Extending Neural Relational Inference

COMP9417 2023 T2 Project

Chin Pok Leung Lejiang Yang Russell Ung
Yang Song Fengyi Shen

1. Introduction

Dynamical systems have long fascinated researchers due to their inherent complexities and intriguing patterns. While the beauty of these systems lies in their interconnectivity and interactions, unraveling their mysteries requires an understanding of the countless components and their relations. Traditionally, researchers have employed mathematical models and classical algorithms to decode the patterns embedded within dynamical systems. However, with the rise of neural networks and the growing capabilities of machine learning, newer methods are emerging to tackle the challenges posed by these systems.

In this context, the "Neural Relation Inference for Interacting Systems" (NRI) [1] stands as a promising model. Its core strength lies in its ability to predict the trajectories of dynamical systems solely based on observations. This unique capability emerges from the intelligent blend of various neural network architectures, including the Graph Neural Network (GNN), Variational Autoencoder (VAE), and Recurrent Neural Network (RNN). Such a composite model allows it to capture both the static and dynamic nuances of interacting systems.

But like any model, there is always room for improvement and innovation. This report aims to delve deeper into the original architecture and experiment with augmentations that can potentially enhance its predictive power. Specifically, we will investigate the effectiveness of introducing learnable positional encoding and also replacing GNN in the decoder with Graph Neural Diffusion [2].

To provide a tangible context for our experiments, we will utilize one of the evaluation tasks of [1], the CMU Motion Capture Database [3], focusing on the walking category of subject #35. This dataset, rich with human motion dynamics, will serve as an ideal playground to test the robustness and accuracy of our augmented models. Furthermore, by reusing the code of the base model, we ensure that our enhancements and changes remain consistent and comparable.

Through the following sections, we will delve into the intricacies of the base model, explore the proposed extensions, and detail our experimental setup, ultimately culminating in a

discussion of the results and their implications.

2. Preliminaries

2.1. Graph Neural Networks

Graph neural networks (GNN) are neural architectures designed to handle graph data, which consists of nodes and edges. Standard neural network architectures, like feed-forward and convolutional networks are not directly applicable to graph data due to its non-Euclidean nature.

Central to GNN operation is the idea of message passing. Nodes aggregate information from their neighbors. This aggregation is usually done using operations like sum, mean or max. After aggregating the messages, nodes update their embeddings, which are vector representations capturing the node's information and its context in the graph. Just like deep learning models, GNNs can have multiple layers, each propagating and transforming the node embeddings.

GNNs operate on graph structures $G = (V, E)$ where V is the set of nodes and E is the set of edges. Each node $v \in \mathrm{V}$ carries features $x_v(t)$ representing its state at time t. The primary goal of a GNN is to learn a function that maps nodes to a vector space. This function is constructed in a way that node embeddings capture the graph's topological structure and nodes' features.

A commonly used method in GNNs is the message-passing framework:

$$h_v^{(k+1)} = \sigma\left(W^{(k)} \cdot \mathrm{AGGREGATE}^{(k)}\left(\left\{h_u^{(k)} \middle| u \in N(v)\right\}\right)\right)$$

Where $h_v{}^{(k)}$ is the feature vector of node v at iteration k, $N(v)$ is the neighboring node of v, AGGREGATE is an aggregation function, $W^{(k)}$ is a weight matrix at iteration k and σ is a non-linear activation function.

In NRI [1], we consider GNN on a directed complete graph without self-loops. A learned neural network f_{e} is first used to compute the edge features from the concatenated features of the two connected nodes. Then apply average pooling to the connected in-edges as denoted below,

$$\mathrm{AGGREGATE}^{(\mathrm{k+1})} = \frac{1}{\deg^-(v)} \sum_{u \in N^-(v)} f_{\mathrm{e}}\left(h_v^{(k)} \parallel h_u^{(k)}\right) .$$

Where $\deg^-(v)$ is the in-degree of v, $N^-(v)$ is the set of in-neighbors of v, and $\cdot\|\cdot$ denotes the concatenation operator (which the authors of [1] refer to as `node2edge`).

This aggregation function is refered to as `edge2node` in [1]. The above aggregator can also be interpreted as an average aggregation over the in-edges of a vertex.

2.2. Variational Autoencoders

VAEs are generative models that learn to encode and decode data in a way that allows for easy and structured generation of new data points. Given an input data point, the encoder produces a distribution over the latent variables. The latent space is the compressed representation of the data in a continuous space. It's described by parameters of mean and variance output by the encoder. From a sample in latent space, the decoder reconstructs the original data.

For example, given data x, the encoder models the posterior $q_\phi(z \mid x)$, where z is a latent variable and ϕ represents the encoder's parameters. Ideally, this posterior should be close to the true posterior $p_\theta(z \mid x)$, where θ represents the model parameters.

VAEs are trained using a combination of reconstruction loss and a Kullback-Leibler divergence (KL-divergence) term which ensures the latent space has good properties for generative tasks. These allow us to calculate the loss function called the Evidence Lower Bound (ELBO) and it can be written as:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \mathsf{KL}\left(q_\phi(z|x) \parallel p(z)\right)$$

Where $\mathbb{E}$ is the expectation, $\mathsf{KL}(\cdot\|\cdot)$ is the KL-divergence of 2 probability distributions and $p(z)$ is a prior distribution of the latent variable z, usually taken as a standard normal distribution. The expected log likelihood term is the reconstruction loss. The KL-divergence measures the distance between 2 distributions, it acts as a regularization term to ensure the distribution of z is close to our prior.

In NRI [1], our encoder $q_\phi\big(z(t)\big|x(1), \ldots, x(T)\big)$ is given a trajectory of T observations of all entities in the system $x(1), \ldots, x(T)$ and predicts the relation type $z(t)$ between each pair of vertices at time t. Here, z is a discrete categorical distribution which we estimate using softmax.

Our decoder $p_\theta(x|z) = \prod_{t=2}^{T} p_\theta\big(x(t)\big|x(1), \ldots, x(t-1), z(t)\big)$ autoregressively predicts the next observation given the first observation $x(1)$ and the predicted relation type $z(t)$. We assume that the distribution of $p_\theta(x|z)$ is Gaussian.

Our model minimizes the ELBO loss as the objective,

$$\min_{\theta,\phi} -\log p_\theta(x|z) + \mathsf{KL}\left(q_\phi(z\mathrm{x}) \parallel p(z)\right)$$

Where we use the Gaussian negative log likelihood for the log likelihood term:

$$\min_{\theta,\phi} -\log p_\theta(x|z) = -\min_{\theta,\phi} -\log\left(\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right) = \min_{\theta,\phi}\frac{(x-\mu)^2}{\sigma^2}$$

We used a constant value of $\sigma^2 = 5 \cdot 10^{-5}$. Note that the objective resembles a scaled mean squared error.

For the categorical KL-divergence loss:

$$\mathrm{KL}\left(q_\phi(z|x) \parallel p(z)\right) = \sum_{i=1}^{etype} q_\phi(z|x)\log\frac{q_\phi(z|x)}{p(z)}$$

Where $etype$ is the number of relation types.

2.3. Gated Recurrent Unit

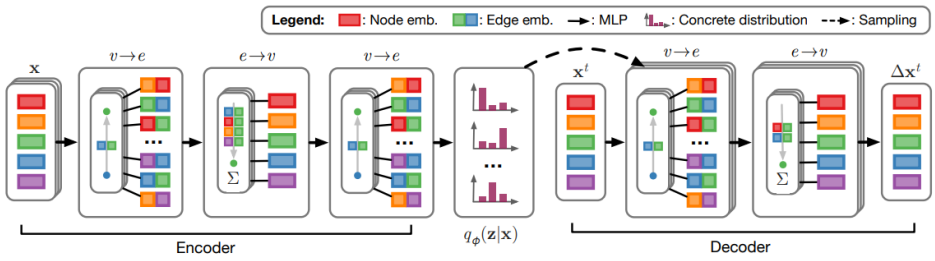
The Gated Recurrent Unit [4], commonly referred to as GRU, is a recurrent neural network (RNN) variant introduced to address the vanishing gradient problem experienced by traditional RNNs. This facilitates the GRU's capability to efficiently learn long-term dependencies.

3. Neural Relational Inference

The neural relational inference (NRI) model in [1] predicts the future states of a dynamical system with observations in the past. Given a trajectory of past observations of a dynamical system $x(1), \ldots, x(t)$, the goal of NRI is to:

1) forecast the future state of the system $x(t+1), \ldots, x(t+N)$,
2) Infer the relationship between all possible ordered tuples of entities in the system (edge types in the graph) such that the predicted result is explainable.

The model consists of an encoder and a decoder. The encoder and the decoder are jointly trained. The architecture of the model is shown in the below figure [1].



3.1. Encoder

The encoder predicts the probability distribution of the relation type of all possible pairs of entities (vertices) based on a trajectory of observations in the past.

The encoder is modelled as

$$q_\varphi(z_{ij}|X) = \text{softmax}(f_{enc}, \varphi(X)_{ij}, 1{:}K)$$

where $f_{enc}, \varphi(X)$ represent a GNN on a fully connected graph with no self-loops.

The encoder first computes the embeddings from the input trajectory:

$$h_j^1 = f_{emb}(x_j)$$

Then followed by one round of edge aggregation:

$$v \to e{:}\ h_{(i,j)}^1 = f_e^1([h_i^1, h_j^1])$$

$$e \to v{:}\ h_j^2 = f_v^1\left(\sum_{i \neq j} h_{(i,j)}^1\right)$$

Finally, compute the edge embeddings and compute the logits of the distribution of relation types for each edge in the graph.

$$v \to e{:}\ h_{(i,j)}^2 = f_e^2([h_i^2, h_j^2])$$

φ summarizes the parameters of the neural networks of above equations. embedding $h_{(i,j)}^1$ depends on x_i and x_j, and h_j^2 uses information from the whole graph. And $f(...)$ are neural networks that map between the respective representations. We used MLPs for the $f(...)$.

3.2. Sampling

To use reparameterization techniques for gradient calculation on discrete distributions, we transform the sampling of discrete variables into sampling from a continuous distribution and obtain the gradient by applying reparameterization techniques on the continuous distribution. [1] used concrete distributions to approximate discrete distributions:

$$z_{i,j} = \text{softmax}\left(\frac{h_{(i,j)}^2 + g}{\tau}\right)$$

Where $g \in \mathbb{R}^{etypes}$ is a random variable and τ is the softmax temperature.

Randomness is introduced through the random variable $g \sim \text{Gumbel}(0,1)$. We use hard softmax to sample z as an one-hot vector.

3.3. Decoder

The decoder autoregressively predicts the future continuation of the interacting system's dynamics based on the relational type of the vertex pairs and the last observation of the system.

Any GNN algorithm can be used as a decoder.
The state is location and velocity, and z is the ground-truth graph we have:

$$p_\theta(x^{t+1}|x^t, \dots, x^1, z|)$$

[1] run one round of edge aggregation; The only difference is that we have a separate MLP for each edge type.

$$v \to e: \quad \tilde{\mathbf{h}}_{(i,j)}^t = \sum_k z_{ij,k} \tilde{f}_e^k([\mathbf{x}_i^t, \mathbf{x}_j^t])$$
$$e \to v: \quad \boldsymbol{\mu}_j^{t+1} = \mathbf{x}_j^t + \tilde{f}_v(\textstyle\sum_{i \neq j} \tilde{\mathbf{h}}_{(i,j)}^t)$$
$$p(\mathbf{x}_j^{t+1}|\mathbf{x}^t, \mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_j^{t+1}, \sigma^2 \mathbf{I})$$

$z_{ij,k}$ is the k-th element of z_{ij}, σ^2 is constant, the model learns the change during Δx_j^t.

For non-Markovian dynamics like human motions in [2], [1] used a GRU based RNN decoder to capture temporal patterns. It utilizes a slightly different version of GRU shown below:

$$r(t) = \sigma\left(f_{r,x}(x(t)) + f_{r,h}(h(t-1))\right)$$

$$i(t) = \sigma\left(f_{i,x}(x(t)) + f_{i,h}(h(t-1))\right)$$

$$n(t) = \tanh\left(f_{n,x}(x(t)) + r(t)^\intercal \cdot f_{n,h}(h(t-1))\right)$$

$$h(t) = \left(1 - i(t)\right)^\intercal \cdot n(t) + i(t)^\intercal \cdot h(t-1)$$

Where $x(t)$ is the input state at time t; $h(t)$ is the hidden state at t; $\sigma(\cdot)$ denotes the sigmoid function; and each f denote a fully connected layer. A zero vector is used as the initial hidden state $h(0)$ of the GRU.

3.4. Training

The ELBO involves only single step predictions. The problem with optimizing this objective is that the interactions only have a small effect on short-term dynamics. In action simulations a fixed velocity assumption can be a good approximation for a short time period. This leads to a suboptimal decoder that ignores the latent edges completely and achieves only a marginally worse reconstruction loss.

When we predict multiple steps into the future, the "degenerate" decoder would perform unsatisfactory. We have a separate MLP for each edge type. This makes the dependence on the edge type more explicit and harder to be ignored by the model.

We denote our decoder as $\mu_j(t+1) = f_{dec}\left(x_j^t\right)$ we will have:

$$\begin{aligned} \mu_j(2) &= f_{dec}\left(x_j^1\right) \\ \mu_j(t+1) &= f_{dec}\left(\mu_j^t\right) \quad t = 2, \ \ldots, M \\ \mu_j(M+2) &= f_{dec}\left(x_j^{M+1}\right) \\ \mu_j(t+1) &= f_{dec}\left(\mu_j^t\right) \quad t = M+2, \ \ldots, 2M \end{aligned}$$

With the errors accumulate for M steps, we would obtain a greater gradient step.

We perform backpropagation on the ELBO loss. We use the below equation to compute the reconstruction loss:

$$-\sum_{j \in V} \sum_{t=2}^{T} \frac{\left\| x_j(t) - \mu_j(t) \right\|_2^2}{2\sigma^2}$$

For the KL-divergence term:

$$\sum_{\substack{i \neq j, \\ i,j \in V}} \sum_{i=1}^{etype} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z)}$$

4. Extensions

4.1. Learnable Positional Encoding

Positional encoding has been used in transformers to distinguish tokens in the sequence [6]. The positional encoding is injected into the token embeddings by addition to add topological information to the embedding.

For static graphs like a traffic network or a human skeleton, we can use positional encoding to

distinguish features of one vertex to another. We augment the vertex features in the encoder and decoder with a set of learnable positional encoding using the below equation,

$$x'_v(t) = x_v(t) + p_v.$$

Where $x_v(t)$ is the embedding of vertex v at the t-th frame, and p_v is the positional encoding of vertex v learned by the neural network.

We initialize the positional encoding with random values in $\mathsf{Uniform}(-1,1)$. We used a separate encoding in the encoder and the decoder. In the encoder, we add the positional encoding to the vertex embeddings after the first MLP. In the decoder, we use the positional encoding as the hidden state at the first frame, $h(0)$.
See `models/nri_PE.py` for our implementation.

4.2. Graph Neural Diffusion

A dynamical system can be naturally described by a set of differential equations. Hence, we explore the application of Graph Neural Diffusion (GRAND) [2] in the NRI decoder.

4.2.1. Diffusion Equation on Graphs. GRAND [2] views the message propagation in a GNN as a diffusion process over the input graph. The diffusion equation below is used to model the diffusion of heat energy (can be particle or momentum as well) over a continuous space with initial condition $x(u,0)$.

$$\frac{\partial}{\partial t}x(u,t) = \mathsf{div}[g(u,t)\,\nabla x(u,t)]$$

Using the diffusion of heat as an analogy: $x(u,t)$ is the temperature at time t and position u; $g(u,t)$ is the diffusivity; $\nabla x(u,t)$ denotes the gradient of x; and $\mathsf{div}[\cdot]$ is the divergence operator.

In the continuous diffusion equation, $x(u,t)$ is influenced by the infinitesimally close points of u along the coordinate axes.

If we discretize u along the spatial coordinate axes with finite difference, each point in the mesh is influenced by its immediate neighbor. In another words, diffusion now occurs along the edges in the mesh. Take a $u \in \mathbb{N}$ discretization over $\mathbb{R}$ as an example. With forward difference,

$$\frac{d}{du}x(u,t) = \frac{x(u,t)-x(u-\delta,t)}{\delta_u}.$$

If we take $\delta_u = 1$, we obtain the discretized 1D diffusion equation:

$$\frac{d}{dt}x(u,t) = g(u,t)\big(x(u+1,t) - x(u,t)\big) + g(u-1,t)\big(x(u,t) - x(u-1,t)\big).$$

If we consider the 1D mesh as an undirected graph and the diffusivity as an edge feature, we can rearrange the equation into:

$$\frac{d}{dt}x(u,t) = \sum_{v \in N^-(u)} a(v \to u, t)\big(x(v,t) - x(u,t)\big)$$

Where $a(v \to u, t)$ is the diffusivity along edge $v \to u$ at time t.

Now take a step further and consider the diffusion of $x(u,t)$ on an arbitrary directed graph $G(V,E)$, u is now vertices in V. Converting the equation to matrix form, we obtain:

$$\frac{d}{dt}x(u,t) = \big(A(t) - D(t)\big)X(t)$$

Where $A_{|V|\times|V|}(t)$ is the diffusivity matrix, $A_{i,j}(t)$ is the diffusivity of edge $i \to j$; $D_{|V|\times|V|}(t)$ is a diagonal matrix containing the row-sum of A: $D_{i,i}(t) = \sum_{j=1}^{|V|} A_{i,j}(t)$; and $X_{|V|\times d}$ is the node feature matrix with feature dimension d. Note that the direction of the flow of $X(t)$can be in either direction regardless of the edge direction.

If we use self-attention [6] like [2]: $A_{i,j}(t) = \mathrm{softmax}\left(\frac{(W_K x(i,t)^\intercal) W_Q x(j,t)}{\sqrt{d}}\right)$ as the diffusivity, since all rows sum to 1, we obtain Equation (2) in [2]:

$$\frac{d}{dt}x(u,t) = (A(t) - I)X(t).$$

This ordinary differential equation (ODE) can be interpreted as the attention-based aggregation in [5] with continuous time stepping.

To make use of the equation, GRAND computes the attention matrix A and the initial state $x(u,0)$ with MLPs. Then solve the neural ODE with solvers that support backpropagation.

4.2.2. NRI with GRAND. When applying the graph diffusion framework to model a dynamical system, we assume the flow of x is directly proportional to its first-order gradient. When modelling a transient system, we add a drift term $\delta(u,t)$ to the equation to represent external influence:

$$\frac{d}{dt}x(u,t) = \alpha\big(A(t) - D(t)\big)X(t) + \beta\delta(u,t)$$

Where α and β are learnable scalar parameters added for numerical stability when evaluating the ODE at the early phase of the training.

Since in physical diffusion, diffusivity is non-negative real numbers. We use the scaled dot-product [6] to model the diffusivity:

$$A(t) = \frac{\mathsf{ReLU}\left(W_Q X(t)\right) \cdot \mathsf{ReLU}\left(W_K X(t)\right)^{\mathsf{T}}}{\sqrt{d}}$$

Where W_Q and W_K are learnable matrices. $\mathsf{ReLU}(\cdot)$ is applied such that $A_{i,j}(t) \geq 0$ since the diffusivity is always positive in physical diffusion.

We can view the aggregation function in the NRI decoder as a single discretized time step of $\frac{d}{dt}x(u,t)$ with the edge aggregation function described in Section 2.1. used as an approximator of the differential equation.

In our implementation, we replaced the edge aggregation layer with a neural ordinary differential equation. Since ground truth is occasionally injected to the autoregressive decoder, we keep the GRU to reconcile the output of our neural ODE with the new observation. Hence, we approximate the state of the system at the next frame with a piecewise continuous ODE with the hidden state at the previous frame used as initial value. We used a separate network to model the diffusivity for each relation type. We used the hidden state at the previous frame as both the initial value of the ODE and the drift term.

With this architecture, we expect the change in $X(t)$ would be small. We approximate $A(t)$ with (0) ; and $\delta(u,t)$ with $\delta(u,0)$. The approximation gives us a linear ODE that is: 1) easier to solve; 2) requiring less computation to evaluate; 3) requiring less memory to back-propagate. Finally, we use the solution of the ODE at $t = 1$ to predict the state of the next frame.

We used the dopri5 algorithm in the `torchdiffeq` package [7] for our neural ODE solver. See `models/grand.py` for our implementation using the formulation of diffusivity in [2]. See `models/grand_new.py` for the model with our formulation of diffusivity.

5. Experiments

5.1. Dataset

The CMU Motion Capture Database [3] is a collection of human motions captured with markers placed on a human subject. We focus on the walking category performed by subject 35. The dataset consists of 23 trials, with 12 trials used for training, 4 trials used for validation and 7 trials used for testing.

The `.asf` file contains the position of the markers (which we will refer to as joints), format of the data in `.amc` files and other metadata. We parse the `.asf` file to obtain the topology of the skeleton graph, which is a tree.

The `.amc` files record the motion in frames. The motion is defined as Euler angles. To convert Eular angles of child joints into cartesian coordinates, we first compute the transformation matrix with the below equation,

$$M(t) = R_x\big(\theta_x(t)\big)R_y\left(\theta_y(t)\right)R_z\big(\theta_z(t)\big).$$

Where $R_x\big(\theta(t)\big)$, $R_y\big(\theta(t)\big)$ and $R_z\big(\theta(t)\big)$ are rotation matrices created from Euler angle $\theta(t)$.

We then compute the local transformation matrix with the below equation,

$$L(t) = C^{-1}M(t)C.$$

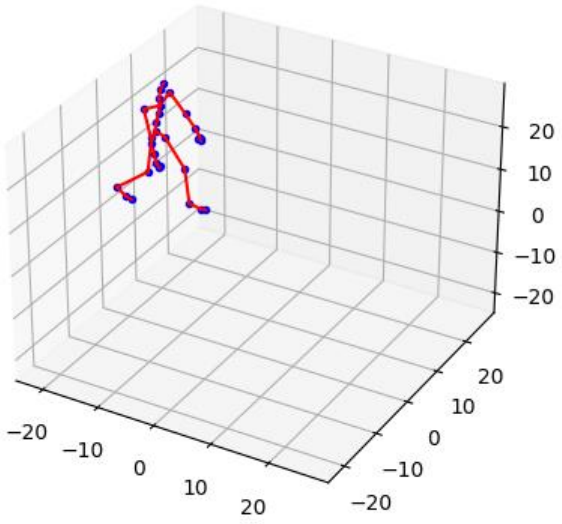
Where C is the rotation matrix created from the axis of rotation defined in the `.asf` file.

Finally, we can compute the cartesian coordinate of the joint $x(t)$ with the below equation,

$$x(t) = x_p(t) + l \cdot L(t) \cdot d$$

Where $x_p(t)$ is the coordinate of the parent joint, l is the length of the bone (i.e. the Euclidean distance between the parent joint and the child joint), and d is the direction vector defined in the `.asf` file.

We preprocess the raw data into cartesian coordinates stored in PyTorch tensors before training. For details, see `data/preprocess.py`. A visualization of the 20-th frame in trial #1 is shown in the below image.



At run time, we compute the vertex features as the concatenation of position: $x(t)$, and velocity: $x(t+1) - x(t)$, of the joints. For numerical stability, we normalize the features with the below equation,

$$x' = \frac{2(x - \min(x))}{\max(x) - \min(x)} - 1,$$

such that $|x'| \leq 1$.

For details, see `data/dataset.py`.

5.2. Implementation Details

In our experiments, we used NRI as our baseline (which we denote as **`nri`**). For NRI augmented with positional encoding, we denote the model as **`nri-pe`**. And we denote the model with GRAND decoder implemented with the diffusivity in Section 4.2.2 with softmax applied and positional encoding as **`grand`**.

For **`nri`** and **`nri-pe`**, we used a hidden state dimension of 128. We observed **`grand`** failed to generalize to the validation set after the first few epochs, so we restart the training process with a larger hidden state dimension of 256.

For other hyperparameters, we keep them the same as [1] in order to have a fair comparison. We used a learning rate of 0.0005 decaying every 200 epochs (same as [1]). For optimizer, we used Adam. A dropout probability of 0.5 is used to avoid overfitting.

We train all models on every possible continuous subsequence with 50 frames in the training set with a batch size of 8. We used 4 relation types with the distribution [0.91,0.03,0.03,0.03] as our sparse prior.

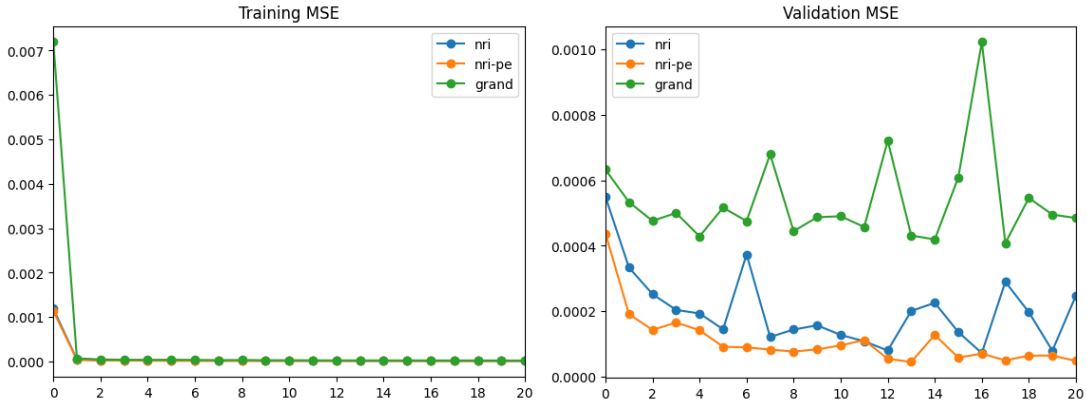
Due to the time constraint and limited resources, we only trained for 20 epochs and we did not perform cross validation.

5.3. Results

We report the mean squared error (MSE) of the best model (mean of all timesteps, all vertices and all state dimensions) on the validation set.

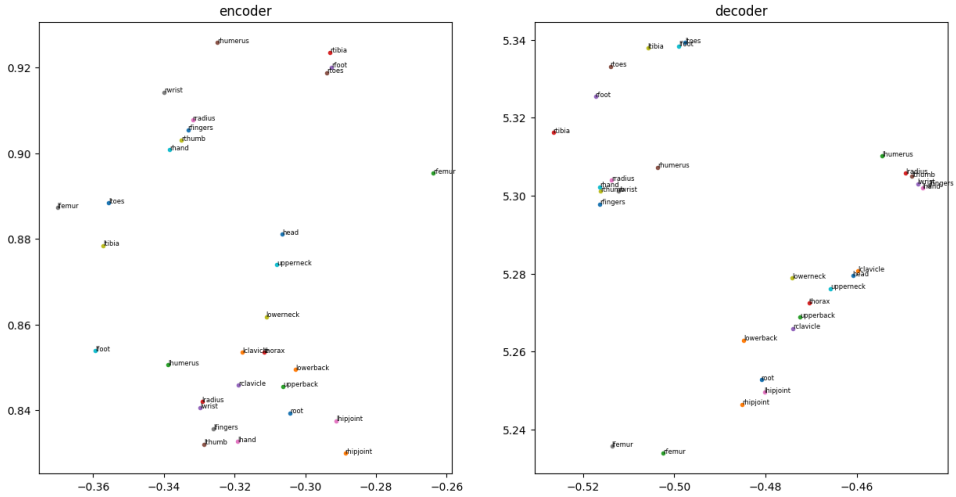
Model	Training MSE	Validation MSE
`nri`	9.99e-6	7.24e-5
`nri-pe`	**9.46e-6**	**4.55e-5**
`grand`	2.05e-5	4.08e-4

The plots of mean squared error is shown below:



See Appendix A for the plots of loss functions.

From the results, we noticed that augmenting the input with positional encoding noticeably improved the accuracy without increasing the amount of computation for training and inference. We examine the quality of the learned positional encoding in **`nri-pe`** with a t-SNE plot.



From the plot, vertices representing joints that are close to each other in the skeleton, especially joints in the hands) formed clusters. Hence, our positional encoding injected useful topological context to the input.

We observed that the performance of **`grand`** is subpar. We attempted the followings to improve the performance of the model:

1) Adding an additional MLP to compute the initial value of the ODE from $x(t)$ and $h(t-1)$ such that our ODE can start with a reconciled value;
2) Removing $\texttt{ReLU}(\cdot)$ from our formulation of diffusivity described in Section 4.2.2 such that diffusivity can be negative and $x(u,t)$ can flow in either direction;
3) Compute the drift term with edge aggregation used in NRI [1];
4) Regularizing the model with L2 norm and kinetic energy of the ODE [8];

No significant improvements are shown in the first few epochs, hence we halted the training process. This may be a hint that the dynamics of human motions does not depend on the second order derivative of the internal state and using the diffusion equation in the model would not be a good fit. This highlighted a limitation of neural ODE based models: the choice of the differential equation requires the modeler to have prior knowledge of the system.

In addition to the low performance, the **`grand`** model takes significantly longer time to train even if we use the same hidden state dimension (around 10 times longer than **`nri`** and **`nri-pe`**). This is due to the fact that the ODE needs to repeatedly compute $\frac{d}{dt}x(u,t)$ (which involves matrix multiplication) when computing the solution.

All code used in the project is available at https://github.com/D31T4/COMP9417-project.

6. Conclusion

In our work into NRI [1], we explored various extensions to the base model with the intension of improving the motion prediction task on subject #35 from the CMU Motion Capture Database. Our experiments revealed that for instance, incorporating learnable positional encoding to the encoder and decoder improved the accuracy of predictions. Replacing the GNN layers in the decoder with Graph Neural Diffusion provided a smoother trajectory but at the cost of computational efficiency. The key limitation of our approach is the need for vast computational resources for training the enhanced model and also the restriction to static graphs when using learnable positional encoding, which might not be generalizable to dynamic systems with evolving graphs. While our results are promising, there remain areas of uncertainty such as how the model would perform on other subjects or different types of motion given our focus on subject #35 from the walking category in the CMU motion database.

In conclusion, our study into enhancing the NRI model for motion prediction sheds light on the potential of integrating multiple neural network architectures. While our extensions brought improvements in some areas, they also presented new challenges. Nonetheless, it has set a foundation for further investigations, with the hope that future iterations can seamlessly integrate various neural architectures to effectively capture the countless precisions of interacting dynamical systems.

7. Further Work

7.1. Continuous Time Domain

Although our diffusion equation based ODE failed to achieve good performance in the motion prediction task, differential equation based models still open the possibility for continuous or adaptive time stepping. Some of the potential application of this property are:

1) Up-sampling: predicting states of the system between 2 frames;
2) Handling unevenly spaced time series as system state.

We think using a transformer based architecture equipped with a temporal encoding to compute vertex embedding would be suffice for (1) and (2).

7.2. Estimating Uncertainty

Also, [1] used a constant variance when computing the log likelihood loss. But from the results of the paper, the MSE increases as the number of timesteps predicted into the future increases. This indicated that the variance is certainly not a constant value. It would be nice if the model can output the variance to give us a sense of the certainty of the result.

7.3. Extending to Larger Graphs

The encoder and the decoder always evaluates a fully connected graph, this involves $O(|V|^2)$ message passing operations. This is not practical for a lot of real life graphs.

The time complexity can be reduced by:

1) performing the message passing operations with sparse matrices,
2) reducing the number of edges by graph re-wiring in [2] or k-nearest neighbor approximation of self-attention.

7.4. Utilizing Prior Knowledge of Edges

In the motion prediction task, we and [1] never utilized the prior topological information of the skeleton in the encoder. To make use of this information, we can create learnable edge embeddings.

7.5. Dynamic Graph Re-evaluation

The relation type of edges may change overtime during the time evolution of the system. The encoder is evaluated periodically in [1] and [9] such that the decoder can get updated information.

7.6. Capturing Long Term Dependencies

With RNNs in [1] and [9] struggled to predict accurate results as we look further into the future, [10] used an external memory pool to store key events in the past.

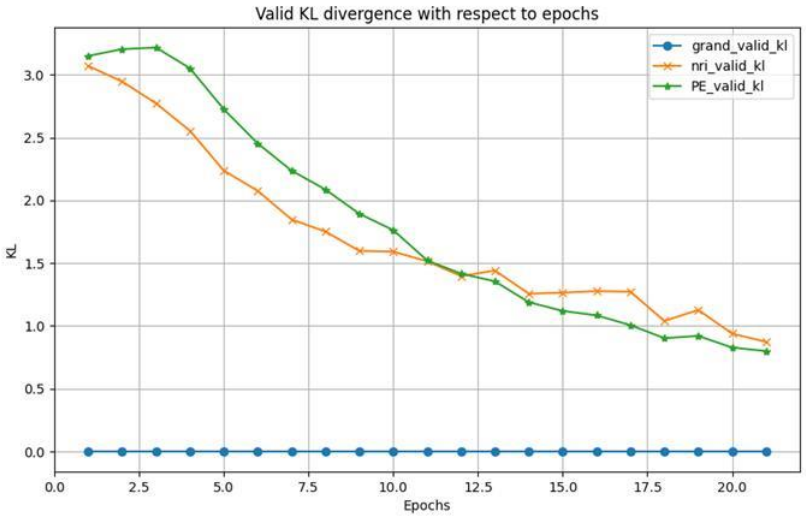
7.7. Others

Some other works include experimenting with other systems and other forms of differential equations.

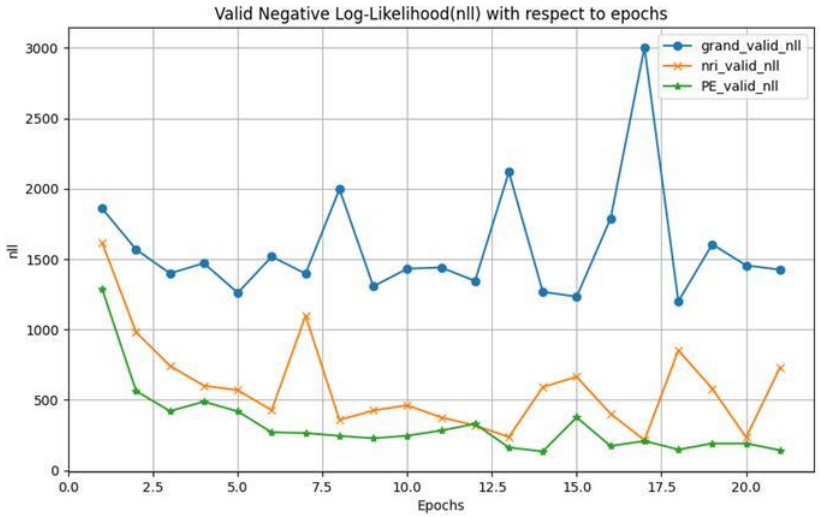
Reference

1. Kipf, T. Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. In *International Conference on Machine Learning (ICML)*, 2018.
2. Chamberlain, B., Rowbottom, J., Gorinova, M.I., Bronstein M., Webb, S., Rossi, E. GRAND: Graph Neural Diffusion. In *International Conference on Machine Learning (ICML)*, 2021.
3. CMU. Carnegie-Mellon Motion Capture Database. http://mocap.cs.cmu.edu, 2003.
4. Cho, K., Van Merrienboer, B., Gulcehre, C., Bahdanau, ¨ D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
5. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.
6. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, A., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NeurIPS*, pp. 5998–6008, 2017.
7. Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In *NeurIPS*, pp. 6571–6583, 2018.
8. Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. M. How to train your neural ode: The world of Jacobian and kinetic regularization. In *International Conference on Machine Learning (ICML)*, 2020.
9. Graber, C., Schwing, A. G. Dynamic Neural Relation Inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 8513-8522
10. Gong, D., Zhang, Z., Shi, J. O., Hengel, A. Memory-augmented Dynamic Neural Relational Inference. In *IEEE International Conference on Computer Vision (ICCV)*, 2021.

Appendix A



Plot of KL-divergence loss



Plot of negative log likelihood loss