Memory-augmented Dynamic Neural Relational Inference

Dong Gong, Zhen Zhang, Javen Qinfeng Shi, Anton van den Hengel*
The Australian Institute for Machine Learning, The University of Adelaide
https://donggong1.github.io/memdnri

Abstract

Dynamic interacting systems are prevalent in vision tasks. These interactions are usually difficult to observe and measure directly, and yet understanding latent interactions is essential for performing inference tasks on dynamic systems like forecasting. Neural relational inference (NRI) techniques are thus introduced to explicitly estimate interpretable relations between the entities in the system for trajectory prediction. However, NRI assumes static relations; thus, dynamic neural relational inference (DNRI) was proposed to handle dynamic relations using LSTM. Unfortunately, the older information will be washed away when the LSTM updates the latent variable as a whole, which is why DNRI struggles with modeling long-term dependences and forecasting long sequences. This motivates us to propose a memory-augmented dynamic neural relational inference method, which maintains two associative memory pools: one for the interactive relations and the other for the individual entities. The two memory pools help retain useful relation features and node features for the estimation in the future steps. Our model dynamically estimates the relations by learning better embeddings and utilizing the long-range information stored in the memory. With the novel memory modules and customized structures, our memory-augmented DNRI can update and access the memory adaptively as required. The memory pools also serve as global latent variables across time to maintain detailed long-term temporal relations readily available for other components to use. Experiments on synthetic and real-world datasets show the effectiveness of the proposed method on modeling dynamic relations and forecasting complex trajectories.

1. Introduction

Interacting systems are ubiquitous in computer vision, in which the entities influence and restrict each other. The characteristics of each entity are highly correlated with others in various ways [35, 37, 38, 21]. For example, multiple instances in a scene influence each other following certain physical rules or underlying purposes, and human body joints are interacting and are influenced by each other in body movements [21]. Modeling and reasoning the interactive relations are crucial for understanding the dynamic system and can benefit other subsequent tasks, such as behavior prediction. The underlying relations can usually be perceived easily but may not be seen and measured directly [35, 21]. For example, except for the observable skeleton, more latent relations among the human body joints cannot be directly observed. They change dynamically with erratic movements and external influence. Thus, it is difficult to obtain the ground truth of the latent interactions, making dynamic relational inference and prediction challenging.

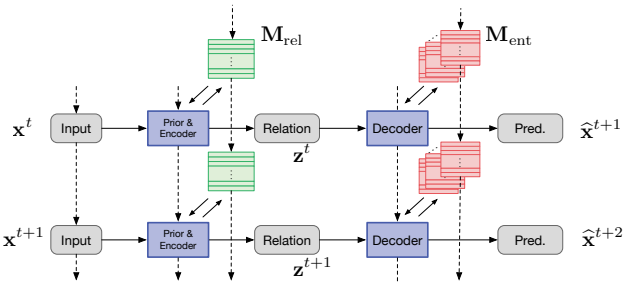


Figure 1. Memory-augmented dynamic neural relational inference. The prior/encoder term is used to dynamically estimate the relation type distribution $\mathbf{z}^t$ for each step, which is used to predict the future via the decoder. We use dual memory pools and the corresponding memory modules to maintain the long-term temporally global information of the relations and entities.

There has been an amount of work proposed to *implicitly* model and learn the interacting system relying on the graph neural networks (GNNs) with messaging passing on the fully connected graph [37, 30, 18, 39, 9, 38] or attention models [3, 27]. Although the implicit relation modeling can benefit the learning, it does not provide much interoperability and powerful prior to the interaction system. Thus, neural relational inference (NRI) [21] was proposed to *explicitly* represent and infer the interaction among the entities in a dynamic system. NRI infers the interaction relations as latent variables and applies them to per-

*This work was partly funded by the Centre for Augmented Reasoning at the Australian Institute for Machine Learning.

form forecasting on a graph defined by the inferred relations. However, NRI [21] assumes the interactive relations are static along the observed trajectory, which is not suitable for many realistic tasks [12]. Dynamic neural relational inference (DNRI) [12] was proposed to estimate the time-step-specific relations dynamically based on a long short-term memory model (LSTM) [17]. The LSTM is learned to model the temporal dynamics for relational inference and trajectory prediction. However, the LSTM based DNRI lacks the ability to capture long-term dependencies and cannot handle the long-range dynamics. The DNRI model stores all the sequential information in a state variable of LSTM, which can only be updated and accessed as a whole. The state is unstructured, and easy to forget the history information [13, 29]. The relation inference and temporal forecasting usually require temporally global information and long-distance correspondences.

In this paper, we propose memory-augmented dynamic neural relational inference (MemDNRI). We formulate the relational prediction as a latent variable model, where latent variables are used to represent the connection type and strength between the entities, similar to NRI and DNRI. We train the model to dynamically infer the relations from the observed sequence (relying on the encoder/prior) and then forecast the unseen trajectories (with the decoder) [12]. Our MemDNRI model maintains two external associative memory pools, *i.e.*, relation memory (RelMem) and entity memory (EntMem), as *temporally global latent variables* to store the long-term information for both the occurring interaction relations and the individual entities, respectively. More than only two memory pools, RelMem and EntMem also indicate two memory-augmented sequential models. They contain the cooperatively learned read and write head and the corresponding controllers for accessing the memory adaptively. Considering the characteristics of the tasks (and subtasks), we customize novel memory modules specifically for RelMem and EntMem for scalability and practicability. At each step, MemDNRI (with RelMem and EntMem) incrementally writes proper knowledge into the memory pools and reads out the most relevant contents for relation estimation and trajectory forecasting.

The main contributions of this paper can be summarized as the following:

- We propose a new memory-augmented neural relational inference method (MemDNRI) for predicting the interaction relations and forecasting trajectories on the graph-structured temporal data. We design MemDNRI as a latent variable model augmented by external associate memory. Unlike the LSTM based DNRI [12] forgetting the long-range dynamic pattern easily, MemDNRI uses memory pools as temporally global latent variables to capture long-term correspondence in the dynamic process.
- We formulate the memory augmentation as dual memory modules (*i.e.*, RelMem and EntMem) for both relation and entity. We design novel structures for the memory modules with customized storage structures, addressing strategies, and read and write head to fulfill the requirements of the tasks (see Sec. 4.1.2).
- The proposed MemDNRI can automatically update and access the memory with proper contents, which naturally uses long-term dependencies. Experiments on multiple synthetic and real datasets show the effectiveness of the proposed method.

2. Related Work

Neural relation inference [21] was proposed to represent the interactions in the dynamic system explicitly. Due to without supervision on the interaction, NRI formulates the problem as a variational autoencoder (VAE) [20, 31], where the latent variable is used to represent the type of interaction relations, *i.e.*, the edges in an initial fully-connected graph. NRI assumes the relations are static along the whole sequence, which DNRI [12] breaks this assumption by modeling the relation prediction with sequential prediction models, *i.e.*, LSTMs. The LSTM based DNRI can dynamically predict specific relations at each step. Some other works extend NRI by applying different techniques such as the factorized graphs [41], additional structural priors [24], and modular meta-learning [2].

Graph neural networks (GNNs) [25, 36, 10] have been generally applied in NRI works [21, 2, 12] to perform encoding and decoding on the structured data, *i.e.*, interactive entities. In NRI and DNRI, a message-passing process similar to [10] is applied. There have also been many variants of the GNNs [26, 43] and message passing methods [37, 38]. GNNs have also been used to model the relational objects [5, 4] with a static graph. Rossi *et al.* [32] proposed a temporal graph network to handle the temporally varying graph, which uses a memory pool to store the messages of the nodes and requires edge ground truth for training.

Memory-augmented neural networks (MANNs) have attracted increasing interest for solving different problems [13, 14, 40, 33, 11, 16, 34]. Graves *et al.* [13] proposed Neural Turing Machine (NTM), which uses external memory to extend the capability of neural networks. NTM uses content-based and location-based attention and memory module to learn programs from examples. Santoro *et al.* [34] propose a memory-augmented relational recurrent neural network (RRNN), which uses self-attention to model the relation between the memory items. In [33], MANN is used to handle one-shot learning problem. In NTM and RRNN, memory is reset at the beginning of each sequence, and the model is trained to learn the ability to process sequence data with memory. In some other methods, such as [11], the memory is learned to store some representative and

prototypical patterns seen during training.

Many prior works have been studied to learn the dynamic system specifically for different applications, such as physical system understanding relying on simulated trajectories [4, 15, 30], and human motion estimation [1, 23]. They are mainly based on the known or assumed graph structure and do not infer the interaction explicitly.

3. Background: Neural Relational Inference

Problem Definition Neural relational inference [21, 12, 2] aims to infer the interaction relations between the entities in a dynamic system and then perform trajectory prediction according to the estimated relations. On the other hand, since there is no ground truth of the relations, the trajectory prediction task can be seen as a surrogate task for the relation prediction. Formally, given input trajectories with V entities from a dynamic system, we let $\mathbf{x}_i^t$ denote the feature vector of the entities $i \in \mathcal{V}, \mathcal{V} = \{1, ..., V\}$ at time step t, such as location and velocity. For simplicity, we let $\mathbf{x}^t = \{\mathbf{x}_1^t, ..., \mathbf{x}_V^t\}$ represent the set of features of all entities at time step t. The V entities can be modeled as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where the vertices $v \in \mathcal{V}$ are the entities and the edges $e = (v, v') \in \mathcal{E}$ represent the relations between the entities. We define a latent variable $\mathbf{z}$ with $\mathbf{z}_{ij} \in \{1, ..., K\}$ to represent the existences and discrete edge types between any two entities $i \in \mathcal{V}$ and $j \in \mathcal{V}, j \neq i$. The model is learned to assign meaning to the relation/edge latent variable $\mathbf{z}$ without explicit supervision.

NRI and DNRI. The general NRI task is to simultaneously learn to predict the edge type $\mathbf{z}_{ij}$ for each edge and learn to predict the trajectory in the future. The task is accomplished by learning a variational auto-encoder (VAE) [31, 20] that maximizes the evidence lower bound (ELBO):

$$\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})), \quad (1)$$

where ϕ and θ are the parameters of the probability distributions. The VAE formulation consists of three main components, *i.e.*, encoder $q_\phi(\mathbf{z}|\mathbf{x})$, prior $p_\theta(\mathbf{z})$, and decoder $p_\theta(\mathbf{x}|\mathbf{z})$. Both static NRI [21] and DNRI [12] are derived from Eq. (1) but define the components in different ways.

In NRI [21], the **encoder** $q_\theta(\mathbf{z}|\mathbf{x})$ is directly formulated as a factorized categorical distribution $q_\theta(\mathbf{z}|\mathbf{x}) = \prod_{i \neq j} q_\theta(\mathbf{z}_{ij}|\mathbf{x})$. Relying on a fully-connected (FC) GNN, the model learns embeddings for all edges and produces the relation type probability for each edge, *i.e.*, each pair of the entities. Given the $p(\mathbf{z}_{ij}|\mathbf{x})$ produced by the encoder, the edge types are sampled from the corresponding concrete distribution [19, 28]:

$$\mathbf{z}_{ij} = \mathrm{softmax}((\mathbf{h}_{ij} + \mathbf{g})/\tau), \quad (2)$$

where $\mathbf{h}_{ij}$ is the predicted logits for $\mathbf{z}_{ij}$, $\mathbf{g}$ is a vector sampled from $\mathrm{Gumbel}(0, 1)$ distribution [19], and τ is a temperature parameter. This sampling strategy makes the model differentiable. NRI [21] learns to obtain a static prediction of the edge type $\mathbf{z}$ for the whole trajectory.

The **prior** $p(\mathbf{z}) = \prod_{i \neq j} p(\mathbf{z}_{ij})$ in NRI is a uniform distribution over relation types or some given distributions according to the assumptions, such as sparsity. To model the dynamically varying relations, DNRI learns an auto-regressive model of the prior distribution. At each time step, the prior is conditioned on the observation and relation prediction from previous steps. DNRI learns to predict the dynamic edge type for each step. In [12], LSTMs are used to formulate the dynamic prior and the encoder.

Given the sampled relation types $\mathbf{z}$, the **decoder** $p_\theta(\mathbf{x}|\mathbf{z})$ is then used to predict the future states of the entities $\mathbf{x}$. NRI assumes the relation types $\mathbf{z}$ remaining static across the observed sequence [12]. Thus decoder is represented as the formulation conditioning on the static $\mathbf{z}$ sampled from the encoder: $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z})$. The decoder is also formulated based on GNN with message passing process determined by the predicted edge type variable $\mathbf{z}$. Since the relation types $\mathbf{z}$ are predicted dynamically in DNRI, the decoder for each step is formulated as an auto-regressive model $p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z}^{1:t})$.

4. Memory-augmented Dynamic Neural Relational Inference

To dynamically estimate the latent interactive relation and predict future states, we propose Memory-augmented Dynamic Neural Relational Inference (MemDNRI). Unlike static NRI [21], we predict a specific relation $\mathbf{z}^t$ for each time step t, as the setting of DNRI. Thus, the model can better understand the dynamic system where the relations change over time. We formulate MemDNRI with the VAE and ELBO formulation in Eq. (1), which consists of three main components, *i.e.*, prior, encoder, and decoder [12].

The DNRI method in [12] dynamically estimates the time-step-specific relations and predicts the future relying on an LSTM (or a GRU) for capturing the temporal dynamics. However, in the LSTM based DNRI method, only a single latent variable is used to maintain the system state and the temporal history information, which can only be updated and read as a whole [29]. The latent state is easy to forget the history information [13], limiting the capability of the LSTM based DNRI on modeling the long-term temporal correspondence (see Fig. 4).

To address the above concerns, we propose using additional external associative memory to capture the long-term correspondence. Specifically, we maintain dual external memory pools with corresponding head and controllers for both the relational inference (with the prior and encoder) and the trajectory prediction (with the decoder), as illustrated in Figure 1. The two memory modules maintain the memory pools storing the knowledge of the relations/edges

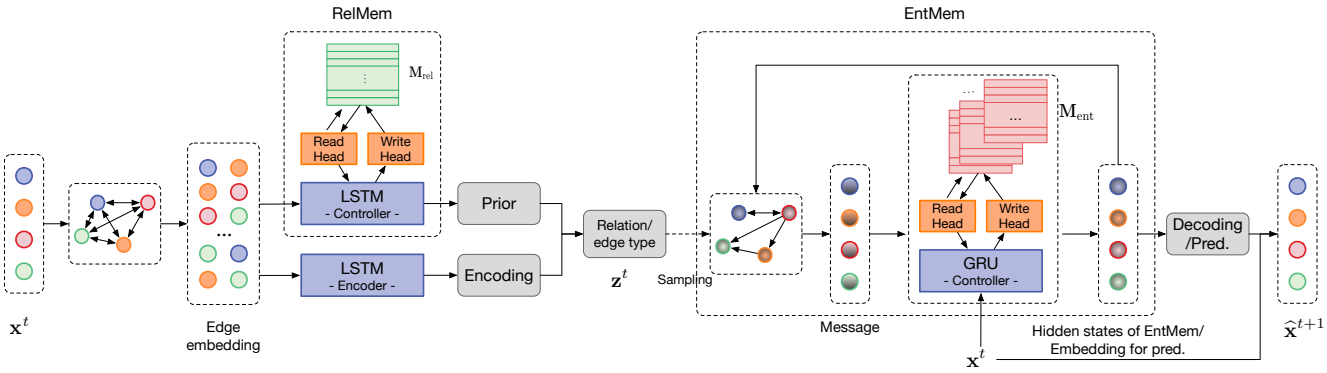


Figure 2. Diagram of the proposed MemDNRI. MemDNRI consists of three main components: prior, encoder and decoder. At each time step t, the input $\mathbf{x}$ is feed into a GNN to obtain edge embeddings on an FC graph. The edge embeddings are then passed to the proposed RelMem module to get the relation embedding, which only takes the history information for dynamic relation estimation. The RelMem module maintains an associative memory $\mathbf{M}_{\text{rel}}$ to store the knowledge of the relation seen in the sequence, which learns the corresponding controller, and read and write heads for automatically updating and accessing the memory pool. The edge type can be sampled from the probabilitic distribution from prior and/or encoder. The estimated edges are used in the GNN in the EntMem based decoder. EntMem maintains a memory pool for the entities and uses it with the GRU based controller. Different to previous MANN, we specifically design novel memory modules for RelMem and EntMem, respectively.

(RelMem) and the entities states (EntMem), respectively.

4.1. Prior Modeling with Relational Memory

To handle the dynamically varying relation, we learn an auto-regressive model of the prior probabilities of the relation type variable $\mathbf{z}$ as [12]:

$$p_\phi(\mathbf{z}|\mathbf{x}) = \prod_{t=1}^{T} p_\phi(\mathbf{z}^t|\mathbf{x}^{1:t}, \mathbf{z}^{1:t-1}), \tag{3}$$

where the relation prior at each step t is conditioned on the previous relations $\mathbf{z}^{1:t-1}$ and the observed inputs $\mathbf{x}^{1:t}$. We implement the prior with a GNN and a memory module, *i.e.*, the relation memory Module (RelMem).

4.1.1 Edge Feature Embedding with GNN

As shown in Figure 2, the input of the feature is firstly feed into a GNN to produce the embedding for each edge of an FC graph:

$$\mathbf{h}_{i,1}^t = f_{\text{emb}}(\mathbf{x}_i^t), \tag{4}$$

$$v \to e: \quad \mathbf{h}_{ij,1}^t = f_{e,1}([\mathbf{h}_{i,1}^t, \mathbf{h}_{j,1}^t]), i \neq j \tag{5}$$

$$e \to v: \quad \mathbf{h}_{j,2}^t = f_{v,1}(\textstyle\sum_{i \neq j} \mathbf{h}_{ij,1}^t), \tag{6}$$

$$v \to e: \quad \mathbf{h}_{ij,\text{emb}}^t = f_{e,2}([\mathbf{h}_{i,2}^t, \mathbf{h}_{j,2}^t]). \tag{7}$$

Eq. (4) - (7) represent the message passing processes on an FC graph [21]. The $f(\cdot)$'s are multilayer perceptron (MLP) for producing embeddings on the entities or edges. $\mathbf{h}_{ij,\text{emb}}^t$ is the output feature embedding on each edge, which is an initial representation of the relation between two entities i and j. For convenience, we represent $\mathbf{h}_{ij,\text{emb}}^t$ as a row vector in the shape of $1 \times C$ and let $\mathbf{h}_{\text{emb}}^t \in \mathbb{R}^{E \times C}$ be the embedding features for all the edges at time step t. We then feed the embedding into the memory module defined in the following to produce prediction of the relation types.

4.1.2 Relation Memory Module

Here, the memory module is not only a memory pool. It is a sequential model that handles the sequential inputs with an external memory pool that can be read and written at each step when required. The memory pool is for maintaining the information in a sequence, which is reset (*i.e.*, initialized as random values) at the beginning of each sequence. In the prior model, the memory module is used to handle each pair of entities' relation embedding. We thus call it a relation memory module (RelMem). The proposed memory module consists of a memory pool, a controller, and a pair of read and write heads, similar to the NTM [13].

The RelMem takes the graph edge embeddings of each pair of nodes from Eq. (7) and modeling the varying of the relations with the memory augmentation:

$$\mathbf{h}_{\text{prior}}^t, \mathbf{s}_{\text{rel}}^t, \mathbf{M}_{\text{rel}}^t = \text{RelMem}(\mathbf{h}_{\text{emb}}^t, \mathbf{s}_{\text{rel}}^{t-1}, \mathbf{M}_{\text{rel}}^{t-1}), \tag{8}$$

where $\mathbf{h}_{\text{prior}}^t$ is the relation prior embedding of all the edges, $\mathbf{s}_{\text{rel}}^t$ and $\mathbf{M}_{\text{rel}}^t$ denote the state variables of RelMem and the memory pool at each time step t, respectively. $\mathbf{M}_{\text{rel}}$ is initialized as empty at $t = 0$ and is gradually updated at each step via the write head. Given $\mathbf{h}_{\text{prior}}^t$ at each step, the prior probability in Eq. (3) output can be obtained via:

$$p_\phi(\mathbf{z}_{ij}^t|\mathbf{x}^{1:t}) = \text{softmax}(f_{\text{prior}}(\mathbf{h}_{ij,\text{prior}}^t)), \tag{9}$$

where the logits $f_{\text{prior}}(\mathbf{h}_{ij,\text{prior}}^t)$ can be used to perform relation type sampling in Eq. (2). The details of the RelMem will be introduced in the following.

Memory controller. The memory controller can be seen as the "entry" of the memory module in Eq. (8). It is defined as an LSTM:

$$\mathbf{q}_{ij,\text{rel}}^t = \text{LSTM}_{\text{rel}}\left([\mathbf{h}_{ij,\text{emb}}^t,\ \mathbf{r}_{ij,\text{rel}}^{t-1}],\ \mathbf{q}_{ij,\text{rel}}^{t-1}\right), \tag{10}$$

where $\mathbf{h}_{ij,\text{emb}}^t$ is the input edge embedding from the GNN, $\mathbf{r}_{ij,\text{rel}}^{t-1}$ is the memory readout from the last step, and $\mathbf{q}_{ij,\text{rel}}^t$ denotes the hidden state of the LSTM. $\mathbf{q}_{ij,\text{rel}}^t$ will be used as the query for accessing the memory.

Memory pool. We define the relation memory pool as a matrix $\mathbf{M}_{\text{rel}} \in \mathbb{R}^{N_{\text{rel}} \times M_{\text{rel}}}$, which is shared by all the edges (*i.e.*, all the entity pairs (i, j)'s) for scalability.

Memory reading. Given the memory pool and the addressing weights $\mathbf{w}_{ij,\text{rel}}^t$ for reading, we can read out from the memory via:

$$\mathbf{r}_{ij,\text{rel}}^t = \mathbf{w}_{ij,\text{rel}}^t \mathbf{M}_{\text{rel}}^{t-1} \tag{11}$$

Memory writing considering the addressing conflicts. Apart from the query key, the write head also generates erase vector $\mathbf{e}_{ij,\text{rel}}^t \in [0,1]^{1\times M_{\text{rel}}}$ and add vector $\mathbf{a}_{ij,\text{rel}}^t \in \mathbb{R}^{1\times M_{\text{rel}}}$ for each edge (i, j). For each $\mathbf{h}_{ij,\text{emb}}^t$, we can get updating items for the memory as

$$\mathbf{E}_{ij,\text{rel}}^t = \mathbf{w}_{ij,\text{rel}}^{t\mathsf{T}} \mathbf{e}_{ij,\text{rel}}^t, \ \text{ and } \ \mathbf{A}_{ij,\text{rel}}^t = \mathbf{w}_{ij,\text{rel}}^{t\mathsf{T}} \mathbf{a}_{ij,\text{rel}}^t, \tag{12}$$

which reflect the updating information from each elements (i, j) for the memory. Relying on $\mathbf{E}_{ij,\text{rel}}^t$ and $\mathbf{A}_{ij,\text{rel}}^t$, the memory can be updated via:

$$\mathbf{M}_{\text{rel}}^t = (\mathbf{1} - \mathbf{E}_{ij,\text{rel}}^t) \odot \mathbf{M}_{\text{rel}}^{t-1} + \mathbf{A}_{ij,\text{rel}}^t, \tag{13}$$

where $\odot$ denotes elementwise product operation. Note that read and write heads do not share addressing weights.

However, the memory pool $\mathbf{M}_{\text{rel}}$ is shared by all the edges, and we need to write multiple elements (corresponding to multiple edges (i, j)'s) into the memory at one step, which may cause conflicts since the write head is not aware other edges while producing writing addressing weights $\mathbf{w}_{ij,\text{rel}}^{t\mathsf{T}}$ for each edge simultaneously. The addressing conflicts often cause gradient exploding, making training difficult. Considering that the memory is mainly for capturing the temporal correspondence, we can consider giving each edge an individual memory. However, the edge number of an FC graph $E = V \times (V - 1)$ could be very large, making computation impractical. To handle the addressing conflicts, we introduce to normalize the addressing weights $\mathbf{w}_{\text{rel}}^t$ by considering the "hitting rate" of each memory slots:

$$\mathbf{w}_{ij,\text{rel}}^t = \mathbf{w}_{ij,\text{rel}}^t \odot \frac{\mathbf{w}_{ij,\text{rel}}^t}{\sum_{i\neq j} \mathbf{w}_{ij,\text{rel}}^t}, \tag{14}$$

where $\odot$ denotes the element-wise product and the division is also element-wise. $\mathbf{w}_{ij,\text{rel}}^t / \sum_{i\neq j} \mathbf{w}_{ij,\text{rel}}^t$ is used to represent the "hitting rate" of each memory slot at same time.

Asymmetrical memory addressing. Given a query $\mathbf{q}_{ij,\text{rel}}^t$, the *read head* and *write head* will generate keys $\mathbf{k}_{ij,\text{rel}}^t \in \mathbb{R}^{1\times M_{\text{rel}}}$ for addressing the memory for reading and writing operation, respectively. Given a key $\mathbf{k}_{ij,\text{rel}}$, we firstly can obtain the memory addressing weights $\mathbf{w}_{ij,\text{rel}} \in \mathbb{R}^{1\times N_{\text{rel}}}$ relying on the content-based attention:

$$\mathbf{w}_{ij,\text{rel}}(i') = \frac{\exp(d(\mathbf{k}_{ij,\text{rel}}, \mathbf{m}_{i'}))}{\sum_{j'=1}^{N_{\text{rel}}} \exp(d(\mathbf{k}_{ij,\text{rel}}, \mathbf{m}_{j'}))}, \tag{15}$$

where $\mathbf{w}_{ij,\text{rel}}(i')$ denotes the i'-th elements in $\mathbf{w}_{ij,\text{rel}}$ and $d(\cdot,\cdot)$ is cosine similarity.

Different from many MANNs [13, 14] applying the same addressing strategy for reading and writing, we propose to use *asymmetrical addressing* for the two heads. Except for the content addressing in Eq. (15), we also learn the location-based addressing operations [13] for the write head to facilitate it shifting between the memory slots, letting it make better use of the memory slots during the step-by-step writing. On the other hand, we only apply the content-based addressing for the read head and ignore the location since the state at one step may be correlated to any previous step.

Obtaining prior embedding. At each time step, RelMem performs reading and writing on the memory $\mathbf{M}_{\text{rel}}$ once. After getting $\mathbf{r}_{ij,\text{rel}}^t$, we obtain the prior embedding by merging the readout and the LSTM controller output: $\mathbf{h}_{ij,\text{prior}}^t = f_{\text{rel-merge}}\left([\mathbf{r}_{ij,\text{rel}}^t,\ \mathbf{q}_{ij,\text{rel}}^t]\right)$, where $[\cdot,\cdot]$ is a concatenation operation and $f_{\text{rel-merge}}(\cdot)$ is an MLP.

4.2. Encoder

Encoder approximates the distribution of the relations at each time step by looking at the information from the whole sequence (not limited to the history information). Following DNRI [12], a reverse LSTM [22, 8] is used to capture the future state of sequence, which takes the initial edge embedding $\mathbf{h}_{ij,\text{emb}}^t$ as input: $q_\phi(\mathbf{z}_{ij}^t|\mathbf{x}^{1:t}) = \text{softmax}(f_{\text{enc}}([\mathbf{h}_{ij,\text{enc}}^t, \mathbf{h}_{ij,\text{prior}}^t]))$, where $\mathbf{h}_{ij,\text{enc}}^t = \text{LSTM}_{\text{enc}}\left(\mathbf{h}_{ij,\text{emb}}^t,\ \mathbf{h}_{ij,\text{enc}}^{t-1}\right)$. The encoder is only used to guide the learning during training. During testing, only the prior is used to predict relation types for each edge dynamically. For simplicity, we do not use memory module in the modeling of the encoder.

4.3. Decoder with Entity Memory

The decoder is used to predict the trajectory given the observed entities and sampled relation types. Similar to the prior model in Eq. (3), the decoder can also be written as an auto-regressive model: $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^T p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z}^{1:t})$. We implement the decoder as a memory-augmented module with the GNN message passing [21, 12] to process the hidden variable and a GRU [6] as the controller, with the same configuration in [21, 12].

Since the task of the decoder is to perform prediction on the entities, we maintain the memory pool to store the information of the entities observed in the sequence. Thus the memory module in the decoder is referred to as entity memory, *i.e.*, EntMem. We define $\mathbf{M}_{\text{ent}}$ to denote the memory pool for the EntMem. As shown in Figure 2, at each time step t, EntMem takes the entity features $\mathbf{x}^t$ as input and predict the entity distribution for the next step:

$$\boldsymbol{\mu}^{t+1}, \mathbf{h}_{\text{ent}}^{t+1}, \mathbf{M}_{\text{ent}}^{t+1} = \text{EntMem}(\mathbf{x}^t, \mathbf{h}_{\text{ent}}^t, \mathbf{M}_{\text{ent}}^t, \mathbf{z}^t), \quad (16)$$

$$p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z}^{1:t}) = \mathcal{N}(\boldsymbol{\mu}^{t+1}, \sigma^2\mathbf{I}), \quad (17)$$

where $\mathbf{h}_{\text{ent}}^t$ denotes the state of the GRU controller in EntMem, and the decoder probability is modeled as a Gaussian distribution and $\boldsymbol{\mu}^t$ denotes the mean variable of the decoder probability. EntMem reads and updates (writes to) the memory $\mathbf{M}_{\text{ent}}$ at each step. At each step, EntMem performs GNN based message passing for the hidden variables on the entities/nodes. EntMem module and the corresponding memory pool are designed according to the characteristics of the task of the decoder, which is different from the RelMem in prior modeling.

4.3.1 Message Passing for Entity Feature Embedding

The hidden states per entity are firstly passed into a GNN to produce entity feature embeddings according to the graph structure and the edge type $\mathbf{z}^t$ predicted from the prior model:

$$v \to e: \quad \mathbf{h}_{ij,\text{ent}}^t = \sum_k z_{ij,k}^t \tilde{f}_{e,k}([\mathbf{h}_{i,\text{ent}}^t, \mathbf{h}_{j,\text{ent}}^t]), \quad (18)$$

$$e \to v: \quad \mathbf{m}_j^t = \textstyle\sum_{i \neq j} \mathbf{h}_{ij,\text{ent}}^t, \quad (19)$$

where the edge type $\mathbf{m}_j^t$ is message from the entity's neighbor and will be used as the entity feature embedding, $\mathbf{z}_{ij}^t$ is represented as a vectorized distribution, $z_{ij,k}$ denotes the probability of assigning the k-th edge type to the edge (i, j). Eq. (18) and (19) show one round of the two-step message passing in the GNN, which can be repeated several times to perform more rounds of message passing.

4.3.2 Entity Memory Module with GRU

After obtaining the messages $\mathbf{m}^t$ for all the entities, we feed them into a GRU with the input features $\mathbf{x}^t$ and the memory readout $\mathbf{r}_{\text{ent}}^{t-1}$ from the previous step:

$$\mathbf{h}_{j,\text{ent}}^{t+1} = \text{GRU}([\mathbf{m}_j^t, [\mathbf{x}_j^t, \mathbf{r}_{j,\text{ent}}^t]], \mathbf{h}_{j,\text{ent}}^t), \quad (20)$$

where $[\cdot, \cdot]$ denotes the concatenation operator, $\mathbf{x}_j^t$ and $\mathbf{r}_{j,\text{ent}}^t$ are concatenated together as the input for the GRU. We take the hidden state $\mathbf{h}_{j,\text{ent}}^{t+1}$ as the query $\mathbf{q}_{j,\text{ent}}^{t+1}$ to access the entity memory $\mathbf{M}_{\text{ent}}$.

Entity-wise memory pool and memory accessing. Different to the relation memory, we define the memory pool for the entities as a 3D tensor $\mathbf{M}_{\text{ent}} \in \mathbb{R}^{V \times N_{\text{ent}} \times M_{\text{ent}}}$. Considering that the entity number is not too large to cause much computation burden, we assign each entity a memory pool to directly avoid addressing conflicts that may happen while writing the memory, as discussed in Sec. 4.1.2. Such entity-wise memory pool design also enables us to use small N_{ent} to achieve better modeling ability. We also apply the *asymmetrical addressing* strategy for EntMem. Thus the memory addressing, reading, and writing are designed to have the similar operation used in RelMem but are implemented as an entity-wise version, which means the i-th entity will access the i-th slice of the memory tensor, $\mathbf{M}_{\text{ent},(i,:,:)}$.

Obtaining the decoding/prediction embedding. Given $\mathbf{q}_{\text{ent}}^{t+1}$ for all the entities, the read head can obtain the readout $\mathbf{r}_{\text{ent}}^{t+1}$. For each entity, we obtain the prediction via $\boldsymbol{\mu}_i^{t+1} = f_{\text{dec-pred}}(\mathbf{h}_{i,\text{dec}}^{t+1}) + \mathbf{x}_i^t$, where $\mathbf{h}_{i,\text{dec}}^{t+1} = f_{\text{ent-merge}}\left([\mathbf{r}_{i,\text{ent}}^{t+1}, \mathbf{h}_{i,\text{ent}}^{t+1}]\right)$, and $f_{\text{ent-merge}}(\cdot)$ and $f_{\text{dec-pred}}(\cdot)$ are implemented as MLP.

4.4. Training and Inference

During training, we learn the parameters for the encoder, prior and decoder, including the parameters of the GNNs and the memory modules. Note that the memory pools are not learned, which are initialized as "empty" (with random values) at the beginning of each sequence and then updated via the writing head while processing the sequences. The memory heads are trained to conduct the memory manipulations according to the current states and observations. The ground truth states are provided to the decoder as input during training. our model is trained relying according to the ELBO in Eq. (1). The reconstruction term $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ is estimated as $-\sum_j \sum_{t=2}^T \frac{||\mathbf{x}_j^t - \boldsymbol{\mu}_j^t||^2}{2\sigma^2} + \text{const}$. The KL divergence is computed as in [12].

In the testing phase, we use the prior to predict the relation types between the entities and then use the decoder to perform prediction of the trajectory states. At the beginning of each sequence, the memory pools and initial states for both RelMem and EntMem are all reset as random values.

4.5. Implementation Details

The MLPs for feature embedding and message passing are all two-layer MLPs with 256 hidden units and output dimensions, and the ELU activation. The LSTMs used in the encoder and prior models are all single-layer LSTMs with 64 hidden units.

In both memory modules, we use single linear embedding to produce the addressing keys from the query vectors. In RelMem, all the edges share the same memory. We set the memory size as $N_{\text{rel}} = 128$ and $M_{\text{rel}} = 16$, which is

a balance between the representation capability and computation resources. In EntMem, since the entities have the specific memory pools, the memory size parameters are set as $N_{\text{ent}} = 32$ and $M_{\text{ent}} = 16$. The contents stored in the memory can be seen as a compressed encoding of the observation and history. Models are trained using Adam with the learning rate as 5×10^{-4}.

5. Experiments

We conduct experiments on both synthetic and realistic data. In the experiments, we mainly compare with the previous state-of-the-art NRI and DNRI, which are also the most related work to us. For continence, we use the same parameter configuration, as introduced above, for all experiments.

5.1. Synthetic Physics Simulation

We first evaluate the models with a physical simulation system from [12], *i.e.*, moving particles with dynamic relations. In the system, each trajectory consists of three particles. Two of them move with a constant velocity in some direction. And the third (green in Figure 3) is initialized with a random velocity but is pushed away by others when they are too close (with a distance smaller than a threshold). In this system, part of the relations between the particles dynamically changes depending on the distances.

Since the ground truth relation types are available in the synthetic dataset, we can directly evaluate the relation type accuracy. Table 1 shows the relation prediction accuracy and the trajectory prediction mean squared error (MSE) and the comparison with NRI [21] and DNRI [12]. Figure 3 visualizes one example of the prediction results. The results show that MemDNRI can achieve a significant improvement on both the relation estimation and the prediction.

Table 1. Results on physics data [12] with more dynamic relations

	Pred. MSE ($\times 10^{-4}$)			Relation est. (2 types)			
	Step #1	#15	#25	Prec.	Rec.	F1	Acc.
NRI	0.199	15.0	39.3	0.21	0.50	0.29	0.90
DNRI	0.177	3.48	5.36	0.76	0.46	0.57	0.97
MemDNRI	**0.126**	**2.98**	**4.56**	**0.91**	**0.54**	**0.68**	**0.98**

How the memory augments the long-term sequential modeling. To show how the memory works, we analyze how the write operations correlate to the read operations in the following steps by taking RelMem learned on the physics data as the example. Given any two steps t and t' with $t' > t$, we define the read-write correlation score as $s_{(t,t')} = \sum_{ij,i'j'} \mathbf{w}^{t}_{ij,\text{w}} \mathbf{w}^{t'\text{T}}_{i'j',\text{r}}$, which approximately denotes how the write operation at t influences the read out at t'. Figure 4(a) shows that the write operation at one step (*i.e.*, each curve starting at one step) can enduringly correlate to the following steps via the memory and may play an important role after many steps. According to Figure 4(a), we can obtain the most (and 2nd) correlated write for each step as shown in Figure 4(b). At early steps, the read operations are highly correlated to the latest writing operations since most memory slots are "empty". We can see, at the end of the prediction, when the particle has been pushed away to a situation very similar to the beginning stage, the memory pool helps the model to estimate the relations using the long-range correspondence, *i.e.*, recalling the correlated features seen at very early steps. The analyses show the memory can capture interpretable and meaningful long-term correspondence.

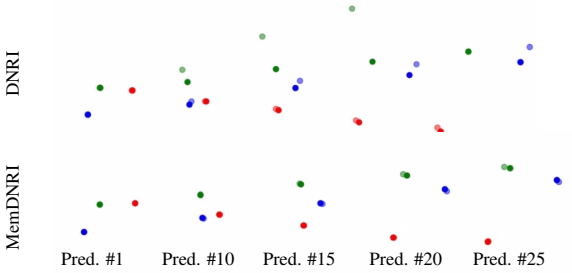
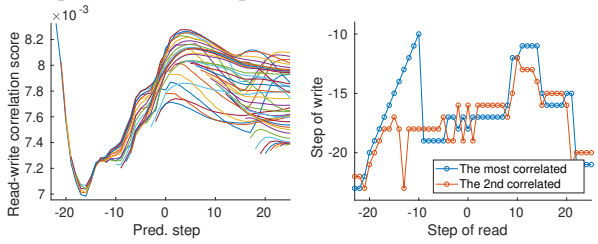


Figure 3. Visualization the results from dynamic physics simulation data [12]. The solid points denote the ground truth, and the dashed points indicate the predicted location.



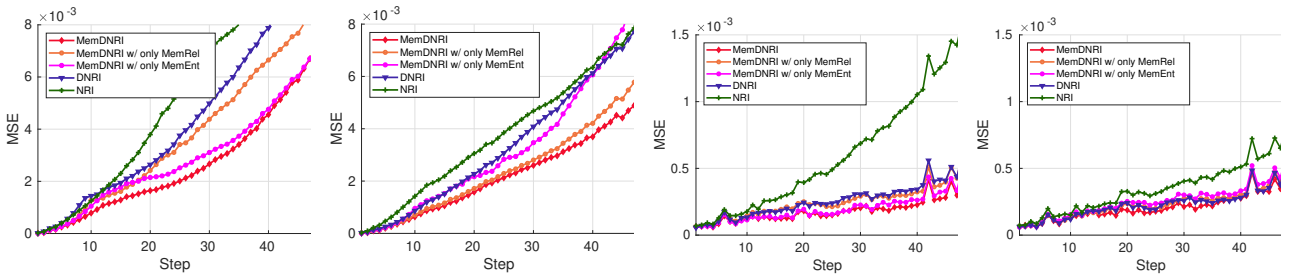
(a) Read-write correlation score (b) The most correlated write

Figure 4. Visualizing how the memory maintains and deliveries the sequential correspondence. (a) Each curve visualizes the read-write correlation of a writing operation and the read in the following steps, which shows how each writing operation influences the following read. (b) The curves show the most and second correlated write step for each read step. We denote the first prediction step as #1, and the observed burn-in steps are negative.

5.2. Motion Capture Data

We then study the motion capture data from the CMU motion capture databases [7]. Following [21, 12], we focus on the experiments on two subjects, #35 and #118. #35 consists of walking trajectories, in which the human skeletons and joints move slowly and stably. The relation inference and prediction are easier. #118 consists of trials where the subject stands stationary and then jumps forward quickly. Prediction on #118 is more challenging, and thus, moving trajectories are usually determined by subtle movements in temporally global structures, requiring a stronger ability to capture long-term dependency. Following the standard setting in [21], we train the model using the sequences of length 50 and evaluate the sequences of length 99 by giving the first 49 frames. In our experiments, we train and evaluate the model on the two datasets using the same protocol.

Figure 5 shows the results on #35 and #118 of different



(a) #118, 2 relation type (b) #118, 4 relation type (c) #35, 2 relation type (d) #35, 4 relation type

Figure 5. MSE for trajectory prediction on the motion capture data. The curves show an average of the results from 3 different initialization.

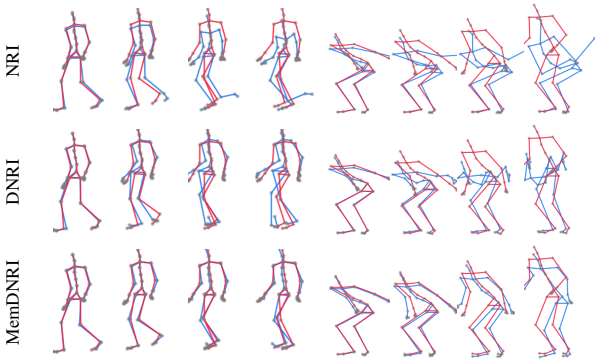


Figure 6. Motion prediction example from NRI [21], DNRI [12], and MemDNRI. Left 4 columns: step #10, #25, #40, and #47 from motion #35 (2 rel.); Right 3 columns: step #10, #20, #30, and #40 from motion #118 (4 rel.). Red: ground truth; Blue: the prediction.

methods with 2 or 4 edge types. The results are the average of the 3 times trial with different random initializations. We compare the proposed MemDNRI with the state-of-the-art NRI [21], and DNRI [12]. We also compare our full model with RelMem (in the prior modeling) and the EntMem (with the decoder) and other model variants that only have RelMem or EntMem.

Figure 5(a) and 5(b) shows the results on #118. Instead of evaluating with the sliding window and averaging way in [13], we directly evaluate the model to do the prediction after seeing the burn-in steps. MemDNRI brings significant improvement on #118 and performs more stable than previous DNRI and NRI with the increasing of the step numbers (*e.g.*, step #40). Due to there are more irregular movements in #118, the proposed MemDNRI can capture the long-term global dependency more easily for handling the complex prediction. Figure 6 shows that MemDNRI can predict good results even at very late frames (frame 48) in #118, since the memory augmentation helps to utilize more global and history information to predict the challenging future stats. Figure 5(c) and 5(d) show the prediction MSE on #35. All the methods perform better with 4 edge types. The improvements from MemDNRI are milder than that on #115, since #35 is stable and easy to forecast. Nevertheless, we can still observe the improvement from MemDNRI visually in Figure 6.

Ablation study. In Figure 5, we show the results of several variants of the proposed full model, *i.e.*, the MemDNRI with only the RelMem or the MemDNRI with only the EntMem, in which the plain prior LSTM or GRU based decoder are used to replace the model with memory augmentation. We can see both of these two modules contribute to the results. As shown in the result curves, the memory module on the decoder, *i.e.*, EntMem, seems to influence the future prediction slightly more than RelMem.

5.3. Basketball Data

We conduct experiments on the basketball player trajectory data [42]. Each trajectory contains the 2D positions of 5 offensive team players, which is processed into 49 frames spanning 8 seconds of play. All models are trained to predict 19 steps by observing the first 30 frames. Figure 7 shows the MSE of the trajectory prediction. The memory-augmented models obtain obvious improvements on the prediction results.

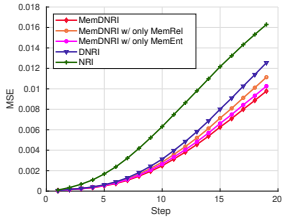


Figure 7. MSE of the prediction on the basketball player dataset [42]. The curves show an average of the results from 3 different initialization.

More results, visualizations, and the details of the method are let in the supplementary material.

6. Conclusion

This paper proposed a memory-augmented method for dynamic neural relation inference and prediction (MemDNRI), which predicts the relation types between the interactive entities and then uses the predicted relation to forecast the future. We proposed two memory modules to maintain the long-term information for the interaction relation (RelMem) and the entities (EntMem) for relation prediction and forecasting. The designs of the two memory modules are explicitly tailored according to the different purposes.

References

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016.

[2] Ferran Alet, Erica Weng, Tomás Lozano-Pérez, and Leslie Kaelbling. Neural relational inference with fast modular meta-learning. *Advances in Neural Information Processing Systems*, 2019.

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

[4] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[5] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *International Conference on Learning Representations*, 2017.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] CMU. *Carnegie-Mellon Motion Capture Database*, 2003.

[8] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pages 2199–2207, 2016.

[9] Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*, 2017.

[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[11] Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton van den Hengel. Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1705–1714, 2019.

[12] Colin Graber and Alexander G Schwing. Dynamic neural relational inference. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 8513–8522, 2020.

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

[14] 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.

[15] Nicholas Guttenberg, Nathaniel Virgo, Olaf Witkowski, Hidetoshi Aoki, and Ryota Kanai. Permutation-equivariant neural networks applied to dynamics prediction. *arXiv preprint arXiv:1612.04530*, 2016.

[16] Tong He, Dong Gong, Zhi Tian, and Chunhua Shen. Learning and memorizing representative prototypes for 3d point cloud semantic and instance segmentation. In *European Conference on Computer Vision*, pages 564–580, 2020.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017.

[19] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *International Conference on Learning Representations*, 2017.

[20] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[21] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2688–2697, 2018.

[22] Rahul G Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[23] Hoang M Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated multi-agent imitation learning. *arXiv preprint arXiv:1703.03121*, 2017.

[24] Yaguang Li, Chuizheng Meng, Cyrus Shahabi, and Yan Liu. Structure-informed graph auto-encoder for relational inference and simulation. In *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

[25] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[26] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pages 4255–4265, 2019.

[27] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[28] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[29] Francesco Marchetti, Federico Becattini, Lorenzo Seidenari, and Alberto Del Bimbo. Mantra: Memory augmented networks for multiple trajectory prediction. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7143–7152, 2020.

[30] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[31] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[32] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML Workshop on Graph Representation Learning*, 2020.

[33] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

[34] 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. In *Advances in neural information processing systems*, pages 7299–7310, 2018.

[35] Adam Santoro, David Raposo, David GT Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *Advances in Neural Information Processing Systems*, 2017.

[36] 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, 2008.

[37] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29:2244–2252, 2016.

[38] Sjoerd Van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[40] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint*, arXiv:1410.3916, 2014.

[41] Raymond A Yeh, Alexander G Schwing, Jonathan Huang, and Kevin Murphy. Diverse generation for multi-agent sports games. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4610–4619, 2019.

[42] Eric Zhan, Stephan Zheng, Yisong Yue, Long Sha, and Patrick Lucey. Generating multi-agent trajectories using programmatic weak supervision. *arXiv preprint arXiv:1803.07612*, 2018.

[43] Zhen Zhang, Fan Wu, and Wee Sun Lee. Factor graph neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.