat{A}_r\right)$$

$$\boldsymbol{M}_{c,t}[i] = \boldsymbol{\alpha}_{c,i} * \boldsymbol{M}_{c,t-1}[i] + (1 - \boldsymbol{\alpha}_{c,i}) * \tilde{\boldsymbol{M}}_{c,t}[i] \qquad (12)$$

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

- The **r_aggregate** operator in Eq. (7) is simply an averaging, i.e.,

$$\boldsymbol{r}_t^* = \frac{1}{C} \sum_{c=1}^{C} \boldsymbol{r}_{tc}$$

4.2 RDMN for multi-task learning

RDMN can be easily applied for multi-task learning. Suppose that the dataset contains n tasks. We can use the query to indicate the task. If a graph is from task k, the query for the graph is an one-hot vector of size n: $\boldsymbol{q} = [0, 0, ..., 1, 0, ...]$, where $\boldsymbol{q}^k = 1$ and $\boldsymbol{q}^j = 0$ for $j = 1, ..., n, \ j \neq k$. The task index now becomes the input signal for RDMN. With the signal from the task-specific query, the attention can identify which substructure is important for a specific task to attend on.

4.3 Training

Given a training set, in which each data instance has the form (`query,` $\{$`graph`$_1$`,graph`$_2$`,...,graph`$_J\}$, `answer`), RDMN is trained in a supervised fashion from end-to-end, i.e., by minimizing a loss function. The typical loss is $-\log P\left(\boldsymbol{y} \mid q, \{X_c\}_{c=1}^J\right)$. As the entire system is differentiable, gradient-based techniques can be applied for parameter optimization[2].

5 Experiments and results

In this section, we demonstrate RDMN on three applications: software vulnerability detection (single query, Section 5.1), molecular activity prediction (multiple queries, Section 5.2), and chemical-chemical interaction (query about graph-graph interaction, Section 5.3).

5.1 Software vulnerability

This application asks if a piece of source code is potentially vulnerable to security risks. In particular, we consider each Java class as a piece, which consists of attribute declarations and methods. A class is then represented as a graph, where nodes are methods and edges are function calls between methods (thus, there is only one relation type). Class-level declaration is used as query, and the answer is binary indicator of vulnerability. The

---

[2]Discrete operators can also be implemented with help of policy gradient algorithms.

dataset was collected from [13], which consists of 18 Java projects. The dataset is pre-processed by removing all replicated files of different versions in the same projects. This results in 2,836 classes of which 1,020 are potentially vulnerable.

Embedding of graph nodes and query is pretrained as follows. Methods and attribute declarations of Java classes are treated as sequences of code tokens and their representation is learned through language modeling using LSTM, that is to predict the next token w_t given previous tokens $\boldsymbol{w}_{1:t-1}$ via $P\left(w_t \mid \boldsymbol{w}_{1:t-1}\right)$. Rare tokens are collectively designated as <UNK>. The feature vector of each sequence is the mean of all hidden states outputted by the LSTM. After this step, each sequence is represented as a feature vector of 128 units.

For comparison, we implemented three strong non-neural classifiers: Support Vector Machine (SVM), Random Forest (RF) and Gradient Boosting Machine (GBM) running on the averaged feature vector of all methods and attribute declarations. Fig. 2 reports the performance measured by AUC (Area under the ROC curve) and F1-score for RDMN and these three classifiers. RDMN is competitive, although the improvement over the best performing method (GBM) is not large (2 points on AUC and 1 point on F1-score).
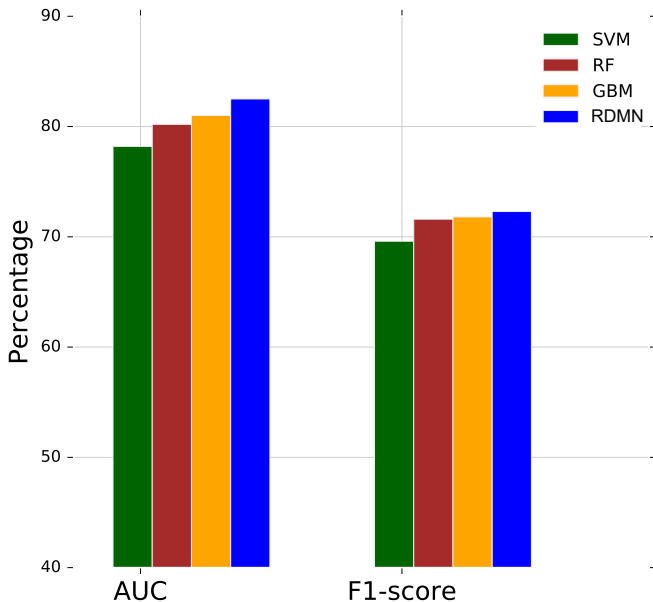

Figure 2: Performance on code vulnerability detection, measured in AUC and F1-score (%). Best viewed in color.

5.2   Molecular bioactivities

This application asks if a chemical compound is biologically active on a given disease. Here each compound molecule is represented as a graph, where nodes are atoms and edges are bond types between them. Activity and disease form a query (e.g., coded as an one-hot vector). The answer will be binary indicator of the activity with respect to the given

disease.

Datasets We conducted experiments on nine NCI BioAssay activity tests collected from the PubChem website[3]. Seven of them are activity tests of chemical compounds against different types of cancer: breast, colon, leukemia, lung, melanoma, central nerve system and renal. The others are AIDS antiviral assay and Yeast anticancer drug screen. Each BioAssay test contains records of activities for chemical compounds. We chose the two most common activities for classification: "active" and "inactive". The statistics of data is reported in Table 1. The datasets are listed by the ascending order of number of active compounds. "# Graph" is the number of graphs and "# Active" is the number of active graph against a BioAssay test These datasets are unbalanced, therefore "inactive" compounds are randomly removed so that the Yeast Anticancer dataset has 25,000 graphs and each of the other datasets has 10,000 graphs.

Table 1: Summary of 9 NCI BioAssay datasets.

No.	Dataset	# Active	# Graph
1	AIDS Antiviral	1513	41,595
2	Renal Cancer	2,325	41,560
3	Central Nervous System	2,430	42,473
4	Breast Cancer	2,490	29,117
5	Melanoma	2,767	39,737
6	Colon Cancer	2,766	42,130
7	Lung Cancer	3,026	38,588
8	Leukemia	3,681	38,933
9	Yeast Anticancer	10,090	86,130

We used RDKit[4] to extract the structure of molecules, the atom and the bond features. An atom feature vector is the concatenation of the one-hot vector of the atom and other features such as atom degree and number of H atoms attached. We also make use of bond features such as bond type and a binary value indicating if a bond is in a ring.

5.2.1 Experiment settings

Baselines For comparison, we use several baselines:

- *Classic classifiers running on feature vectors extracted from molecular graphs.* Classifiers are Support Vector Machine (SVM), Random Forest (RF), Gradient Boosting Machine (GBM), and multi-task neural network (MT-NN) [41]. For feature extraction, following the standard practice in the computational chemistry literature, we use the RDKit to extract molecular fingerprints – the encoding of the graph structure of the molecules by a vector of binary digits, each presents the presence or absence of particular substructures in the molecules. There are different algorithms to achieve molecular fingerprints and the state of the art is the extended-connectivity circular fingerprint (ECFP) [42]. The dimension of the fingerprint features is set by

---

[3]https://pubchem.ncbi.nlm.nih.gov/
[4]http://www.rdkit.org/

1,024. SVM, RF and GBM predict one activity at a time, but MT-NN predict all activities at the same time.

- *Neural graphs which learn to extract features.* In particular, we use Neural Fingerprint (NeuralFP) [17]. This predicts one activity at a time.

Model setting We set the number of hops to $T = 10$ following recommendation in [38]. Other hyper-parameters are tuned on the validation dataset.

5.2.2 Results

The impact of joint training We have options to train each task separately (query set to unity, one memory per task) or to train jointly (query is an one-hot vector, shared memory for all tasks). To investigate more on how multi-task learning impacts the performance of each task, we reports the F1-score of RDMN in both separate and joint training settings on each of the nine datasets (Fig. 3). Joint training with RDMN model improves the performance of seven datasets on different types of cancers by 10%-20% on each task. However, it has little effect on AIDS antiviral and Yeast anticancer datasets. These could be explained by the fact that cancers share similar genomic footprints and thus each cancer can borrow the strength of statistics of the cancer family. This does not hold for the other conditions, which are very different from the rest. However, joint training is still desirable because we need to maintain just one model for all queries.
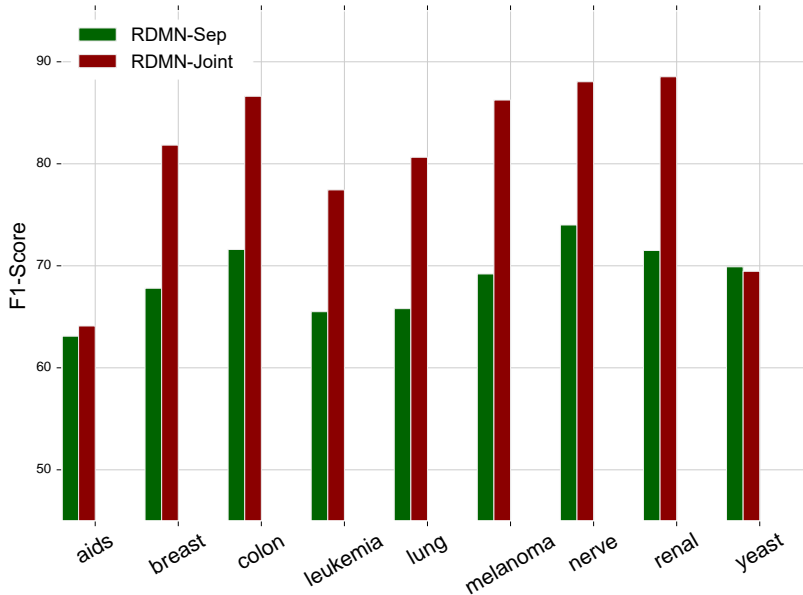

Figure 3: The comparison in performance of RDMN when training separately (RDMN-Sep) and jointly (RDMN-Joint) for all datasets. Best viewed in color.

The impact of more tasks We evaluate how the performance of RDMN on a particular dataset is affected by the number of tasks. We chose AIDS antiviral, Breast Cancer and Colon Cancer as the experimental datasets. For each experimental dataset, we start to train it and then repeatedly add a new task and retrain the model. The orders of the first three new tasks are: (AIDS, Breast, Colon) for AIDS antiviral dataset, (Breast, AIDS, Colon) for Breast Cancer dataset and (Colon, AIDS, Breast) for Colon Cancer dataset. The orders of the remaining tasks are the same for three datasets: (Leukemia, Lung, Melanoma, Nerve, Renal and Yeast).

Fig. 4 illustrates the performance of the three chosen datasets with different number of jointly training tasks. The performance of Breast and Colon Cancer datasets decreases when jointly trained with AIDS antiviral task, increases after adding more tasks, and then remains steady or slightly reduces after seven tasks. Joint training does not improve the performance on the AIDS antiviral dataset.
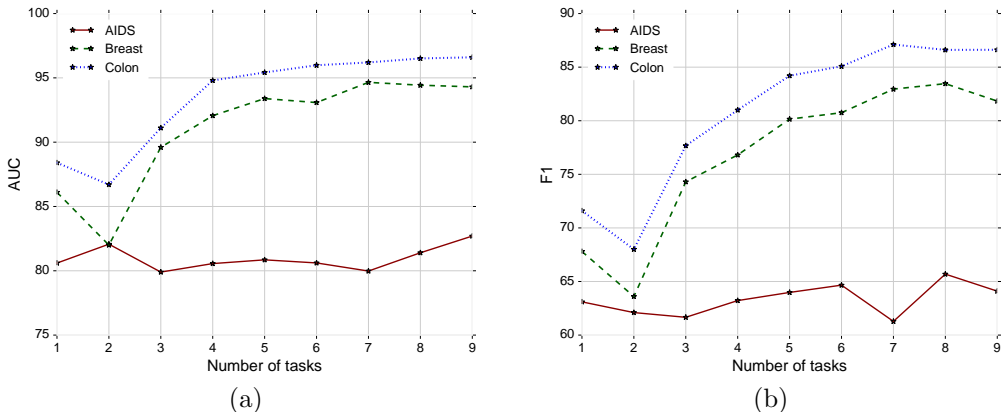

Figure 4: The performance of three datasets when increasing the number of jointly training tasks, reported in (a) AUC and (b) F1-score.

Comparative results Table 2 reports results, measured in Micro F1-score, Macro F1-score and the average AUC over all datasets. The best method for separated training on fingerprint features is SVM with 66.4% of Micro F1-score and on graph structure is RDMN with the improvement of 2.7% over the non-structured classifiers. The joint learning settings improve by 9.1% of Micro F1-score and 10.7 % of Macro F1-score gain on fingerprint features and 8.7% of Micro F1-score and 11.6% of Macro F1-score gain on graph structure.

5.3   Chemical-chemical interaction (CCI)

For this application we asks if two or more molecules interacts. Here the queries can be about the strength of interaction given environmental conditions (e.g., solution, temperature, pressure, presence of catalysts).

Table 2: Performance over all datasets, measured in Micro F1, Macro F1 and the average AUC.

Model	MicroF1	MacroF1	Average AUC
SVM	66.4	67.9	85.1
RF	65.6	66.4	84.7
GB	65.8	66.9	83.7
NeuralFP [17]	68.2	67.6	85.9
MT-NN [41]	75.5	78.6	90.4
RDMN	**77.8**	**80.3**	**92.1**

Datasets We conducted experiments on chemical-chemical interaction data downloaded from the STITCH database [29] (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 [31], 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[5] was used to extract graph structures, atom and bond features, fingerprints and SMILES of the molecules.

5.3.1   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 [48], which are two strong classifiers in many applications [38]. 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 [31], 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 hyper-parameters for DeepCCI as suggested by the authors [31].

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
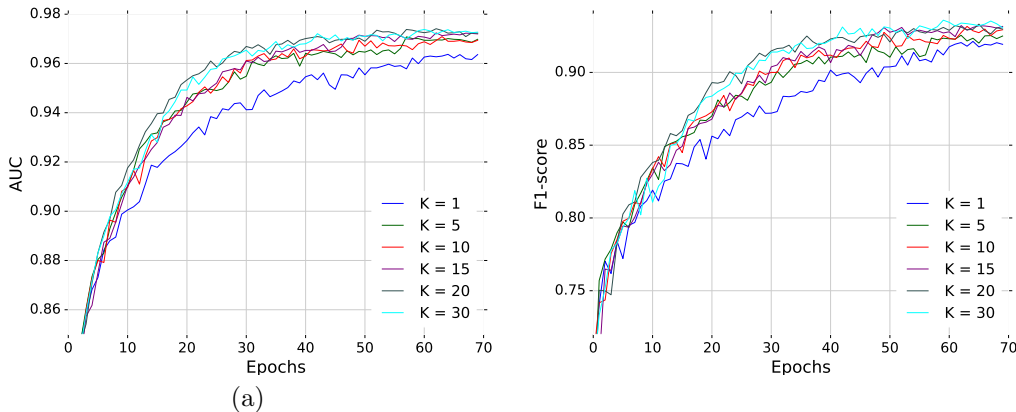
---

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

Figure 5: 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.

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.

5.3.2 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. 5 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 3 that using fingerprint vectors or the hidden state generated by DeepCCI
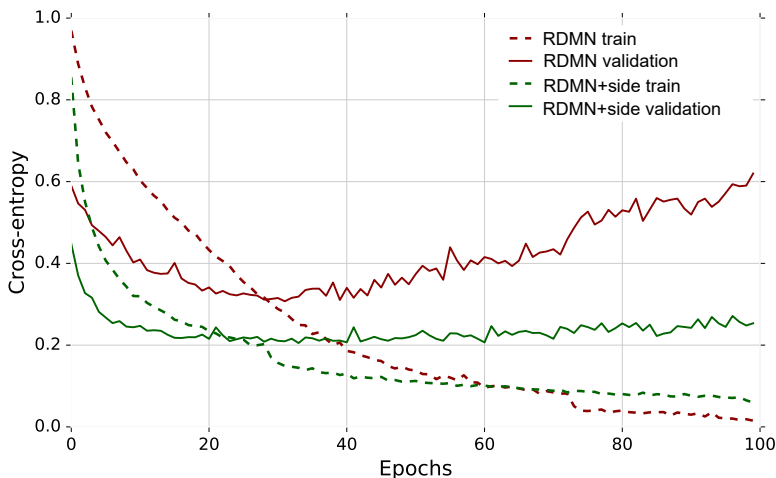
13

Figure 6: Training and validation losses of RDMN with and without side information during training. Best viewed in color.

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. 6 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 3 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-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.

6   Related work

Working memory and reasoning This paper is partly inspired by the concept of "working memory", a brain faculty for holding and manipulating information for an extended period of time (e.g., seconds to minutes) [3]. Working memory is a critical component for high-level cognition (e.g., reasoning, meta-reasoning) [14]. Observations found that working memory arises through functional coordination between brain regions [49], which may suggest the graph-theoretical approach for cognitive modeling. An important feature

	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 [31]	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 3: The performance on the CCI datasets reported in AUC and F1-score. *FP* stands for fingerprint and *multiAtt* stands for multiple attentions.

of working memory is the fast loading (or binding) of information into the memory, allowing rapid task-switching and selective attention [6]. Our work draws inspiration from these findings and theory, but does not aim to be biologically relevant. Rather we aim for building a neural network capable of *neural reasoning* [24]. This capability is needed for tasks such as graph traversal [22], knowledge graph completion [47] and question answering [50, 52]. To facilitate relational reasoning, recent neural networks [45] have been proposed to model pair-wise relationship between objects. A recurrent variant allows more dynamic reasoning process [44]. Our work contributes to this line of research by introducing a controller into a relational memory module, and thus helping in answering arbitrary queries.

Graph representation and querying   There has been a surge of interest in learning graph representation for the past few years [9, 34, 36, 53]. Deep spectral methods have been introduced for graphs of a given adjacency matrix [9], whereas we allow arbitrary graph structures, one per graph. Several other methods extend convolutional operations to irregular local neighborhoods [2, 36, 39], or employ recurrent paths along the random walk from a node [46]. Graph dynamics has been recently studied, e.g., Gated Graph Transformer [25], which models graph state transition over time. Related to but distinct from our work is the work about querying on graphs in the database community [35]. There are sub-problems that can benefit from RDMN, however, including graph matching, maximum clique and segmentation (clustering). Our work adds to the existing literature by allowing arbitrary querying from a single or multiple graphs. Note that our system can query about inferred knowledge instead of just factual knowledge.

Memory-augmented neural nets (MANN)   MANNs are new developments but have found a wide range of applications, including question-answering [30, 50], graph processing [22], algorithmic tasks [22], meta-learning [43], healthcare [33] and dialog systems [32]. Notable MANN architectures include Neural Turing Machine [21] and its recent cousin, the Differentiable Neural Computer [22]. Here all operations are fully differentiable allowing end-to-end training with and the memory dynamically updated during the reasoning process. This design is powerful but poses great challenges for

implementation. End-to-End Memory Networks [50] simplify this by loading the entire input matrix into the memory and fixing the memory content. Similar architectures have been subsequently proposed to solve specific tasks, including the Dynamic Memory Networks for question-answering problems [30], the Recurrent Entity Net [23] for tracking entities, and the key-value structure for rare-events [26]. Limited work has been done for the structured memory [5, 37]. Our RDMN differs from these models by using a dynamic memory organized as a graph of cells. The cells interact not only with the controller but also with other cells to embed the substructure in their states. A potential mechanism for a dynamic working memory is introduced in [16, 44], where relations between cells are either learned [16] or dynamically estimated through self-attention [44]. Our work builds on top of this memory module by introducing a controller to manage update of the memory matrix.

Source code modeling Our application of RDMN for software is in line with the current direction of modeling source code using structures such as trees [12] and graphs [1]. The recurrent nature of the RDMN, in principle, would capture a program execution without actually running the code [10].

Molecular graphs modeling Our application to chemical compound classification bears some similarity to the work of [17], where graph embedding is also collected from node embedding at each layer and refined iteratively from the bottom to the top layers. However, our treatment is more principled and more widely applicable to multi-typed edges [11, 27]. The controller used in RDMN resembles the virtual atom mentioned in [20] for a single molecule setting. Our application to chemical-chemical interaction (CCI) differs radically from existing machine learning work in the area, whose prediction pipeline typically consists of manual feature engineering before a classifier is applied (e.g., see [31] for references therein). To the best of our knowledge, DeepCCI [31] 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, which are essentially predicting bond changes (e.g., see [15] for references therein).

7   Conclusion

We have proposed RDMN (which stands for Relational Dynamic Memory Network), a new neural network augmented with a graph-structured "working" memory. RDMN supports arbitrary querying over a set of graph-structured objects, that is, solving the problem of the form (`query, {set of graphs}, ?`). We applied RDMN to three tasks: software source code vulnerability, molecular bioactivity and chemical-chemical interaction. The results are competitive against rivals, demonstrating a possibility to implement differentiable neural reasoning for highly complex tasks.

A limitation of the current formulation is that the memory structure in RDMN is fixed once constructed from data graphs. A future work would be deriving dynamic

memory graphs that evolve with time, e.g., using the technique introduced in [53], or a gated message passing between memory cells. Also, multiple controllers may be utilized to handle asynchronous issues (e.g., see [33]) as they can partially share some memory components. Another limitation of the current work is that the model was only validated on single graphs and pairwise graph-graph interaction, but RDMN does not have such intrinsic restriction. For example, it would be interesting to see how RDMN performs on high-order compound interactions – a setting of paramount importance in estimating treatments effects [51].

References

[1] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. *ICLR*, 2018.

[2] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016.

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

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Trapit Bansal, Arvind Neelakantan, and Andrew McCallum. RelNet: End-to-end Modeling of Entities & Relations. *arXiv preprint arXiv:1706.07179*, 2017.

[6] Min Bao, Zhi-Hao Li, and Da-Ren Zhang. Binding facilitates attention switching within working memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 33(5):959, 2007.

[7] Léon Bottou. From machine learning to machine reasoning. *Machine Learning*, 94(2):133–149, 2014.

[8] Urs Braun, Axel Schäfer, Henrik Walter, Susanne Erk, Nina Romanczuk-Seiferth, Leila Haddad, Janina I Schweiger, Oliver Grimm, Andreas Heinz, Heike Tost, et al. Dynamic reconfiguration of frontal brain networks during executive cognition in humans. *Proceedings of the National Academy of Sciences*, 112(37):11678–11683, 2015.

[9] Joan Bruna, W Zaremba, A Szlam, and Yann LeCun. Spectral networks and deep locally connected networks on graphs. In *ICLR*, 2014.

[10] Min-je Choi, Sehun Jeong, Hakjoo Oh, and Jaegul Choo. End-to-end prediction of buffer overruns from raw source code via neural memory networks. *IJCAI*, 2017.

[11] Connor W Coley, Regina Barzilay, William H Green, Tommi S Jaakkola, and Klavs F Jensen. Convolutional embedding of attributed molecular graphs for physical property prediction. *Journal of chemical information and modeling*, 57(8):1757–1772, 2017.

[12] Hoa Khanh Dam, Trang Pham, Shien Wee Ng, Truyen Tran, John Grundy, Aditya Ghose, Taeksu Kim, and Chul-Joo Kim. A deep tree-based model for software defect prediction. *arXiv preprint arXiv:1802.00921*, 2018.

[13] Hoa Khanh Dam, Truyen Tran, Trang Pham, Shien Wee Ng, John Grundy, and Aditya Ghose. Automatic feature learning for vulnerability prediction. *arXiv preprint arXiv:1708.02368*, 2017.

[14] Adele Diamond. Executive functions. *Annual review of psychology*, 64:135–168, 2013.

[15] 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.

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

[17] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[18] Johan Eriksson, Edward K Vogel, Anders Lansner, Fredrik Bergström, and Lars Nyberg. Neurocognitive architecture of working memory. *Neuron*, 88(1):33–46, 2015.

[19] David Fooshee, Aaron Mood, Eugene Gutman, Mohammadamin Tavakoli, Gregor Urban, Frances Liu, Nancy Huynh, David Van Vranken, and Pierre Baldi. Deep learning for chemical reaction prediction. *Molecular Systems Design & Engineering*, 2018.

[20] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the International Conference on Machine Learning*, 2017.

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

[22] 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.

[23] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. *ICLR*, 2017.

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

[25] Daniel D Johnson. Learning graphical state transitions. *ICLR*, 2017.

[26] Łukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. *ICLR*, 2017.

[27] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[28] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

[29] 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.

[30] Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *ICML*, 2016.

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

[32] Hung Le, Truyen Tran, Thin Nguyen, and Svetha Venkatesh. Variational memory encoder-decoder. *NIPS*, 2018.

[33] Hung Le, Truyen Tran, and Svetha Venkatesh. Dual memory neural computer for asynchronous two-view sequential learning. *KDD*, 2018.

[34] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *ICLR*, 2016.

[35] Leonid Libkin, Wim Martens, and Domagoj Vrgoč. Querying graphs with data. *Journal of the ACM (JACM)*, 63(2):14, 2016.

[36] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd annual international conference on machine learning. ACM*, 2016.

[37] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *ICLR*, 2018.

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

[39] Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Column networks for collective classification. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2017.

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

[41] Bharath Ramsundar, Steven Kearnes, Patrick Riley, Dale Webster, David Konerding, and Vijay Pande. Massively multitask networks for drug discovery. *arXiv preprint arXiv:1502.02072*, 2015.

[42] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.

[43] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

[44] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. *NIPS*, 2018.

[45] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Tim Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4974–4983, 2017.

[46] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[47] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems*, pages 926–934, 2013.

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

[49] Mark G Stokes. 'activity-silent' working memory in prefrontal cortex: a dynamic coding framework. *Trends in cognitive sciences*, 19(7):394–405, 2015.

[50] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. *NIPS*, 2015.

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

[52] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *ICLR*, 2017.

[53] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *NIPS*, 2018.