Generative Graph Neural Network Markov State Models

Giacomo Bertin[1]
[1]

Student ID:2029305
15th June, 2021

Abstract

We propose a Generative Graph Neural Network model able to learn to propagate proteins molecular dynamic. Our net is composed by a deterministic encoder, which will maps from high-dimensional configuration space to a small-sized vector, a probabilistic decoder which will generate the next set of coordinates. We implemented a regularization loss in order to generate realistic configurations and a flexible message-parsing layer for the graph convolution

1 Introduction

1.1 Markov State Models

Complex dynamical systems exhibit events on vastly different timescales, for example molecular dynamic of proteins involves small vibration on the timescales of 10^{-15} seconds, while events of biological significance occurs into a timescale of 10^{-3} seconds. If the time resolution is enough coarse a Markovian Model can describe the rapid protein transition between long-lived metastable states. An MSM approximates the molecular kinetics by a matrix of conditional transition probabilities among discrete states[1]:

$$\mathbb{P}(x_{t+\tau} = y | x_t = x) = \chi(x)^T \mathbf{q}(y;\tau) = \sum_{i=1}^{m} \chi_i(x) q_i(y;\tau)$$

where $\chi_i(x) = \mathbb{P}(x_t \in \textit{state } i | x_t = x)$, so $\sum_{i=1}^{m} \chi_i(x) = 1 \forall x$; $q_i(y;\tau) = \mathbb{P}(x_{t+\tau} = y | x_t \in \textit{state } i)$. Learning directly $\mathbb{P}(x_{t+\tau}|x_t)$ is a unnecessarily difficult task, we could instead use its

spectral decomposition[23]:

$$\mathbb{P}(\mathbf{x}_{t+\tau}) = \int \mathbb{P}(\mathbf{x}_{t+\tau}|\mathbf{x}_t)\mathbb{P}(\mathbf{x}_t)d^{3N}\mathbf{x}_t$$

where N is the number of atoms.

1.2 Graph Neural Networks

Graph Convolution Neural Networks are a class of deep learning models that perform inference on graph data. Graph-based methods are becoming popular for their node permutation invariance and parameters sharing, which allows to naturally include physical symmetries without data augmentation techniques. A graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{v_1, v_2, ..., v_N\}$ are the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ are the edges. Here we will deal with directed graph, so $e_{ij} = (v_i, v_j) \subset \mathcal{E}$ is an edge connecting the vertices from v_i to v_j. Nodes and edges have their features, indicated respectively as $\mathbf{h}_v \subset \mathbb{R}^D$ and $\mathbf{f}_e \subset \mathbb{R}^C$.
Convolution operators on graph can be divided in spectral and spatial approaches: the first group act on spectral representation of the graphs and find a solid theoretical background in signal processing, while spatial approaches define the convolution operator directly on the graph. We followed the spatial convolution approach.
The convolution operator is typically expressed as neighborhood aggregation or message passing scheme, which can be generally wrote as[4]:

$$\mathbf{h}_{v_i}^k = \gamma^{(k)}(\mathbf{h}_{v_i}^{k-1}, \square_{j\in\mathcal{N}(i)}\phi^{(k)}(\mathbf{h}_{v_i}^{k-1}, \mathbf{h}_{v_j}^{k-1}, \mathbf{f}_{e_{ij}}^{k-1}))$$

where $\square$ denotes a differentiable, permutation invariant function, e.g., sum, mean or max, and γ and ϕ denote differentiable functions such as MLPs (Multi Layer Perceptrons).

2 Model

2.1 Proteins as a Graph

If we want to use GCNN on proteins we have to find a general workflow for the graph generation and features extractions. For this purpose we created a python library to intuitively interface PyTorch Geometric[4] with PDB files. Our software read the input file and extract relevant chemical information to build edges, nodes features and adjacency matrix. We also used the AMBER99SD-ILDN force field to create a list of all bonds, angles and dihedral in the protein and their parameters. The nodes are connected based on K-Nearest-Neighbour (KNN, computed on a 'linear' configuration of the protein), covalent bond, contacts. Furthermore all the atoms that generate an angle or dihedral are connected together. To allow the net to send 'long range messages' we connected all the $C\alpha$ atoms. To simplify the problem the aromatic rings are represented with only 3 atoms (CD1, CD2, CG, using the Amber nomenclature) and

then reconstructed, like in the Junction Tree Variational Autoencoder[5]. In Table 1 and Table 2 we can see the nodes and edges features computed[6].

Nodes Feature	Values	N features
Is in backbone	True/False	1
Is a $C\alpha$	True/False	1
Is aromatic	True/False	1
Is in ring	True/False	1
res idx	int	1
Element	C, N, O, P, S, H	6
Residue	All residues	20
Charge	float	1

Table 1
Nodes Features

Edges Feature	Values	N features
Is covalent bond	True/False	1
Connect $C\alpha$	True/False	1
Bond type	nb (non bonded), vdw (van der Waals), hp (Hydrophobic), ts (T-stacking), ps (Pi-stacking), pc (Pi-cation), sb (Salt bridge), hbbb (H bond backbone to backbone), hbsb (H bond side-chain to backbone), hbss (H bond side-chain to side-chain)	10
Is aromatic	True/False	1
Is in ring	True/False	1
Atoms connected	C, N, O, P, S, H	6
Charges	float, float	2

Table 2
Edges Features

2.2 Convolution Layer

We tested GENConv[7] and CGConv[8] convolution layers for our model, but both had some problems for this kind of inference, so we defined our convolution layer:

$$\mathbf{h}_{v_i}^{k} = MLP^{(k)}([\mathbf{h}_{v_i}^{k-1}, SoftMax(MLP^{(k)}(\mathbf{a}_{ij}^{k}) + \epsilon : j \in \mathcal{N}(i)\}, \beta^{(k)})])$$

where $SoftMax(\{...\}, \beta)$ is a softmax with learnable inverse temperature β and $\mathbf{a}_{ij} = [\mathbf{h}_{v_j}, \mathbf{h}_{v_j}, \mathbf{f}_{e_{ij}}]$. This aggregation function allows the net to interpolate between mean ($\beta \to 0$) and max ($\beta \to \infty$),[7]. The MLP are 3 layers deep, this is computationally expensive, but lead

to a significant improvement in terms of training speed and net performance. In order to adjourn the edges features we use a MLP:

$$\mathbf{f}_{e_{ij}}^{k} = MLP([\mathbf{h}_{v_i}^{k}, \mathbf{h}_{v_j}^{k}, \mathbf{f}_{e_{ij}}^{k-1}])$$

We implemented a decoder layer to generate a graph from a global vector $\mathbf{g} \in \mathbb{R}^m$:

$$\mathbf{h}_{v_i}^{k} = MLP^{(k)}([\mathbf{g}, \mathbf{h}_{v_i}^{k-1}, SoftMax(\{MLP^{(k)}(\mathbf{a}_{ij}^{k}) + \epsilon : j \in \mathcal{N}(i)\}, \beta^{(k)})])$$

2.3 Encoder

The encoder is a sequence of Convolution and SAGPooling[9] layers, which take as input the graph and the edges length, $\mathcal{D}(\mathbf{x}) = \{||\mathbf{x}_i - \mathbf{x}_j|| : e_{ij} \in \mathcal{E}\}$. In this way we progressively reduce the graph dimensions until when we have one node with m feature, then we apply the SoftMax to get $\mathbf{z} = \chi(\mathcal{D}, \mathcal{G})$, the low dimensional representation of the protein structure.

2.4 Decoder

GCNN models suffers a lack of learnable decoders, indeed GVAE (Graph Variational AutoEncoders) are applied to link prediction problems, so usually the decoded adjacency matrix is computed as the sigmoid of the inner product of the latent representation for itself. For our particular problem, where we want to determine a new set of coordinates, those kind of approaches are impracticable. To get around this problem we used the decoder layer at the beginning of the net, with as global vector $\mathbf{g}$ the latent vector $\mathbf{z} \in \mathbb{R}^8$ concatenated to the noise vector $\epsilon \in \mathbb{R}^6$). The decoder is splitted in two part: the first act on a subgraph made of backbones atoms, with $C\alpha$ atoms fully connected and compute their next configuration, while the second take the output of the first net and build the side chain around it, this allows to avoid overfitting and the generation of too similar configurations.

2.5 Loss Function

We want to train the net in order to generate new configurations, so we adopt the energy distance loss to match the two distributions[10],[1]:

$$\mathcal{L}_{ed} = \mathbb{E}[D_E(\mathbb{P}(\mathbf{x}_{t+\tau}), \mathbb{P}(\hat{\mathbf{x}}_{t+\tau}))$$

Where $\mathbf{x}_{t+\tau} \in \mathbb{R}^{3N}$ is a vector representing the atoms positions at time $t + \tau$ and $\hat{\mathbf{x}}_{t+\tau}$ are generated configurations. Both trajectories were superposed at a average reference configuration, in order to make the loss invariant for rotations and translations. Unfortunately if we aim to generate complex structures we need and additional loss term to regularize bonds,

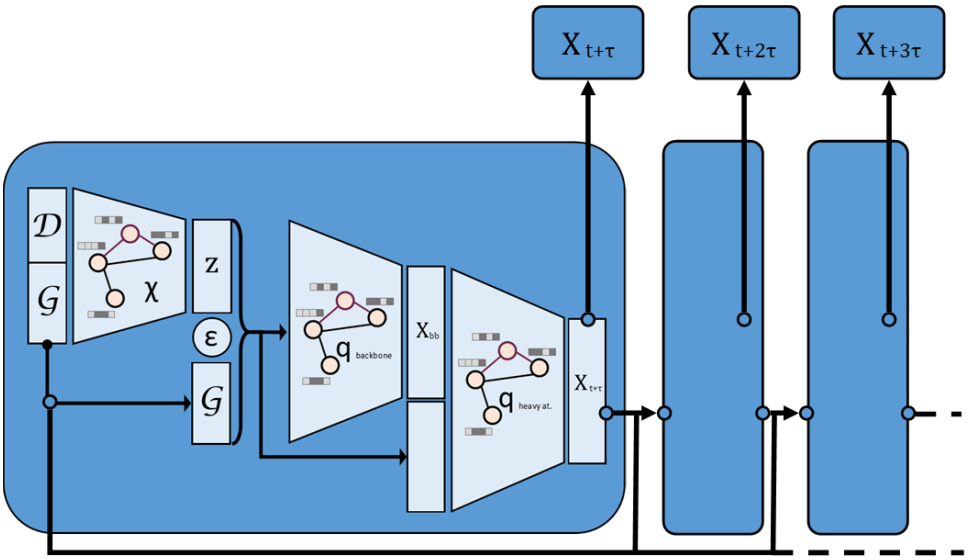

Fig. 1. The trajectory generation process: starting from a configuration x_t we compute the edges length $\mathcal{D}$ ad append them to the edges features, then we apply our net in order to generate the next set of coordinates $x_{t+\tau}$. We iterate this process using the previous layer output to generate the next input

angles and proper dihedral. The GROMACS[11] binding forces worked well for our purpose:

$$V_b(ij) = \frac{1}{2}k^b_{ij}(r_{ij} - b_{ij})^2$$

$$V_a(\theta_{ijk}) = \frac{1}{2}k^\theta_{ijk}(\theta_{ijk} - \theta^0_{ijk})^2$$

$$V_d(\phi_{ijkl}) = k_\phi(1 + cos(n\phi_{ijkl} - \phi_s))$$

$$\mathcal{L}_v = \mathbb{E}[V_b(ij)] + \mathbb{E}[V_a(\theta_{ijk})] + \mathbb{E}[V_d(\phi_{ijkl})]$$

$$\mathcal{L} = \mathcal{L}_v w + \mathcal{L}_{ed}$$

with w an hyper-parameter to merge the two losses. The parameters for $\mathcal{L}_v$ comes from AMBER99SD-ILDN force field.

3 Experiments

3.1 Data Analysis

To verify our model performances we compare the net results with real proteins MD (Molecular Dynamics) trajectories. As assay procedure we use the RMSF of the heavy atoms, computed as: $\sigma_i = \sqrt{< (\mathbf{x}_i - < \mathbf{x}_i >)^2 >}$, the Hausdorff Distance (HD): $\delta H(P,Q) = max(\delta h(P|Q), \delta h(Q|P))$, where $\delta h(P|Q) = \max_{p \in P}(\min_{q \in Q}(d(p,q)))$. To represent the trajectories configurations space we use the first 2 eigenvectors of the PCA decomposition (Principal Component Analysis computed with atoms positions superposed to an average configuration). To investigate further the realism of the configurations we compute the potential energy and the Solvent Accessible Surface Area (SASA) distribution of the real trajectory and generated structures. We estimate the potential energies using OpenMM[12] with implicit solvent model OBC2 (salt concentration = 0.15 $molar$, solute dielectric = 1.0, solvent dielectric = 78.5) and MDTraj[13] for the SASA estimation. In order to compute those quantities we have to add the hydrogen atoms that we removed, then we perform an energy minimization (with tolerance 5 kJ/mol and max 10000 iterations) in order to adjusts atoms positions.

3.2 Pentapeptide

We test our net on a pentapeptide[14]. This is a 5 amino acids protein commonly used for testing deep learning models for protein dynamics.

3.2.1 Simulation Methods

Pentapeptide has been simulated 25 times for $500\ ns$ in implicit solvent (GBSA) with GROMACS[11]. Simulations have been performed with Langevin integrator (times steps = $0.002\ ps$, temperature = $300\ K$), electrostatics interactions have cutoff of $1.5\ nm$ and hydrogen atoms are constrained to heavy atoms. Original files are available with the python package mdshare.

3.2.2 Results

We perform the described data analysis for the Pentapeptide. Each simulation has been splitted in $5\ ns$ for the training set and $495\ ns$ for the test set in order to stress the prediction and generalization ability of our model. For this protein PCA1 and PCA2 explains 94% and 96% of the variance.

	Train	All	Generated
Train	2.6 ± 0.4	3.1 ± 0.2	2.9 ± 0.2
All	3.1 ± 0.2	2.13 ± 0.09	2.83 ± 0.09
Generated	2.9 ± 0.2	2.83 ± 0.09	

Table 3
HD distances

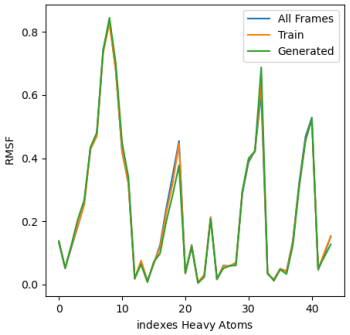

Fig. 2. RMSF of heavy atoms

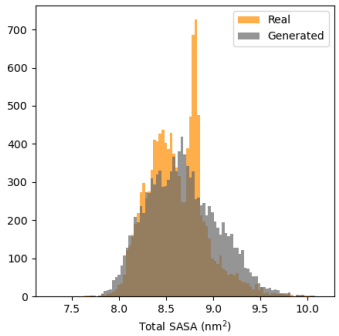

Fig. 3. SASA distribution of real configurations vs generated

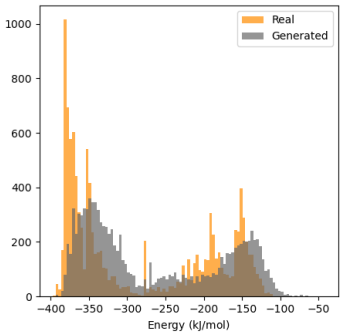

Fig. 4. Potential energy distribution of real configurations vs generated

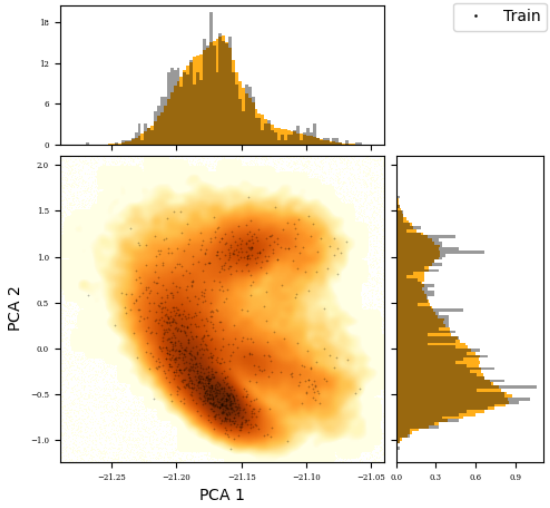

Fig. 5. PCA projection of trajectories in training set

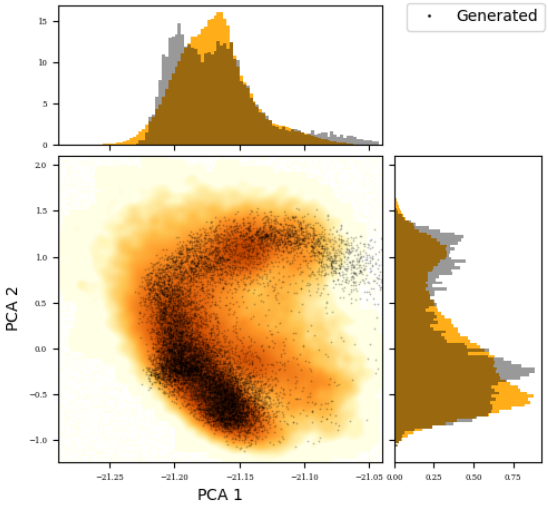

Fig. 6. PCA projection of generated trajectories

3.2.3 Discussion

We start our analysis analyzing the RMSF of the heavy atoms, seems that the net manged to achieve a great result in understanding protein flexibility. The differences between trajectories can be estimated with the Hausdorff Distance, as reference we computed the difference between real trajectories. It seems that the distance between real-real is slightly lower than

the generated-real, but still very close (Tab. 3). Then we check if the fake and real trajectory have the same distribution in the 2-dimensional space generated by the first 2 eigenvectors of the PCA decomposition. The net sampled most of the possible configurations, however the generated distributions are somehow deformed.

In Fig. 12 and 13 we can see some snapshots of the dynamics and check how the net managed to realize realistic configurations. In Fig 3 and 4 we can observe internal energy distribution of the local minima and SASA distribution: it seems that the generated configurations are energetically and geometrically really close to the real ones, so our net managed to learn the implicit solvent contribution from the position distribution.

3.3 Chingolin

We want to check if our net is able to learn the folding and unfolding process of a small protein. For this task we have chosen a mutant Chingolin (PDB ID: 5AWL), a 10 amino acids protein which exhibits a fast folding process (this protein folds and unfold each $\backsim 10\ ns$).

3.3.1 Simulation Methods

We simulated 3 times $100\ ns$ of Chingolin's MD in implicit solvent. Starting from the PDB (Protein Data Bank) file 5AWL we used tleap and ambertools in order to generate the simulation files (prmtop and prmcrd), parametrized with AMBER99SD-ILDN force field. OpenMM[12] has been used to perform the simulation. We used Langevin integrator (times steps $= 0.002\ ps$, friction coefficient $= 1\ ps^{-1}$ and temperature $= 300\ K$), while for the implicit solvent the model OBC2 (salt concentration $=$ 0.15 $molar$, solute dielectric $=$ 1.0, solvent dielectric $=$ 78.5). Electrostatics interactions have cutoff of $1.5\ nm$ and hydrogen atoms are constrained to heavy atoms. Before starting the simulation we perform energy minimization (with tolerance 5 kJ/mol and max 10000 iterations), then $1\ ns$ of MD for the system equilibration. We performed the simulation on a Linux machine with the following characteristics: CPU: Intel core i5 7600 3.8 GHz, DDR3 RAM: 8 GB, GPU: Nvidia GeForce GTX 1050 Ti (CUDA Cores: 768x1,4 GHz, DDR3 SDRAM: 4 GB).

3.3.2 Results

We perform the same data analysis as for the Pentapeptide, splitting each simulation in $15\ ns$ for the training set and $85\ ns$ for the test set. For this protein PCA1 and PCA2 explains 87% and 92% of the variance.

	Train	All	Generated
Train	5.0 ± 0.4	4.7 ± 0.1	4.6 ± 0.07
All	4.7 ± 0.1	4.5 ± 0.1	5.4 ± 0.4
Generated	4.6 ± 0.07	5.4 ± 0.4	

Table 4
HD distances

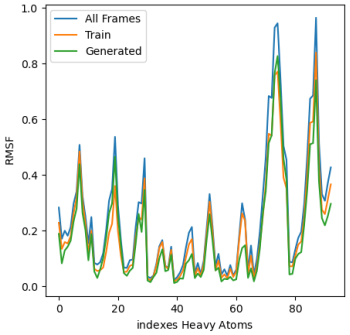

Fig. 7. RMSF of 5AWL

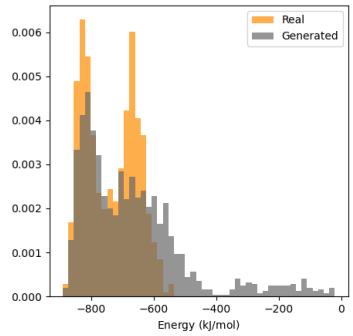

Fig. 8. 5AWL's local minimums potential energy distribution

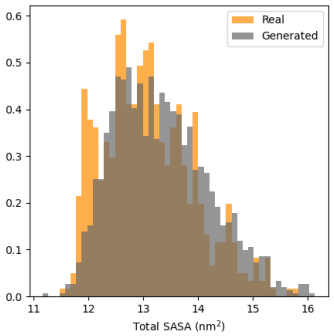

Fig. 9. SASA distribution of real configurations vs generated for 5AWL

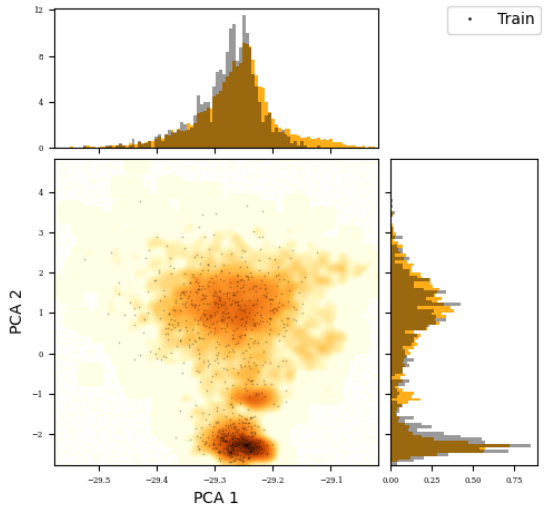

Fig. 10. PCA projection of trajectories in training set for 5AWL

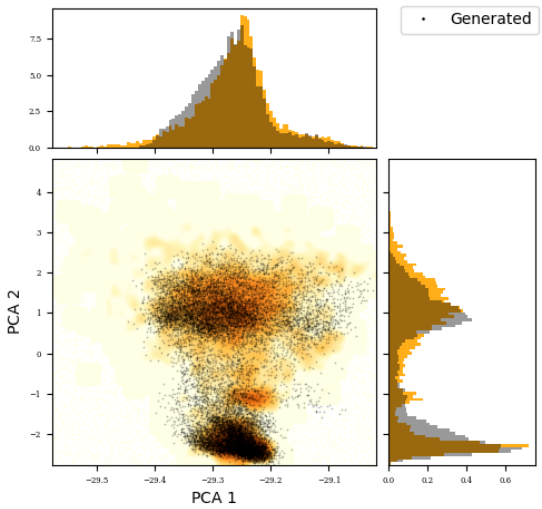

Fig. 11. PCA projection of generated trajectories for 5AWL

3.3.3 Discussion

From a comparison of Fig. 11 and Fig. 10 we can observe how the net managed to generate new configurations far from the training set, however has not been able to identify a small region of the PCA space as a metastable state. Even in this case the RMSF is correctly reproduced and Hausdorff Distance is not very informative. From Fig. 8 we can see how

the net produce configuration in the same energy range of the MD simulation and some new configurations with higher energy which are not sampled in $100\ ns$. Fig. 9 shows how that SASA distribution is qualitatively the same as the real one.

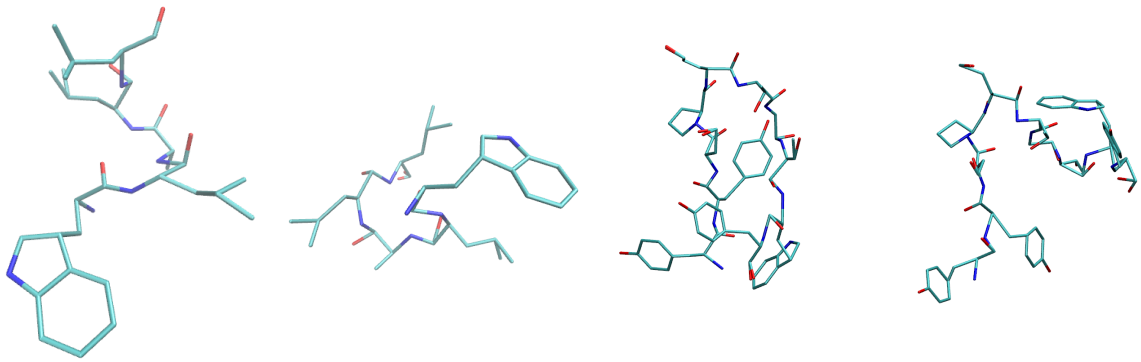

Fig. 12. Examples of generated configurations of Pentapeptide and Chingolin

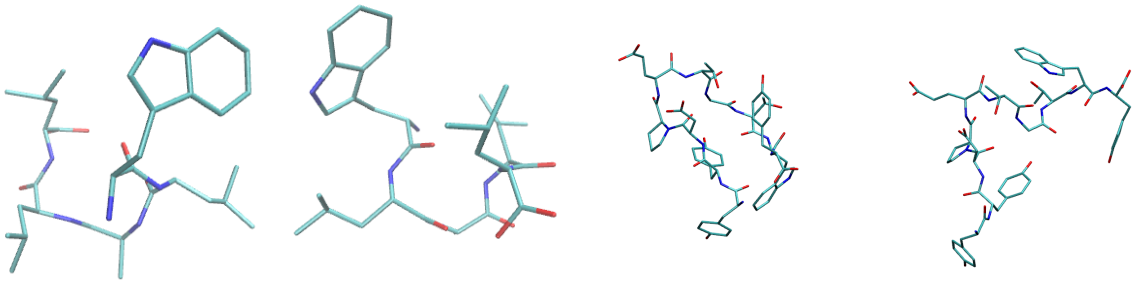

Fig. 13. Examples of real configurations of Pentapeptide and Chingolin

3.4 Deep Markov Chain Monte Carlo (DMCMC)

We want to show how our net could been applied to improve the performance of classical MCMC algorithms. We will compare the performance of the following Monte Carlo Moves on the Chingolin:

- Generalized Hybrid Monte Carlo (GHMC), a Metropolized Langevin dynamic (we chose the same parameters as before and for each step we simulate 10 ps) to propagate the system.
- Pivot Move, an algorithm which randomly choose a backbone atom (with exception of the first atom of the chain and oxygen atoms) and randomly rotate all the atoms from that point on (the maximum rotation around each axis is 2.0°).
- Deep Move (DM), a Metropolized move proposed by the neural network described before.

- Weight Move (WM), for each step choose GHMC or DM with probabilities p and 1-p.

We will check the convergence to the target distribution, the autocorrelation of energies produced, computed as[15]:

$$\rho_k = \frac{\frac{1}{M-k}\sum_{l=1}^{M-k}(\epsilon_l - <\epsilon>)(\epsilon_{l+k} - <\epsilon>)}{\frac{1}{M}\sum_{l=1}^{M}(\epsilon_l - <\epsilon>)^2}$$

. where ϵ_l is the energy at frame l and $<\epsilon>$ is its average. The parameters of the moves have been chosen in order to have an acceptance ratio ≈ 0.6 and to remove autocorrelation between consecutive samples. To run MCMC we used a customized implementation of openmmtools python library, which provide a clean workflow based on OpenMM for those type of tasks; so all our code is GPU-accelerated.

3.5 Results

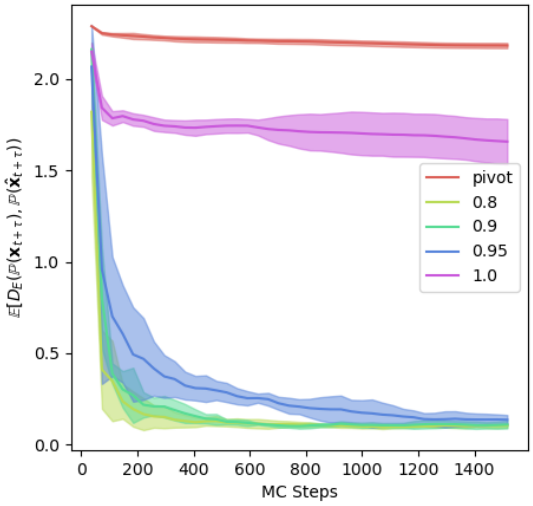

Fig. 14. Energy Distance between MCMC and MD positions probability distribution

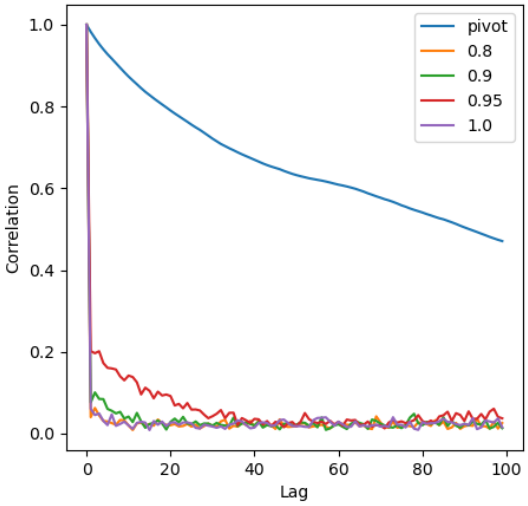

Fig. 15. Energy Correlation in function of the lag

3.6 Discussion

As we can deduce from Fig: 14 even few moves proposed by the neural network improve significantly the convergence of the Monte Carlo, this is due to the ability of our model to propose collective moves that bypass energy barriers and permit to explore a wider region of the phase space. Both the classical moves (GHMC and Pivot moves) get stacked into the folded metastable state. However we can not rely only on the neural network because we can not prove its ergodicity. In Fig: 15 we can see how almost all the moves propose statistical independent samples, with exception of the Pivot move. From the point of view of speed of

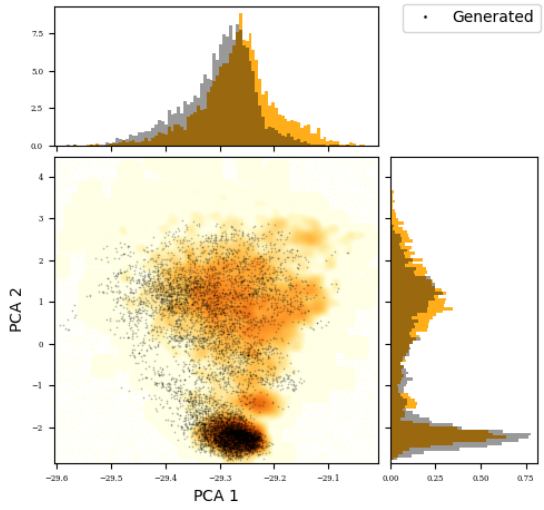

Fig. 16. distribution in PCA space of sampled configuration with WM with p=0.8

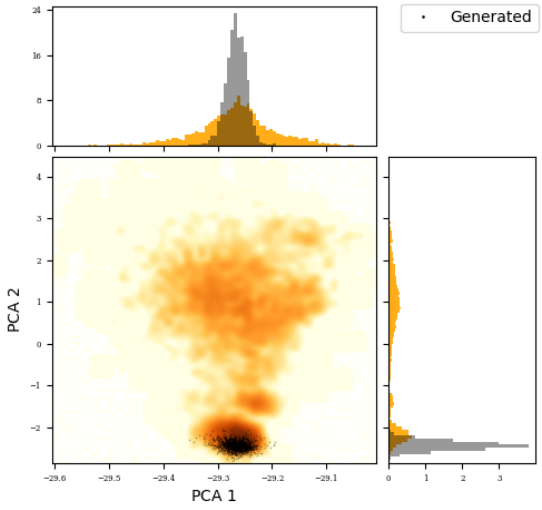

Fig. 17. distribution in PCA space of sampled configuration with GHMC

execution the Pivot move is the fastest ($\approx$ 14 iteration/s against $\approx$ 0.7 iteration/s of GHMC and $\approx$ 1.0 iteration/s of DM) but not not enough to justify the high correlation of the samples generated.

4 Conclusion

We have created a Deep Learning Model which has been able to emulate and propagate the dynamics of a small protein with sufficient precision, generating new realistic configurations. We have shown that for small proteins our net is able to sample most of the configuration space even with a really limited training set and that the generated configurations are geometrically and energetically close to the real ones. We showed how this model can be used in order to improve the GHMC's sampling and correctly reproduce the folding-unfolding dynamic of the Chingolin.

5 References

1. Hao Wu, Andreas Mardt, Luca Pasquali, Frank Noe. Deep Generative Markov State Models. *arXiv* (2019).
2. H. Wu, F. Noé. Variational approach for learning Markov processes from time series data. *arXiv* (2017).
3. Noé F. Machine Learning for Molecular Dynamics on Long Timescales. In: Schütt K., Chmiela S., von Lilienfeld O., Tkatchenko A., Tsuda K., Müller KR. (eds) Machine Learning Meets Quantum. *Physics. Lecture Notes in Physics, vol 968. Springer* (2020).
4. Fey, Matthias and Lenssen, Jan E. Fast Graph Representation Learning with PyTorch Geometric (2019).
5. Wengong Jin, Regina Barzilay, Tommi Jaakkola. Junction Tree Variational Autoencoder for Molecular Graph Generation. *arXiv* (2019).
6. GetContacts. https://getcontacts.github.io/.
7. Guohao Li, Chenxin Xiong, Ali Thabet, Bernard Ghanem. DeeperGCN: All You Need to Train Deeper GCNs. *arXiv* (2020).
8. GTian Xie and Jeffrey C. Grossman. Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties. *Physical Review Letters* (2018).
9. Junhyun Lee, Inyeop Lee, Jaewoo Kang. Self-Attention Graph Pooling. *arXiv* (2019).
10. Jean Feydy. GeomLoss (2020).
11. H.J.C. Berendsen D.van der Spoel R. van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. *Science Direct* (1995).
12. Peter Eastman, Jason Swails, John D. Chodera, Robert T. McGibbon, Yutong Zhao, Kyle A. Beauchamp, Lee-Ping Wang, Andrew C. Simmonett, Matthew P. Harrigan, Chaya D. Stern, Rafal P. Wiewiora, Bernard R. Brooks, Vijay S. Pande. OpenMM 7: Rapid Development of High Performance Algorithms for Molecular Dynamics. *PLoS Comput. Biol* (2017).
13. McGibbon RT, Beauchamp KA, Harrigan MP, Klein C, Swails JM, Hernández CX, Schwantes CR, Wang LP, Lane TJ, Pande VS. MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophys* (2015).
14. Martin K. Scherer, Benjamin Trendelkamp-Schroer, Fabian Paul, Guillermo Pérez-Hernández, Moritz Hoffmann, Nuria Plattner, Christoph Wehmeyer, Jan-Hendrik Prinz, and Frank Noé. PyEMMA 2: A Software Package for Estimation, Validation, and Analysis of Markov Models. *ACS Publications* (2015).
15. Marcos R. Betancourta. Optimization of Monte Carlo trial moves for protein simulations. *The Journal of Chemical Physics* (2011).