Updates and Progress

Yogesh Verma
Doctoral Candidate
Aalto University

Updates

- Paper read: 20/1280
  - EQUIBIND: Geometric Deep Learning for Drug Binding Structure Prediction
  - Hamiltonian Graph Networks with ODE Integrator
  - Graph Normalizing Flows
  - PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows [Reading]
  - A Lagrangian Approach to Information Propagation in Graph Neural Networks [Reading]

Updates

- Paper read: 20/1280
  - EQUIBIND: Geometric Deep Learning for Drug Binding Structure Prediction
  - Hamiltonian Graph Networks with ODE Integrator
  - Graph Normalizing Flows
  - PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows [Reading]
  - A Lagrangian Approach to Information Propagation in Graph Neural Networks [Reading]
- CNF for generation & Coding

Background

Numerical integrators for solving ODEs

- Given a first-order ODE and initial conditions, numerical integration can be used to approximate solutions to the initial value problem

$$\mathbf{y} \equiv \mathbf{y}(t) \quad , \quad \dot{\mathbf{y}} \equiv \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = f_{\dot{\mathbf{y}}}(t, \mathbf{y}) \quad , \quad \mathbf{y}\,(t_0) = \mathbf{y}_0 \tag{1}$$

- The family of Runge-Kutta (RK) integrators are fully differentiable and can generate trajectories via iterations of the form, $y_{n+1} = RK(t_n, \Delta t, y_n, f_{\dot{y}})$

Hamiltonian Mechanics

- In Hamiltonian mechanics, the Hamiltonian, $\mathcal{H}(q,p)$, is a function of the canonical position, q, and momentum, p, coordinates

$$\dot{\mathbf{q}} \equiv \frac{d\mathbf{q}(t)}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}} \quad , \quad \dot{\mathbf{p}} \equiv \frac{d\mathbf{p}(t)}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}} \quad , \quad (\dot{\mathbf{p}}, \dot{\mathbf{q}}) \equiv \frac{d(\mathbf{p},\mathbf{q})}{dt} = f_{\dot{\mathbf{p}},\dot{\mathbf{q}}}(\mathbf{q},\mathbf{p}) \tag{2}$$

Models

Delta graph network (DeltaGN)

- Baseline, DeltaGN directly predicts changes to $(\mathbf{q}, \mathbf{p})$

$$(\mathbf{q}, \mathbf{p})_{n+1} = (\mathbf{q}, \mathbf{p})_n + (\Delta\mathbf{q}, \Delta\mathbf{p})_n \quad , (\Delta\mathbf{q}, \Delta\mathbf{p})_n \leftarrow GN(\Delta t, \mathbf{q}_n, \mathbf{p}_n, c, \phi)$$

Models

Delta graph network (DeltaGN)

- Baseline, DeltaGN directly predicts changes to $(\mathbf{q}, \mathbf{p})$

$$(\mathbf{q}, \mathbf{p})_{n+1} = (\mathbf{q}, \mathbf{p})_n + (\Delta\mathbf{q}, \Delta\mathbf{p})_n \quad , (\Delta\mathbf{q}, \Delta\mathbf{p})_n \leftarrow GN(\Delta t, \mathbf{q}_n, \mathbf{p}_n, c, \phi) \tag{3}$$

- The c are static parameters (masses, spring constants) of the system, and ϕ are the neural network parameters.
- The GN's signature matches the integrator's, and so is analogous to learning an integrator.

ODE graph network (OGN)

- "ODE graph network" (OGN) imposes an ODE integrator as an inductive bias in the GN, by assuming that the dynamics of $(\mathbf{q}, \mathbf{p})$ follow a first-order ODE Eq(1)

ODE graph network (OGN)

- "ODE graph network" (OGN) imposes an ODE integrator as an inductive bias in the GN, by assuming that the dynamics of $(\mathbf{q}, \mathbf{p})$ follow a first-order ODE Eq(1)
- Train a neural network that learns to produce the time derivatives $(\dot{\mathbf{p}}, \dot{\mathbf{q}})$

Models

ODE graph network (OGN)

- "ODE graph network" (OGN) imposes an ODE integrator as an inductive bias in the GN, by assuming that the dynamics of $(\mathbf{q}, \mathbf{p})$ follow a first-order ODE Eq(1)
- Train a neural network that learns to produce the time derivatives $(\dot{\mathbf{p}}, \dot{\mathbf{q}})$
- Node output of a GN to model the per-particle time derivatives, and provide the GN, together with the initial conditions and , to an RK integrator

$$(\mathbf{q}, \mathbf{p})_{n+1} = RK(\Delta t, (\mathbf{q}, \mathbf{p})_n, f^{\mathrm{OGN}}_{\dot{\mathbf{q}}, \dot{\mathbf{p}}}) \quad , f^{\mathrm{OGN}}_{\dot{\mathbf{q}}, \dot{\mathbf{p}}} \equiv GN(\mathbf{q}, \mathbf{p}, c, \phi) = (\dot{\mathbf{p}}, \dot{\mathbf{q}}) \tag{4}$$

Models

ODE graph network (OGN)

- "ODE graph network" (OGN) imposes an ODE integrator as an inductive bias in the GN, by assuming that the dynamics of $(\mathbf{q}, \mathbf{p})$ follow a first-order ODE Eq(1)
- Train a neural network that learns to produce the time derivatives $(\dot{\mathbf{p}}, \dot{\mathbf{q}})$
- Node output of a GN to model the per-particle time derivatives, and provide the GN, together with the initial conditions and , to an RK integrator

$$(\mathbf{q}, \mathbf{p})_{n+1} = RK(\Delta t, (\mathbf{q}, \mathbf{p})_n, f^{\mathrm{OGN}}_{\dot{\mathbf{q}}, \dot{\mathbf{p}}}) \quad , f^{\mathrm{OGN}}_{\dot{\mathbf{q}}, \dot{\mathbf{p}}} \equiv GN(\mathbf{q}, \mathbf{p}, c, \phi) = (\dot{\mathbf{p}}, \dot{\mathbf{q}}) \tag{4}$$

- The $f_{\dot{\mathbf{q}}, \dot{\mathbf{p}}}$ is a function that the integrator can use to operate on any (q, p) and query more than once

Hamiltonian ODE graph network (HOGN

- Defined "Hamiltonian ODE graph network" (HOGN) by computing a single scalar for the system which is Hamiltonian via the global output

Models

Hamiltonian ODE graph network (HOGN

- Defined "Hamiltonian ODE graph network" (HOGN) by computing a single scalar for the system which is Hamiltonian via the global output
- Differentiating it with respect to its inputs, q and p, and, in accordance with Hamilton's equations to predict the evolution of the physical system

Models

Hamiltonian ODE graph network (HOGN

- Defined "Hamiltonian ODE graph network" (HOGN) by computing a single scalar for the system which is Hamiltonian via the global output
- Differentiating it with respect to its inputs, q and p, and, in accordance with Hamilton's equations to predict the evolution of the physical system
- Node output of a GN to model the per-particle time derivatives, and provide the GN, together with the initial conditions and , to an RK integrator

$$(\mathbf{q},\mathbf{p})_{n+1} = RK(\Delta t, (\mathbf{q},\mathbf{p})_n, f^{\mathrm{OGN}}_{\dot{\mathbf{q}},\dot{\mathbf{p}}}) \quad , f^{\mathrm{OGN}}_{\dot{\mathbf{q}},\dot{\mathbf{p}}} \equiv GN(\mathbf{q},\mathbf{p},c,\phi) = (\dot{\mathbf{p}},\dot{\mathbf{q}}) \tag{5}$$

Hamiltonian ODE graph network (HOGN

- Defined "Hamiltonian ODE graph network" (HOGN) by computing a single scalar for the system which is Hamiltonian via the global output
- Differentiating it with respect to its inputs, q and p, and, in accordance with Hamilton's equations to predict the evolution of the physical system
- Node output of a GN to model the per-particle time derivatives, and provide the GN, together with the initial conditions and , to an RK integrator

$$(\mathbf{q},\mathbf{p})_{n+1} = RK(\Delta t, (\mathbf{q},\mathbf{p})_n, f^{\mathrm{OGN}}_{\dot{\mathbf{q}},\dot{\mathbf{p}}}) \quad , f^{\mathrm{OGN}}_{\dot{\mathbf{q}},\dot{\mathbf{p}}} \equiv GN(\mathbf{q},\mathbf{p},c,\phi) = (\dot{\mathbf{p}},\dot{\mathbf{q}}) \tag{5}$$

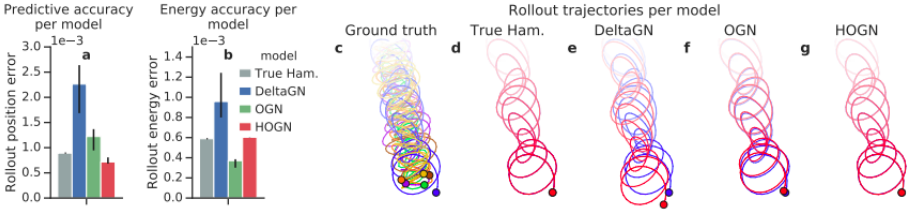
- The $f_{\dot{\mathbf{q}},\dot{\mathbf{p}}}$ is a function that the integrator can use to operate on any (q, p) and query more than once

Results

- Train and Test approach on datasets consisting of particle systems where particle j exerts a spring force on particle i, as defined by Hooke's law, $F^{ij} = -k^{ij}(\mathbf{q}^i - \mathbf{q}^j)$ limiting particles between 4 and 9 particles

Results

- Train and Test approach on datasets consisting of particle systems where particle j exerts a spring force on particle i, as defined by Hooke's law, $F^{ij} = -k^{ij}(\mathbf{q}^i - \mathbf{q}^j)$ limiting particles between 4 and 9 particles
- Rollout position error: RMS position error averaged across all examples, dimensions, particles, and sequence axis.
- Rollout energy error: Energy error is calculated as the RMS of the deviation between the mean energy of a trajectory and the initial energy (normalized by the initial energy, to yield relative errors), averaged across all examples



Connection to our work

- One can describe canonical (latent) kinetic and potential energy terms in a molecule or in any general case

Connection to our work

- One can describe canonical (latent) kinetic and potential energy terms in a molecule or in any general case

$$V = \sum_{i}^{N_v} \sum_{j}^{N_{hops}} V(r_{ij}) = f_{PE}(\mathcal{G}(\mathcal{V}, \mathcal{E}), t) \tag{6}$$

$$U = \sum_{i}^{N_v} \frac{1}{2} m \dot{r_i}^2 = f_{KE}(\mathcal{G}(\mathcal{V}, \mathcal{E})_t, \mathcal{G}(\mathcal{V}, \mathcal{E})_{t-1}, t) \tag{7}$$

$$\mathcal{L} = U - V, \quad \frac{d(U+V)}{dt} = 0 \tag{8}$$

Connection to our work

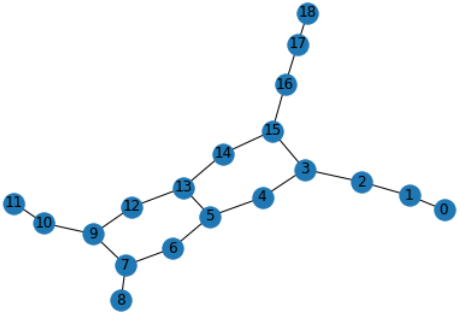
- One can use this as an updating strategy based on Lagrangian, new message can be based on

Connection to our work

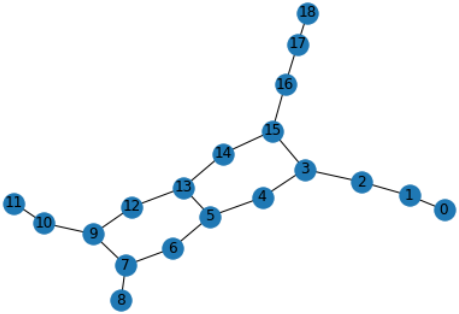
- One can use this as an updating strategy based on Lagrangian, new message can be based on

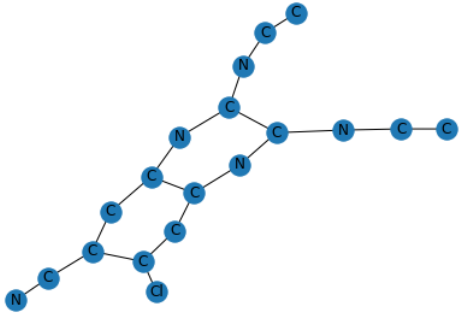
$$v_i(t) = v_i(t-1) + g\Big(\underbrace{\frac{\partial \mathcal{L}}{\partial r_i}}_{\text{Over each node}} , \underbrace{\frac{\partial \mathcal{L}}{\partial r_{ij}}}_{\text{Over each connection}} \Big) \tag{9}$$

Aim

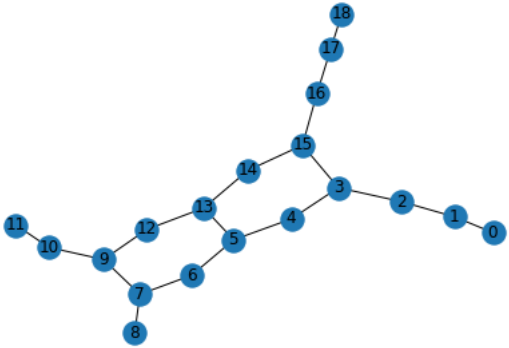


Aim



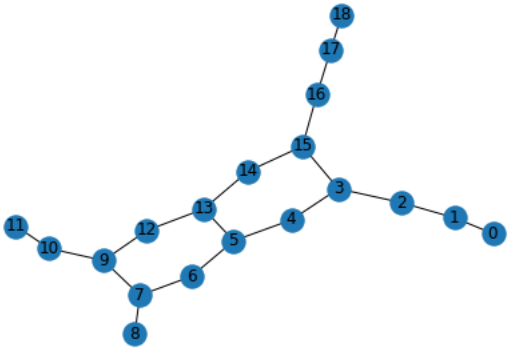


How to do?



- Given an unlabelled connection schema, initialize a joint density at each node label over atom vocabulary

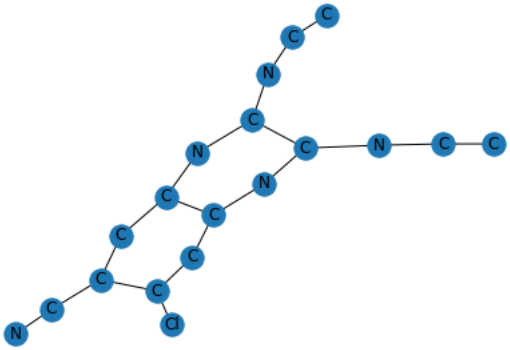
How to do?



- Given an unlabelled connection schema, initialize a joint density at each node label over atom vocabulary
- Use CNF to flow the density, f can be a NN or an MPNN.

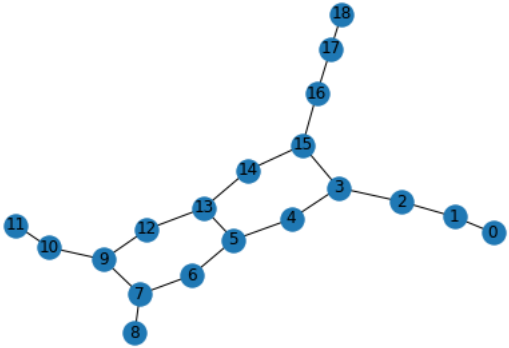
$$log(p(z_1)) = log(p(z_0)) - log|det\frac{\partial f}{\partial z_0}|$$

How to do?



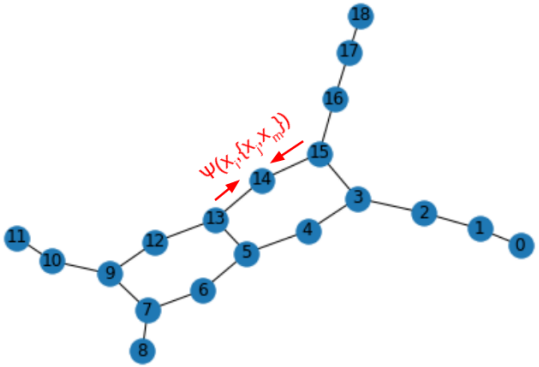
- Given an unlabelled connection schema, initialize a joint density at each node label over atom vocabulary
- Use CNF to flow the density, f can be a NN or an MPNN.
- At the end get node label (type of atom) having the highest density

Neighbourhood Control?



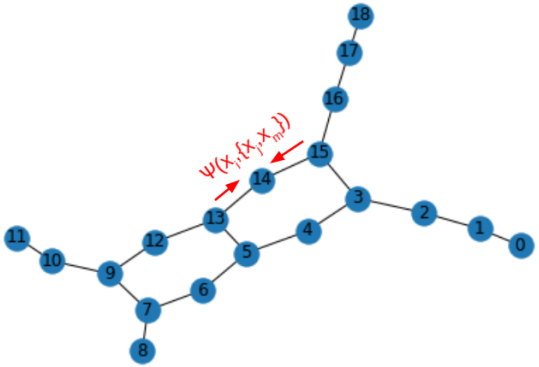
- How to include neighbourhood as a control when flowing the density?

Neighbourhood Control?



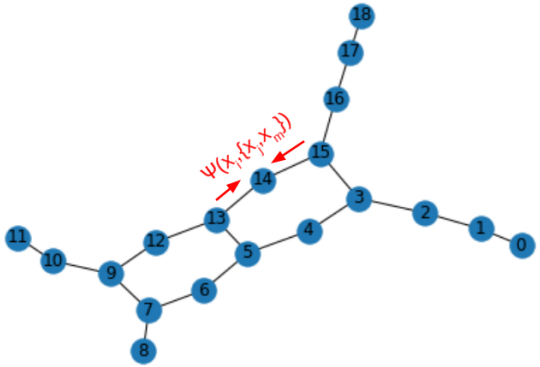
- How to include neighbourhood as a control when flowing the density?
- Using a function $\psi(x_i, \{x_j, x_k, ...\})$

Neighbourhood Control?



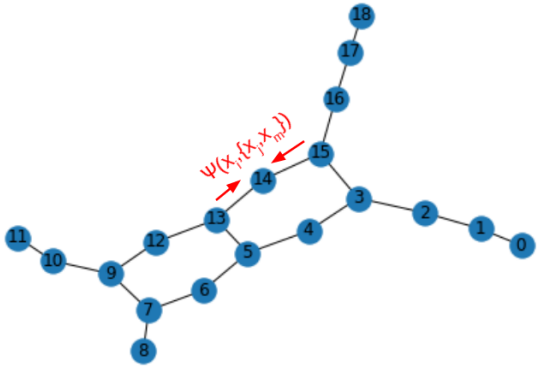
- How to include neighbourhood as a control when flowing the density?
- Using a function $\psi(x_i, \{x_j, x_k, ...\})$
- Choices:
  - NN

Neighbourhood Control?



- How to include neighbourhood as a control when flowing the density?
- Using a function $\psi(x_i, \{x_j, x_k, ...\})$
- Choices:
  - NN
  - Kernel like RBF
    $(\psi(x_i, \{x_j, x_k, ...\}) = \sum_l^{N_{x_i}} \phi(x_i, x_l))$

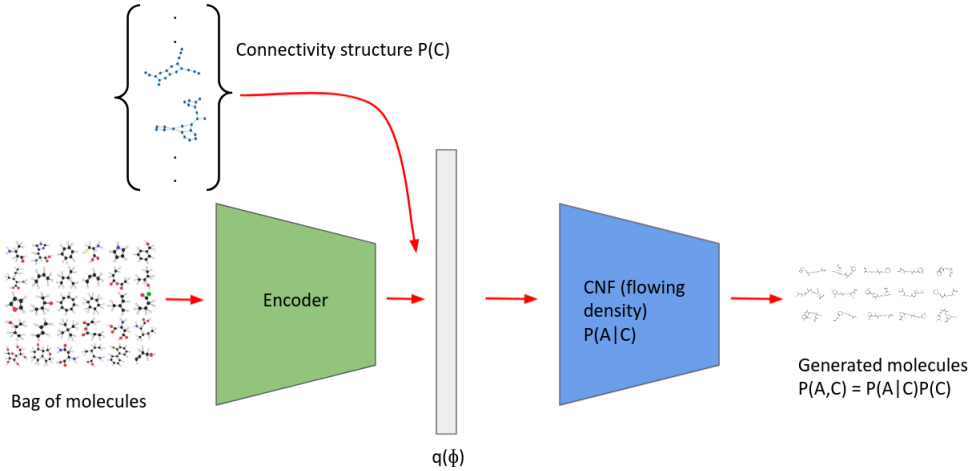
Neighbourhood Control?



- How to include neighbourhood as a control when flowing the density?
- Using a function $\psi(x_i, \{x_j, x_k, ...\})$
- Choices:
  - NN
  - Kernel like RBF
    $(\psi(x_i, \{x_j, x_k, ...\}) = \sum_l^{N_{x_i}} \phi(x_i, x_l))$
  - .
  - .
- Recursively updated during each iteration and used as a feature input

Extension

- We can extend it as an encoder-decoder formalism as shown above



Master Equation

The drug design process can be represented as :

$$P(Prop, f, 3D, A, C) = P(Prop|f, 3D) \cdot P(f|3D, A) \cdot \underbrace{P(3D|A, C)}_{\text{Z-matrix}} \cdot \underbrace{P(A|C)P(C)}_{\text{We are here}} \tag{10}$$

THANK YOU