Modeling molecular kinetics with deep learning

Andreas Mardt

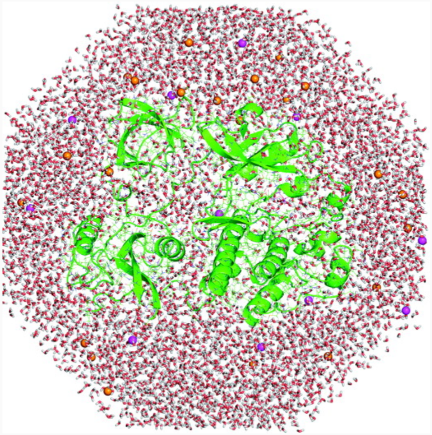
February 23, 2022

Workshop 2022

Mardt et al., "VAMPnets for deep learning of molecular kinetics.", Nature communications 9.1 (2018): 5.

Sampling Problem

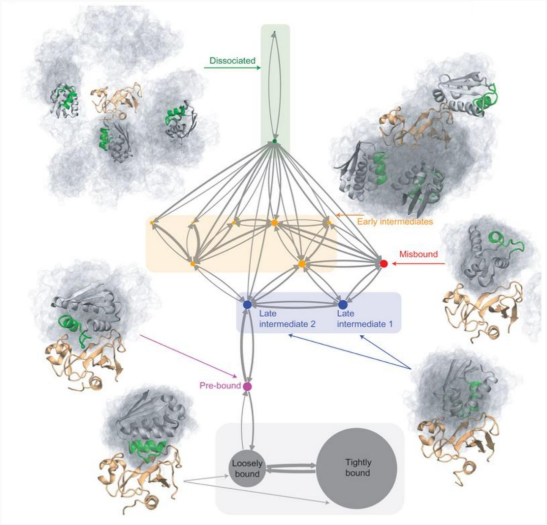
- We want to learn the stationary distribution $p(x)$ (gives access to all thermodynamic properties, conformation = one possible 3D structure) and kinetics
- Time step simulation [fs], interesting timescale [s]
- Instead of the stationary distribution learn the conditional distribution $p(x_{t+\tau}|x_t)$



Karplus et al., "Molecular dynamics and protein function". PNAS, 102.19 (2005): 6679-6685.

Markov State Models

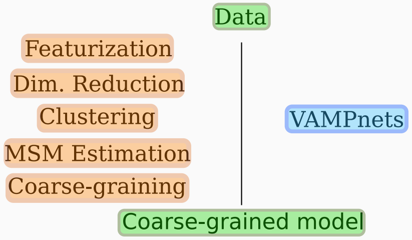
- Equilibrium simulation is not necessary $\rightarrow$ run short trajectories in parallel
- Learning $p(x_{t+\tau}|x_t)$ by mapping into a state space + transition rates
- gives access to $p(x)$ + kinetics



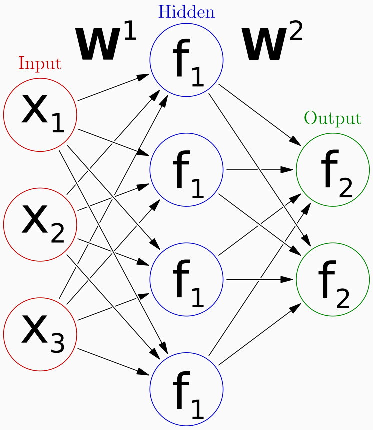
Plattner et al., "Complete proteinprotein association kinetics in atomic detail revealed by molecular dynamics simulations and Markov modelling". Nature chemistry, 9(10), 1005. (2017)

Motivation

- Handcrafted modeling is prone to errors
- We propose a Neural Network framework to substitute the whole pipeline
- Entire mapping from molecular coordinates to coarse-grained model encoded in a neural network



Neural Network



$$\hat{y} = f_2(\mathbf{W_2} f_1(\mathbf{W_1} x))$$

- Layers consist of Nodes, which are connected to the nodes of the subsequent layer
- Connections are weighted by trainable parameters $\mathbf{W_i}$
- Nodes apply nonlinear functions f_i to the sum of inputs
- Universal approximation theorem: 3-layered NNs can approximate any function

NN training

- Process of updating the parameters of the network
- Target of the update is the minimization of a given loss function L:

1. Initialize the weights $\mathbf{W_i}$ and split data
2. Until stopping criterion is reached:
    - 2.1 For every batch of training samples $x^{(i)}$:
        - 2.1.1 Compute output value $\hat{y}^{(i)}$
        - 2.1.2 Compute value of loss function $L(\hat{y})$
        - 2.1.3 Compute $\nabla_{\mathbf{W_i}} L$
        - 2.1.4 Update $\mathbf{W_i}$
    - 2.2 Evaluate L on validation set
3. Evaluate L on test set

VAMP

Given $\boldsymbol{\chi} : \mathbb{R}^n \to \mathbb{R}^m$, we want to minimize the prediction error:

$$\mathbb{E}_t[\boldsymbol{\chi}(\mathbf{x}_{t+\tau})] \approx \mathbf{K}^T \mathbb{E}_t[\boldsymbol{\chi}(\mathbf{x}_t)]$$

The optimal K is given by:

$$\begin{aligned} \mathbf{K} &= \mathbf{C}_{00}^{-1}\mathbf{C}_{01}, \text{ with} \\ \mathbf{C}_{00} &= \mathbb{E}_t[\boldsymbol{\chi}(\mathbf{x}_t)\boldsymbol{\chi}(\mathbf{x}_t)^T] \\ \mathbf{C}_{01} &= \mathbb{E}_t[\boldsymbol{\chi}(\mathbf{x}_t)\boldsymbol{\chi}(\mathbf{x}_{t+\tau})^T]. \end{aligned}$$

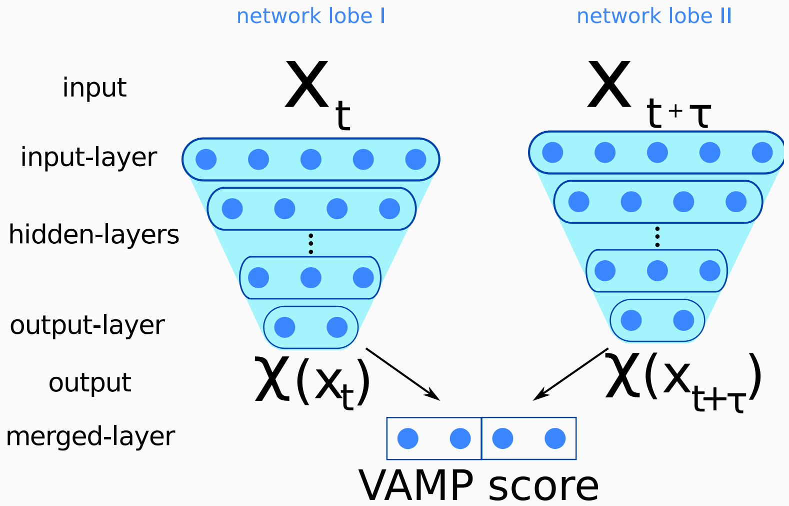
This leaves the choice of $\boldsymbol{\chi}$. The following VAMP score is maximal if $(\chi_1, ..., \chi_m)$ span the m dominant singular functions of the real $\mathcal{K}$:

$$\hat{R}_2 = ||\mathbf{C}_{00}^{-1/2}\mathbf{C}_{01}\mathbf{C}_{11}^{-1/2}||_F^2.$$

Wu, Noe, "Variational approach for learning Markov processes from time series data", 2017, arXiv:1707.04659

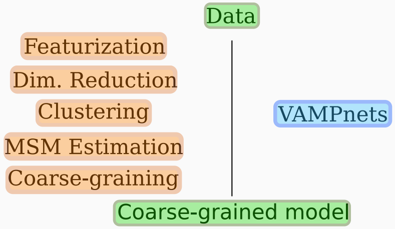
VAMPnets

- NNs are used to represent χ
- Trained to maximize the VAMP-score

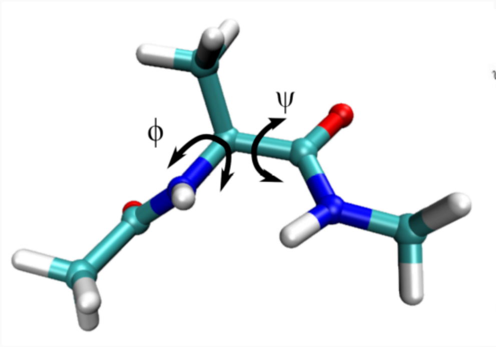


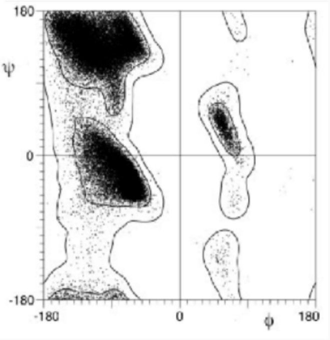
VAMPnets

- Dimensionality reduction through the lower number of output nodes
- Coarse-graining is implemented through softmax output layers
- Output nodes=number of states, value is probability to belong in that state
- We can calculate the $\mathbf{K}$ matrix and test for dynamical processes

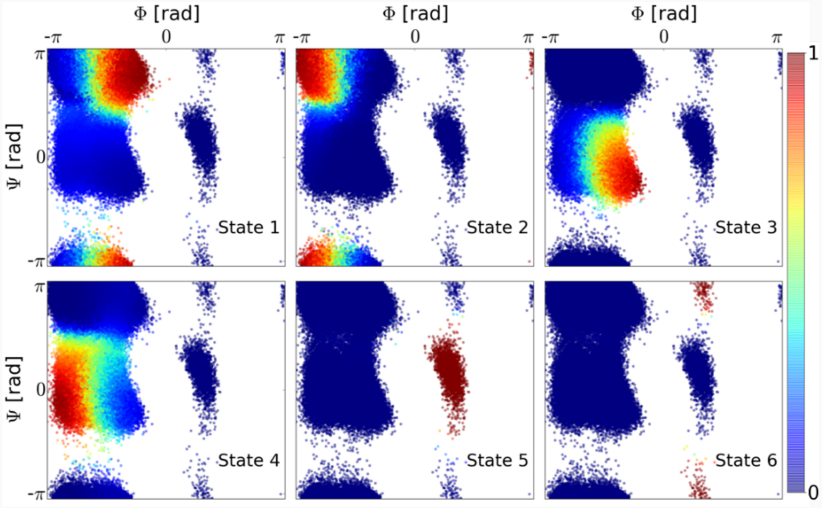


Example: Alanine Dipeptide

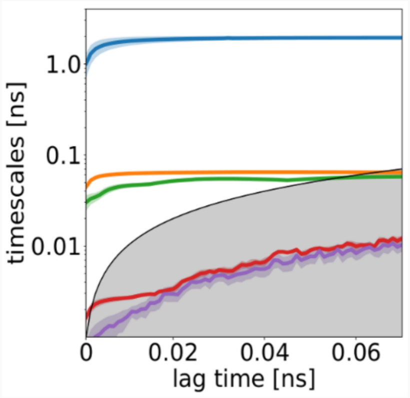


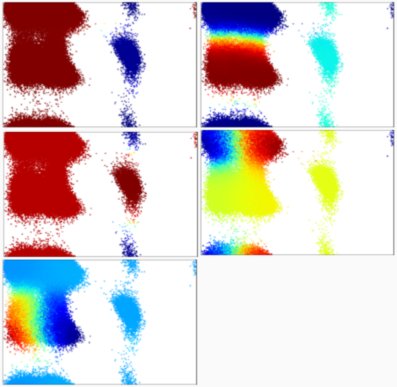


Results: Alanine Dipeptide



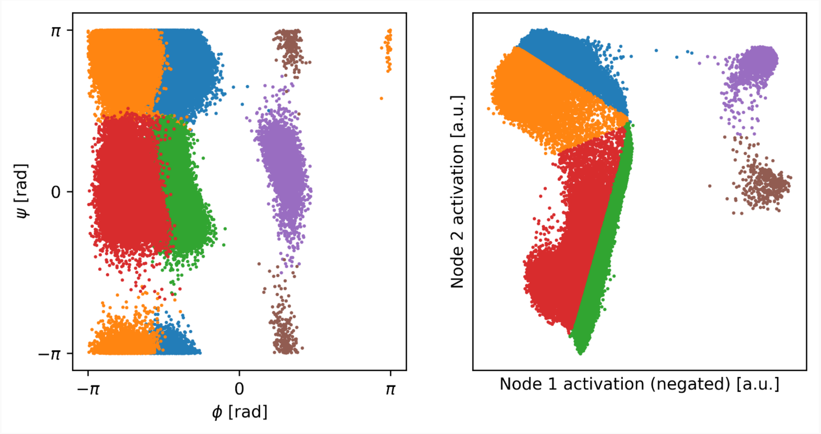
Results: Alanine Dipeptide



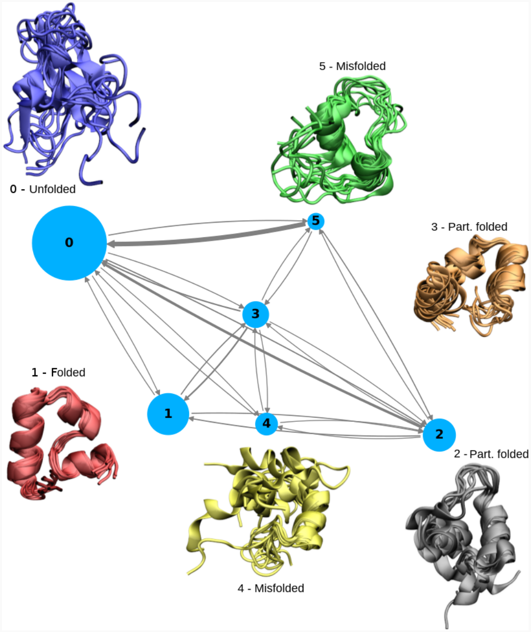


Results: Alanine Dipeptide

- VAMPnets with a 2-node bottleneck
- Network is forced to learn a 2D representation



Villin


0 - Unfolded
5 - Misfolded
3 - Part. folded
1 - Folded
4 - Misfolded
2 - Part. folded
0
1
2
3
4
5

PyTorch installation

Have you installed *PyTorch*? If not, go to https://pytorch.org, scroll down to "Install PyTorch", choose your OS and platform "CPU". Copy the conda command to your terminal, where you have activated the workshop environment, and run it.