Neural Relational Inference for Interacting Systems

Woo Suk Choi

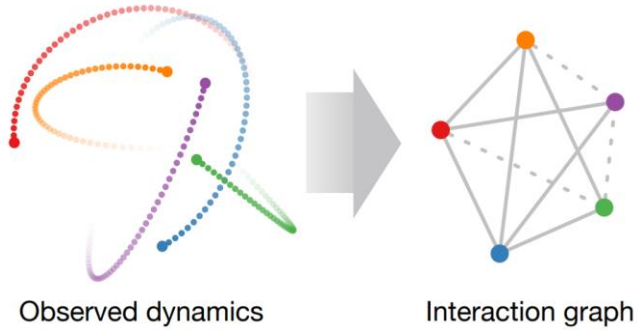
Seoul National University

Introduction

- Interacting system is an interplay of components in dynamics

- A wide range of dynamical systems in physics, biology, sports, and other areas can be seen as groups of interacting components
  - Example: A movement of basketball player can influence other basketball player

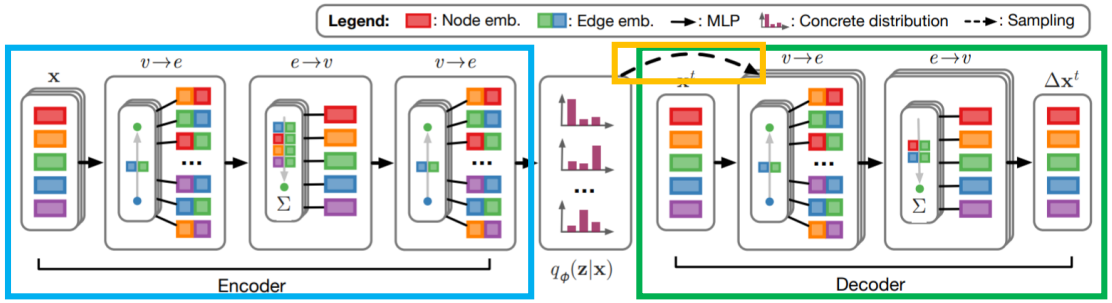
- Recent works for dynamical model of interacting systems
  - Implicit interaction model (GNNs)
    - Send messages over the fully-connected graph,
      - Where the interactions are modeled implicitly by message passing or attention mechanisms



Neural Relational Inference Model (NRI)

- Overview
  - Model: Variational auto-encoder with (discrete) edge types as discrete latent variables to solve the problem of inferring an explicit interaction structure
    - Encoder and decoder are GNN-based
  - Main intuition
    - Encoder: Generates latent interaction probability distribution (how the system interacts)
    - Decoder: Learns dynamical model (trajectory prediction) constrained by the interaction probability distribution of encoder
    - Trained jointly using Gumbel softmax trick as straight-through gradient estimator
  - Target
    - Simultaneously learn to predict the edge types and learn the dynamical model in unsupervised way
    - Formalize the model as a variational autoencoder(VAE) with evidence lower bound(ELBO):

$$\mathcal{L} = E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})]$$



Neural Relational Inference Model

- Single node-to-node message passing operation in GNN
  - Similar to “Neural Message Passing for Quantum Chemistry” (Justin Glimer et al.)
  - Node-to-Edge and Edge-to-Node

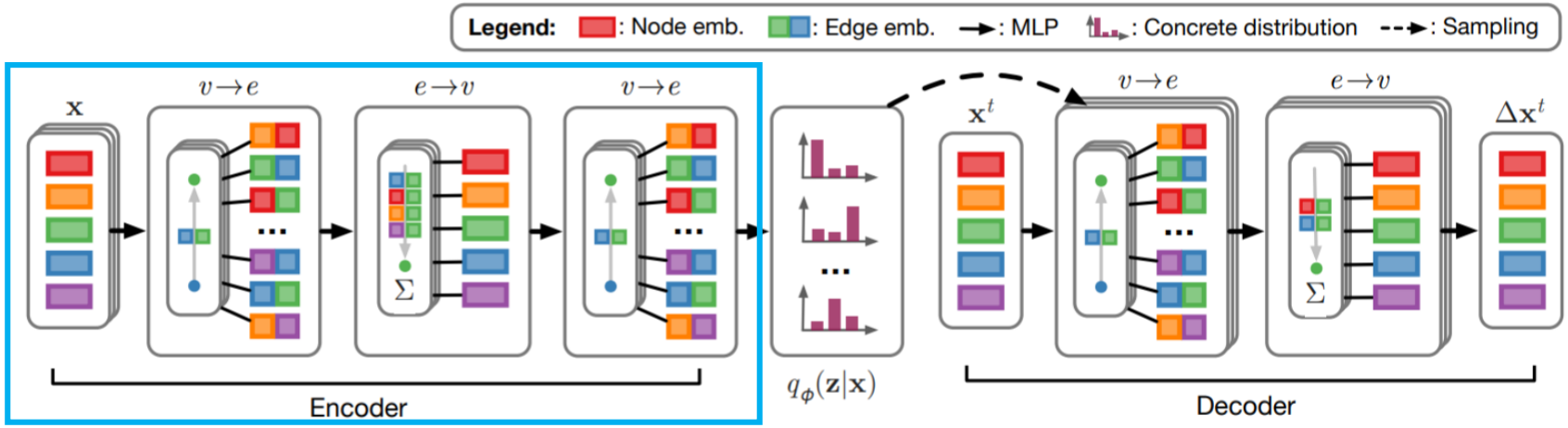
$$v \to e: \quad \mathbf{h}_{(i,j)}^{l} = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$
$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

- Difference between existing VAE and NRI model
  1. Decoder ignores latent code z
     - Train the decoder to predict multiple time steps (not a single step)
  2. Latent distribution is discrete
     - Use continuous relaxation in order to use the reparameterization trick

Neural Relational Inference Model

- Encoder
  - Since we do not know underlying graph,
    - Use a GNN on the fully-connected graph to predict latent graph structure
  - Model the edge type posterior as $q_\phi(z_{ij}|x) = softmax(h^2_{(i,j)})$

$$
\begin{aligned}
& \mathbf{h}_j^1 = f_{\text{emb}}(\mathbf{x}_j) \\
v \to e: \quad & \mathbf{h}_{(i,j)}^1 = f_e^1([\mathbf{h}_i^1, \mathbf{h}_j^1]) \\
e \to v: \quad & \mathbf{h}_j^2 = f_v^1\left(\textstyle\sum_{i \neq j} \mathbf{h}_{(i,j)}^1\right) \\
v \to e: \quad & \mathbf{h}_{(i,j)}^2 = f_e^2([\mathbf{h}_i^2, \mathbf{h}_j^2])
\end{aligned}
$$

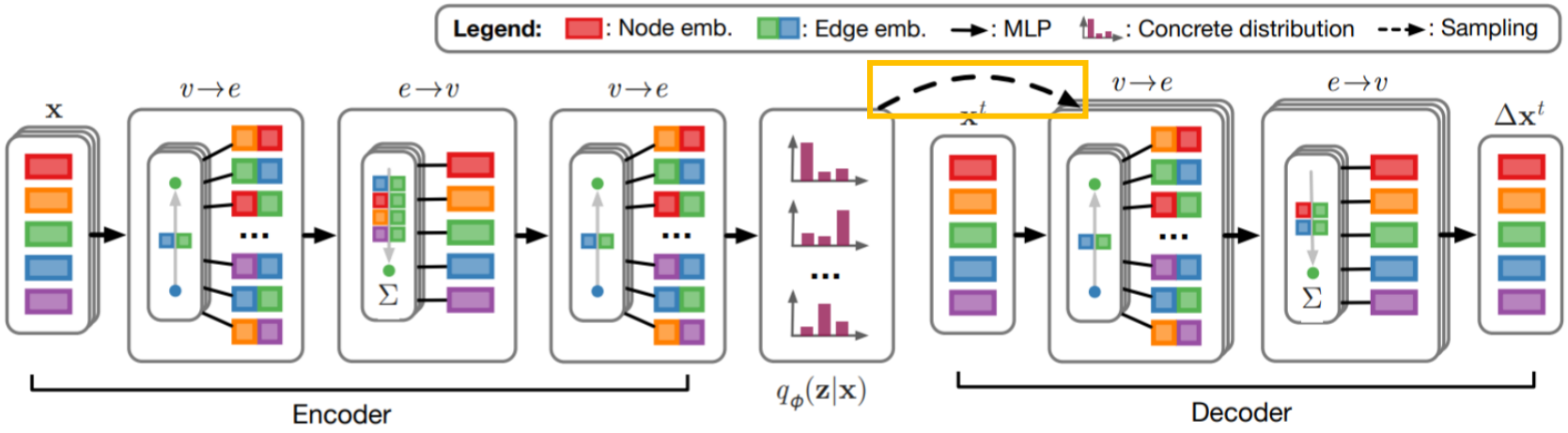


Neural Relational Inference Model

- Sampling
  - The operation of selecting z_{ij} is a discrete decision – therefore, we cannot directly propagate gradients through it.
  - Can use the Gumbel softmax trick to circumvent this:

$$\mathbf{z}_{ij} = \text{softmax}((\mathbf{h}^2_{(i,j)} + \mathbf{g})/\tau)$$

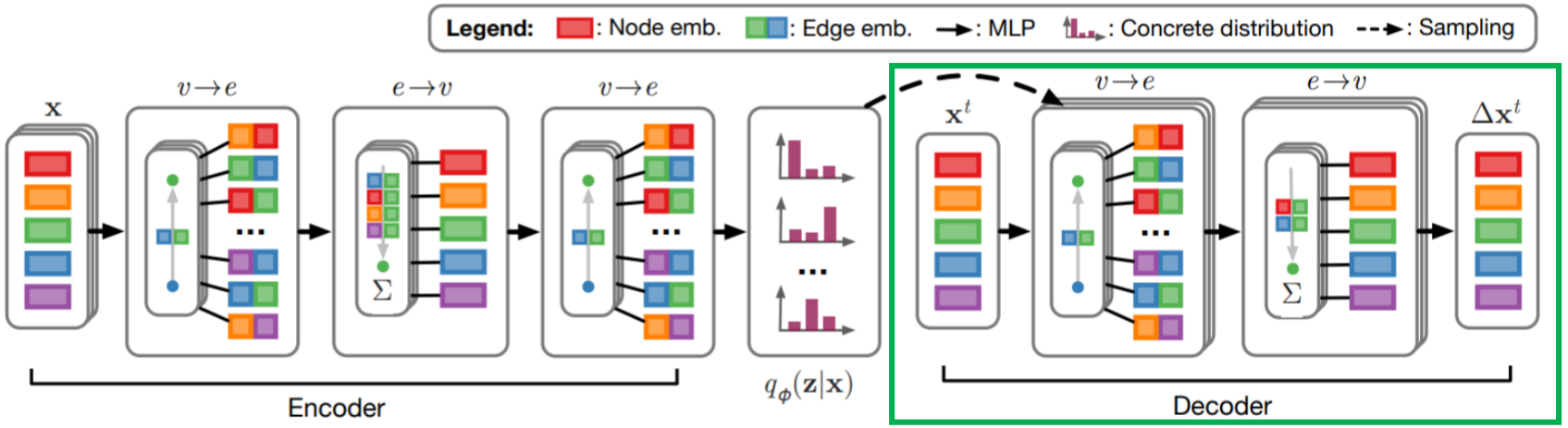
    - Where $g_k \sim Gumbel(0,1)$ and τ is a temperature parameter (converges to one-hot when $\tau \rightarrow 0$)
  - This is a continuous approximation to the discrete distribution



Neural Relational Inference Model

- Decoder
  - Predict the future continuation of dynamics for interacting system
    - With Markovian assumption, $p_\theta(x^{t+1}|x^t, \ldots, x^1, \boldsymbol{z})$
  - Can use any GNN algorithms as our decoder (since it is conditioned on the graph z)

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



Experiments

- Evaluation with various datasets
  - Physics simulation, Motion capture, Sport tracking data etc.
  - NRI can learn to discover ground-truth relations with very high accuracy

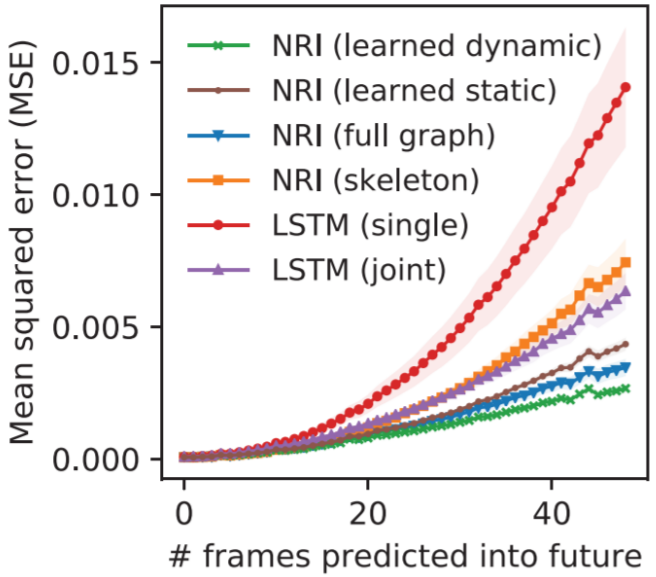
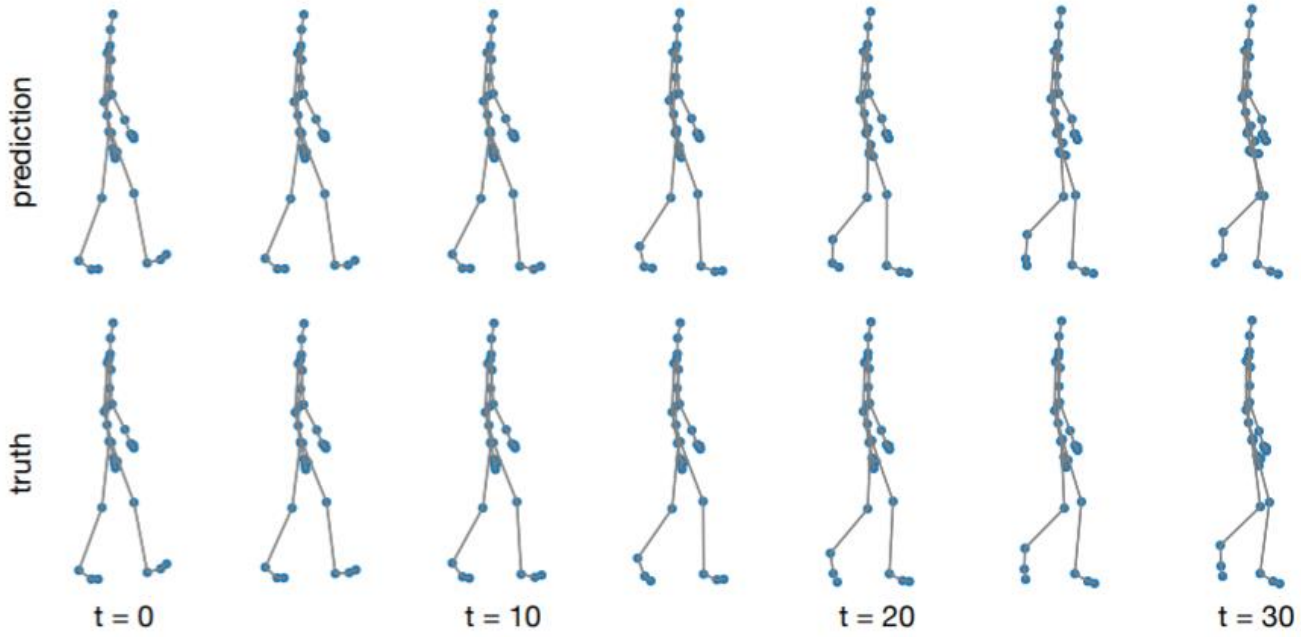


Figure 6. Test MSE comparison for motion capture (walking) data



Conclusion

- NRI is a versatile model for inferring latent interaction graphs from pointwise trajectories
- Latent graph discovery is still in its early phases
- Limitations
  - Sparsity – Most real-world graphs are sparse but proposed model need to start with complete graph and gradually discover sparsity


thank you
danke
рахмат
謝謝
ngiyabonga
teşekkür ederim
Баярлалаа
спасибо
mersi
kia ora
barka
welalin
spas
tack
faafetai lava
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
gracias
kiitos
dankie
nanni
nandri
dhanyavad
mauruuru
köszönöm
hvala
gràçie
bayarlalaa
bedankt
enkosi
dziękuję
akun
dankon
aciū
хвала
asante
manana
obrigada
tenki
murakoze
djiere dieuf
tau
mochchakkeram
дякую
mamnun
chokrane
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
sobodi
děkuji
sukriya
kop khun krap
taiku
arigatô
takk
dakujem
trugarez
obrigado
mèsi
didi madloba
sagolun
najis tuke
ありがとう
tanemirt
grazie
kam sah hamnida
rahmat
terima kasih
rahmet
diolch
dhanyavadagalu
shukriya
mercé
мерси
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci

실험수행 과정 및 결과 (링크: https://github.com/cws7777/gcn_finalproject)

- 코드가 공개되어있어 해당 코드 기반으로 실험을 수행
  - 코드 내부에 몇 가지 에러사항 수정
    - train.py
      - Line 93: os.mkdir → os.makedirs
      - Line 46: default = 'log' → default='./logs'
    - lstm_baseline.py
      - Line 94: args.suffix 부분 수정
  - 코드 실행 과정
    - data/generate_data.py 실행 (command: "python generate_data.py", 소요시간: 8시간)
    - train.py 실행(command: "python train.py", 소요시간 12시간)
    - lstm_baseline.py 실행 (command: "python lstm_baseline.py", 소요시간 17시간)
  - Github 링크의 참고사항
    - Github에는 data/generate_data.py에서 생성한 데이터를 올릴 수가 없었습니다. **(25MB 이상은 업로드 불가)**
    - 직접 실험을 돌리실 경우, 저에게 메일을 주시면 제가 데이터 파일들을 보내드리겠습니다. (wschoi@snu.ac.kr)
    - 직접적으로 실험을 안돌리셔도 logs 폴더에 가보시면, 제가 돌렸던 실험 log들이 있습니다.
      - 첫 번째 폴더(exp2020-06-18T10:51:31.601871)가 NRI에 대한 실험이며 그 안에 저장된 model과 log.txt가 있습니다.
        - log.txt는 epoch 마다 이전 epoch과 비교하여 좋은 성능 결과값이 나오는 것을 찍어놓은 log입니다.
      - 두 번째 폴더(exp2020-06-18T17:20:49.331843)는 lstm_baseline에 대한 실험이며 그 안에 저장된 log.txt가 있습니다.
        - log.txt는 epoch 마다 이전 epoch과 비교하여 좋은 성능 결과값이 나오는 것을 찍어놓은 log입니다.

실험수행 과정 및 결과 (링크: https://github.com/cws7777/gcn_finalproject)

- 코드가 공개되어있어 해당 코드 기반으로 실험을 수행
  - 실험을 돌리기위한 python 및 library requirement
    - Python 2.7 또는 3.6
    - Pytorch **0.2** → 0.2 이상 버전으로 돌릴 경우 simulation decoder가 break 나게 되며 실험이 되지 않음
  - '/data/generate_dataset.py'를 통해 데이터 생성 ('python generate_dataset.py')
    - 데이터 생성 소요시간: 8시간
    - Physical simulation(spring) dataset
    - location, velocity, edge에 대한 training(50000개), validation(10000개), test(10000개) .npy 생성
    - 코드 수행 캡처

```
(nri_test) choiws@choiws:~/Desktop/nri_test/data$ python generate_dataset.py
_springs5
Generating 50000 training simulations
Generating 10000 test simulations
Generating 10000 validation simulations
```

```
Iter: 9900, Simulation time: 0.4425368309020996
(nri_test) choiws@choiws:~/Desktop/nri_test/data$ ls
edges_test_springs5.npy   edges_valid_springs5.npy  __init__.py          loc_train_springs5.npy  __pycache__          synthetic_sim.pyc     vel_train_springs5.npy
edges_train_springs5.npy  generate_dataset.py       loc_test_springs5.npy  loc_valid_springs5.npy  synthetic_sim.py     vel_test_springs5.npy  vel_valid_springs5.npy
```

  - Syntetic_sim.py의 경우 generate_dataset.py에서 데이터 생성시 사용되는 python 파일

실험수행 과정 및 결과 (링크: https://github.com/cws7777/gcn_finalproject)

- 코드의 구조 설명
  - 파일 별 설명
    - lstm_baseline.py
      - 해당 논문에서 LSTM (joint)와 같은 것으로 우리가 알고 있는 기존 LSTM과 비슷한 모델이며 제안하는 모델과 비교하기 위함
        - 다른점: 입력 MLP를 통과한 모든 objects의 input representation을 concatenation
        - RecurrentBaseline class (Line 97 ~ 179)
    - modules.py
      - NRI 모델에서 사용하는 variational autoencoder에서 encoder와 decoder의 종류가 정의 되어있는 곳
        - MLPEncoder (Line 89 ~ 147) : 2-layer MLP(hidden and output dim. of 256, batch norm., drop out, and ELU activation)
          - node2edge (Line 113 ~ 116) : reciever와 sender feature의 edge concatenation을 반환
          - edge2node (Line 118 ~ 123) : 모든 incoming edge features를 sum으로 accumulate
        - MLPDecoder (Line 415 ~ 521) : All/Single time-step 예측
          - single_step_forward (Line 436 ~ 482) : single time-step 예측에 대한 forward function
          - forward (Line 484 ~ 521) : all time-step 예측에 대한 forward function
        - CNNEncoder (Line 150 ~ 226) : trajectory size 변화 encoding을 허용
          - CNN (Line 46 ~ 86) : 1D convolutions with attention
        - RNNDecoder (Line 524 ~ 651) : single step 예측에 GRU 업데이트 방식을 추가
          - single_step_forward (Line 553 ~ 600) : GRU 방식 (Line 586 ~ 590)
        - SimulationDecoder (Line 229 ~ 412) : 각 데이터셋에 따른 Ground-truth 시뮬레이터
          - Motion의 Newtonian equation integrator (Line 242 ~ 267)

실험수행 과정 및 결과 (링크: https://github.com/cws7777/gcn_finalproject)

- 코드의 구조 설명
  - 파일 별 설명
    - utils.py
      - 대표적으로 제안한 모델에서 Sampling할 때 사용되는 Gumbel Softmax, 데이터를 로드하는 load_data 등의 utils가 있음
        - Gumbel Softmax (Line 69 ~ 108) : pytorch community 및 ericjang이라는 유저의 github에서 가져온 코드
    - train.py
      - 여러가지 argument들을 기반으로 training과 test를 수행하는 main 파일
      - Encoder와 Decoder 종류를 사용자가 argument로 설정하고 module.py에서 불러와 수행 (Line 117 ~ 141)
      - Train (Line 184 ~ 292)
        - 설정한 모듈들을 불러와 train을 진행하고 Epoch 별로 train과 validation의 MSE, NLL 등의 수치를 print (Line 267 ~ 276)
        - 성능이 좋은 모델들은 저장 (Line 277 ~ 290)
      - Test (Line 295 ~ 383)
        - train을 기반으로 저장된 best 모델을 불러옴 (Line 305 ~ 306)
        - test를 진행하고 Epoch 별로 nll_test, mse_test, 등을 print (Line 365 ~ 372)
    - train_enc.py와 train_dec.py
      - Encoder와 decoder를 따로 train 할 수 있도록 코드를 나눠 놓음
      - train.py에 있는 코드 부분을 따로 나눠놓은 코드이며 train.py에서 동작하는 방식과 같음

실험수행 과정 및 결과 (링크: https://github.com/cws7777/gcn_finalproject)

- 데이터 생성 후, 실험 진행 ('python train.py')
  - 논문에 기재되어있는 실험 결과와 reproducing 결과 비교
    - 공개한 코드는 해당 논문에서 Table1과 Table 2에서의 Springs부분
      - Table 1. Accuracy of unsupervised interaction recovery에서의 NRI (learned)부분
      - Table 2. MSE in predicting future states for simulations with 5 interacting objects에서의 NRI (learned) 부분
    - 실험 세팅(prediction_steps = 10, num_atoms(objects) = 5)

```
(nri_test) choiws@choiws:~/Desktop/nri_test$ python train.py
Namespace(batch_size=128, cuda=True, decoder='mlp', decoder_dropout=0.0, decoder_hidden=256, dims=4, dynamic_graph=False, edge_types=2, encoder='mlp', encoder_dropout=0.0, encod
er_hidden=256, epochs=500, factor=True, gamma=0.5, hard=False, load_folder='', lr=0.0005, lr_decay=200, no_cuda=False, no_factor=False, num_atoms=5, prediction_steps=10, prior=F
alse, save_folder='./logs', seed=42, skip_first=False, suffix='_springs5', temp=0.5, timesteps=49, var=5e-05)
Using factor graph MLP encoder.
Using learned interaction net decoder.
```

    - 실험 reproduce 결과

Table 1. Accuracy (in %) of unsupervised interaction recovery.

Model	Springs	Charged	Kuramoto
	5 objects		
Corr. (path)	52.4±0.0	55.8±0.0	62.8±0.0
Corr. (LSTM)	52.7±0.9	54.2±2.0	54.4±0.5
NRI (sim.)	**99.8**±0.0	59.6±0.8	–
NRI (learned)	**99.9**±0.0	**82.1**±0.6	**96.0**±0.1
Supervised	99.9±0.0	95.0±0.3	99.7±0.0

	Springs		
Prediction steps	1	10	20
Static	7.93e-5	7.59e-3	2.82e-2
LSTM (single)	2.27e-6	4.69e-4	4.90e-3
LSTM (joint)	4.13e-8	2.19e-5	7.02e-4
NRI (full graph)	1.66e-5	1.64e-3	6.31e-3
NRI (learned)	**3.12e-8**	**3.29e-6**	**2.13e-5**
NRI (true graph)	1.69e-11	1.32e-9	7.06e-6

Table 2. Mean squared error (MSE) in predicting future states for simulations with 5 interacting objects.

NRI(learned) model

```
Optimization Finished!
Best Epoch: 0492
--------------------------------
--------Testing-----------------
--------------------------------
nll_test: 76.4134466094 kl_test: -0.0015021029 mse_test: 0.0000397987 acc_test: 0.9986847310
MSE: [ 0.000000032942 , 0.000000131017 , 0.000000292691 , 0.000000515906 , 0.000000798559 , 0.000001138511 , 0.000001533449 , 0.000001981092 , 0.000002478713 , 0.000003023529 ,
0.000003612710 , 0.000004243413 , 0.000004913697 , 0.000005621429 , 0.000006365979 , 0.000007146337 , 0.000007961426 , 0.000011093128 , 0.000014427284 , 0.000017692602 ]
./logs/exp2020-06-18T10:51:31.601871/
```

LSTM(baseline) model

```
Optimization Finished!
Best Epoch: 0472
--------------------------------
--------Testing-----------------
--------------------------------
nll_test: 199.8037110341 mse_test: 0.000104064436 mse_baseline_test: 0.0001205871
MSE: [ 0.000000022443 , 0.000000111469 , 0.000000318527 , 0.000000732543 , 0.000001500170 , 0.000002852601 , 0.000005140345 , 0.000008876621 , 0.000014790002 , 0.000023887296 ,
0.000037526344 , 0.000057476394 , 0.000086006447 , 0.000126022132 , 0.000181158044 , 0.000255900319 , 0.000355729368 , 0.000489608559 , 0.000663363608 , 0.000886167516 ]
MSE Baseline: [ 0.000078298188 , 0.000312591990 , 0.000701709709 , 0.001244124258 , 0.001937958645 , 0.002780992072 , 0.003770666430 , 0.004904094152 , 0.006178058218 , 0.007589
018904 , 0.009133122861 , 0.010806214996 , 0.012603825890 , 0.014521185309 , 0.016553251073 , 0.018694678321 , 0.020939871669 , 0.023285211995 , 0.025722427294 , 0.028244880959
]
logs/exp2020-06-18T17:20:49.331843/
```