Bachelor Thesis

VAMPnets analysis of the structural dynamics of a peptide helix

submitted by

Lennart König

Freiburg im Breisgau

Date of sumbission

25.02.2020

Supervisor

Prof. Dr. Gerhard Stock

Albert-Ludwigs-Universität Freiburg im Breisgau

Faculty of Mathematics und Physics

Institute of Physics

2020

Zusammenfassung

Proteine sind die Hauptakteure in einer Vielzahl von wichtigen biologischen Prozessen. Enzyme spalten Nährstoffe oder bauen neue Strukturen auf, während Ionenkanäle die Informationsübertragung in den Nervenzellen ermöglichen und somit teilweise für unsere kognitiven Fähigkeiten verantwortlich sind. Die Funktionalität eines Proteins ist nicht nur an die chemischen Eigenschaften der Aminosäuren, aus denen es aufgebaut ist, gebunden, sondern auch an seine Konformation und Konformationsänderungen. Daher ist die Untersuchung der Proteindynamik von wissenschaftlichem Interesse, um die Eigenschaften von Proteinen zu verstehen. Aus physikalischer Sicht kann das Protein als ein N-Teilchen-System verstanden werden, das versucht seine freie Energie mit einer Vielzahl von Randbedingungen zu minimieren. Dies stellt ein klassisches thermodynamisches Problem dar, bei dem wir Konzepte aus der statistischen Mechanik anwenden können, um die Gleichgewichtsverteilung und andere Systemeigenschaften zu finden.

Eine Molekulardynamik-Simulation (MD) (siehe 2.1) ist eine gut etablierte Methode zur Simulation der Dynamik eines Proteins durch numerische Lösung der Newton'schen Bewegungsgleichungen. Dennoch ist nach erfolgreicher Simulation eine aufwendige Analyse der Daten notwendig, um konkrete Informationen über das Verhalten der Proteine zu erhalten. Ein gängiges Analyseverfahren besteht aus drei Schritten [1]. Zuerst muss die Dimensionalität des Problems reduziert werden, da der Fluch der Dimensionalität es unmöglich macht, statistische Analysen in einem hochdimensionalen Raum anzuwenden. Gängige Methoden für diese Aufgabe sind die Hauptkomponentenanalyse (PCA) [2]. (siehe 2.4), zeitverzögerte unabhängige Komponentenanalyse TICA [3] (siehe 2.7) oder der aus der TICA hervorgegangene variationale Ansatz für Markov-Prozesse (VAMP) [4]. Als zweites wird ein Clustering-Algorithmus wie das dichtebasierte Clustering [5] (siehe 2.5) verwendet, um metastabile Konformationen, die als Zustände des Systems definiert sind, zu erkennen und zu unterscheiden. Zuletzt wird ein Markovsches Zustandsmodell (MSM) (siehe 2.6) erstellt, um die Langzeitdynamik zwischen den Zuständen zu beschreiben. Informationen über Zustandsübergänge im MSM sind besser vorstellbar als die ursprüngliche Trajektorie und ermöglichen daher ein besseres Verständnis des Proteins.

Der übliche Arbeitsablauf bei der Analyse von MD-Daten umfasst viele Schritte und erfordert vom Modellierer die Anpassung einer Vielzahl von Parametern. Daher variieren die Ergebnisse je nach der Expertise und den Entscheidungen des Modellierers. VAMP-Netze [6] (siehe 2.8) sind ein Versuch, die Analyse-Pipeline zu vereinfachen und die Abhängigkeit von der Erfahrung des Modellierers zu reduzieren, indem maschinelles Lernen (ML) mit dem variationalen Ansatz für Markov-Prozesse (VAMP) (siehe 2.7) kombiniert wird, um Dimensionalitätsreduktion, Clustering und Modellbildung auf einen einzigen Schritt zu reduzieren. ML hat in den letzten Jahren eine Renaissance durchlaufen. Die Zunahme der Rechenleistung heutiger Workstations hat die Möglichkeit eröffnet, Konzepte wie den seit den 1970er Jahren existierenden Backpropagation Algorithmus [7] auf komplexen Netzwerken zu realisieren. Die Anwendungen von ML reichen von Software für Teilchendetektoren [8] bis hin zur Unterhaltungsindustrie, wo wie im Falle von Netflix die Benutzeroberfläche automatisch an die Präferenzen der Benutzer

angepasst wird [9]. Im Allgemeinen sind ML-getriebene Algorithmen nützlich, wenn es um die Mustererkennung in großen Datensätzen geht. VAMPnets sind ein Ansatz, der versucht Zustandsmuster in MD-Daten zu finden. Das neuronale Netz, das für diese Aufgabe verwendet wird, wandelt die möglicherweise hochdimensionalen Eingabekoordinaten direkt in ein Clustering um.

In dieser Arbeit wenden wir sowohl VAMP als auch VAMPnets auf das kleine Peptid AIB_9 (siehe 3) an, um die Stärken und Schwächen der Methoden zu identifizieren. AIB_9 ist aus früheren Analysen und Studien bereits gut verstanden und daher als Testsystem geeignet. [10] Als Referenz für das Clustering von VAMP-Netzen verwenden wir ein kombinatorisches Ramachandran-Clustering (siehe 3.2.2), das Zustände auf Grund der chiralen Orientierung der Residuen trennt.
Da TICA auf AIB_9 [11] gute Ergebnisse erzielt hat und VAMP auf TICA basiert, erwarten wir ebenfalls gute Ergebnisse von VAMP. Wir vergleichen die Ergebnisse von VAMP mit dene von PCA, von der wir bereits wissen, dass sie aussagekräftige Koordinaten für AIB_9 liefern. Da VAMPnets VAMP zur Berechnung der Kostenfunktion verwenden, erwarten wir, dass wir mögliche Probleme, auf die VAMPnets stoßen könnten, durch die Analyse des Ramachandran-Clustering mit VAMP identifizieren können. Auf der Grundlage dieser Analyse sehen wir, dass VAMPnets voraussichtlich nicht mehr als 10 sinnvolle Zustände von AIB_9 finden werden. Diese Schlussfolgerung wird durch unsere Analyse mit VAMPnets bestätigt. Wir verwenden VAMPnets, um Zustände von AIB_9 zu finden. Wir zeigen, dass die im VAMPnets-Papier [6] vorgestellte tiefe kegelförmige Architektur keine gute Wahl für AIB_9 ist und versuchen, sie zu verbessern. Daher bauen wir kegelförmige Netze mit unterschiedlicher Tiefe und kurze, aber breite rechteckige Netze.
Um die Qualität der mit diesen Netzwerken vorgenommenen Clustermessungen zu bewerten, vergleichen wir sie mit dem Ramachandran-Clustering. Da eine Hauptannahme von VAMPnets ist, dass der VAMP2-Score die Qualität eines Ergebnisses misst, untersuchen wir die Beziehung zwischen der Clustering-Qualität und dem VAMP2-Score. Hier sehen wir, dass der VAMP2-Score keine ausreichende Metrik für die Qualität eines Clustering auf AIB_9. ist. Darüber hinaus sehen wir, dass die Wahl der Netzwerkarchitektur die Clusterings stark beeinflusst, was dem ursprünglichen Anspruch, die nutzerbedingte Variation der Ergebnisse zu reduzieren, widerspricht. Letztlich untersuchen wir mögliche Verbesserungen von VAMPnets und versuchen zu bewerten, ob sie ausreichen, um die in unserer Analyse aufgetretenen Probleme von VAMPnets zu überwinden.

Erklärung

Hiermit versichere ich, die eingereichte Bachelorarbeit selbständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens (lege artis) zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Bachelorarbeit eingereicht wurde.

Ort, Datum ....   Unterschrift ....

Contents

1. Introduction

Proteins are the key players in a variety of important biological processes. Enzymes split up nutrients or build up new structures, while ion channels enable information transfer in neurons and are therefore in parts responsible for our mental capacities. The functionality of a protein is not only tied to the chemical properties of the amino acids it is build from, but also to its conformation and conformational changes. Therefore, the study of protein dynamics is of scientific interest to understand the properties of proteins. From a physical point of view, the protein can be understood as a N-particle system trying to minimize its Gibbs free energy with a multitude of boundary conditions. This represents a classical thermodynamic problem, where we can apply concepts from statistical mechanics to find the equilibrium distribution and other system properties.

A molecular dynamics (MD) simulation (see 2.1) is a well established method to simulate the dynamics of a protein by numerically solving Newtons equations of motion. Still, after successful simulation, an elaborate analysis of the data is necessary to obtain concrete information about the proteins behaviour. A common way of analysis consists of three steps [1]. First the dimensionality of the problem needs to be reduced because the curse of dimensionality makes it impossible to apply statistical analysis in a high dimensional space. Common methods for this task are principal component analysis (PCA) [2] (see 2.4), time-lagged independent component analysis TICA [3] (see 2.7) or the variational approach for Markov processes (VAMP) [4] that has evolved from TICA. Second, a clustering algorithm like density based clustering [5] (see 2.5) is used to detect and distinguish metastable conformations that are defined as states. At last, a Markov state model (MSM) (see 2.6) is built to describe the long term dynamics between states. Information about state transitions in the MSM is much more conceivable than the original trajectory and therefore enables a better understanding of the protein.

The common workflow of analyzing MD data takes many steps and requires the modeler to adjust a variety of parameters. Therefore, the results vary based on the expertise and choices of the modeler. VAMPnets [6] (see 2.8) are an attempt to simplify the analysis pipeline and reduce the dependence on the modelers experience by combining machine learning (ML) with the variational approach for Markov processes (VAMP) (see 2.7) to accomplish dimensionality reduction, clustering and model building at once. ML has gone through a rennaisance in the recent years. The increase in computational power of todays workstations has opened up the opportunity to realize concepts like the back-propagation algorithm that existed since the 1970s [7]. The applications of ML range from software for particle detection [8] to the entertainment industry, where interfaces are adapted to user preferences like Netflix does [9]. Generally speaking, ML driven algorithms are useful, when there is a need of pattern recognition in large data sets. VAMPnets are an approach that tries to find state patterns in MD data. The neural network that is used for this task, directly transforms the possibly high dimensional input coordinates to state labels.

In this work, we apply both VAMP and VAMPnets on the small peptide AIB_9 (see 3) to identify strengths and weaknesses of the methods. AIB_9 is already well understood from

previous analysis and studies and therefore suitable as a test system. [10] As a reference for the clusterings of VAMPnets, we use a combinatoric Ramachandran clustering (see 3.2.2) that separates states based on the chiral orientation of the residues.
Since TICA yielded good results on AIB_9 [11] and VAMP is based on TICA, we expect VAMP to perform well. We compare the results of VAMP to PCA for which we already know that it delivers meaningful coordinates for AIB_9. Since VAMPnets use VAMP to calculate the cost function, we expect to identify possible problems VAMPnets may encounter by analyzing the Ramachandran clustering with VAMP. Based on this analysis we see that VAMPnets are not expected to find more than 10 meaningful states of AIB_9. This conclusion is validated by our analysis with VAMPnets. We use VAMPnets to find states of AIB_9. We show that the original deep cone-shaped architecture introduced in the VAMPnets paper [6] is not a good choice for AIB_9 and try to improve it. Hence we build cone-shaped networks with varying depth and short but wide rectangular shaped networks.
To evaluate the quality of the clusterings done with those networks, we compare them to the Ramachandran clustering. Since a main assumption of VAMPnets is that the VAMP2 score measures the quality of a result, we inspect the relation between clustering quality and VAMP2 score. Here we see that the VAMP2 score is not a sufficient metric for the quality of a clustering on AIB_9.
Furthermore, we see that the choice of network architecture strongly influences the clusterings, which contradicts the original claim to reduce the user-induced uncertainty. In the end we look into possible improvements to VAMPnets and try to evaluate whether they are sufficient to overcome the problems VAMPnets encountered in our analysis.

2. Theory and Methods

The following section will introduce the key concepts and terms used in this work. First we want to discuss common concepts used when generating and analyzing data of molecular dynamics simulations. Afterwards we will motivate VAMP and VAMPnets together with the theory behind neural networks.

2.1. MD Simulations

Molecular dynamics (MD) simulations are used to generate data of complex systems like proteins. Since it is difficult if not impossible to measure protein dynamics with both high precision and high time resolution in an experiment, MD simulations are well established in the field. Even though we are looking at protein dynamics on an atomistic length scale where quantum effects become relevant, MD simulations are usually classical. The Born-Oppenheimer approximation allows to separate the dynamics of the atom cores from the dynamics of the electrons.
we can use a classical effective field approach to produce the correct dynamics. Here quantum effects are described by classical potentials in a so called force field. The force field used to simulate the data set used in this work is the AMBER force field [12]. Its potential is given as

$$\begin{aligned} E_{\text{total}} = & \sum_{\text{bonds}} k_b (l - l_{\text{eq}})^2 + \sum_{\text{angles}} k_a (\theta - \theta_{\text{eq}})^2 \\ & + \sum_{\text{dihedrals}} \sum_{n} \frac{1}{2} V_n [1 + \cos(n\phi - \gamma)] \\ & + \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} f_{ij} \left\{ \epsilon_{ij} \left[\left(\frac{r_{0ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{0ij}}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} \right\} \end{aligned}$$

Here the first two terms describe the bond stretching and angle bending between the atoms as potentials of harmonic oscillators. The third term represents angle torsion and the fourth one non bounded interactions such as van-der-Waals and electrostatic forces. With this force field, a *trajectory* can be generated by using numerical integration of the Newton equations. For this task, software packages like GROMACS [13] were developed. Usually the protein is simulated together with a solvent in a simulation box that has periodic boundary conditions.

2.2. Protein structure

Proteins are long chains build out of amino acids. The sequence of amino acids is also called the *primary structure* of the protein. In the human body there are only 21 different kinds of amino acids. Each of them consists of a C_α atom that has a NH_2 - amine group and a $COOH$ - carboxyl group as functional groups. The rest chain that is also bonded to the C_α atom determines the identity of the amino acid. The amine group is linked via a peptide bond to the carboxyl group of the next amino acids. The chain of amine groups, C_α atoms and carboxyl groups is called the *backbone* of the protein. The resulting chain is illustrated in figure 1.

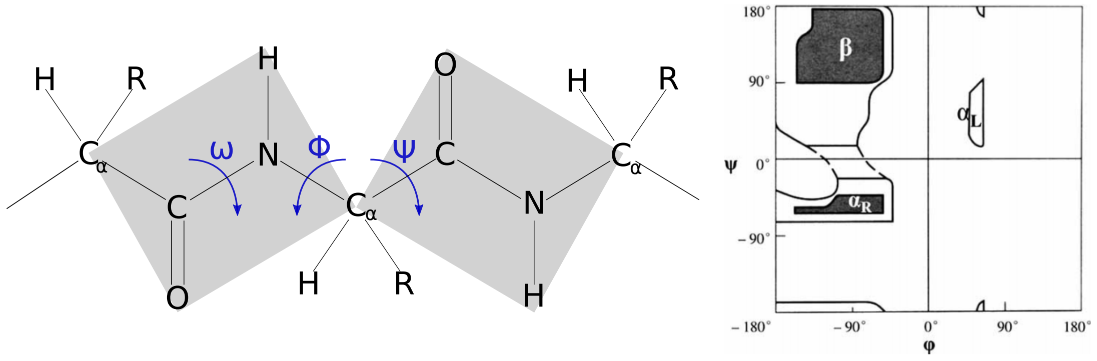


Figure 1: Left: Structural representation of the backbone and location of the dihedral angles. Right: Example of a Ramachandran plot. The regions that indicate a secondary structure are marked with the name of the structure. Images taken from [14]

As can be seen, 2 carbon and 1 nitrogen atom of the backbone belong to one amino acid. Here one amino acid is often referred as a *residue* of the protein. The conformation of the backbone has 3 degrees of freedom per residue that are given as rotations of the angles ω, ϕ and ψ. Due to the electronic configuration in the amide group, ω is approximately constant at low temperatures and the degrees of freedom can therefore be reduced to ϕ and ψ. Those 2 angles are called *dihedral angles.* A Ramachandran plot is a useful tool to visualize the average conformation of a protein at one residue by plotting its free energy landscape in dihedral coordinates. Figure 1 shows a typical free energy landscape of alanine. The locations of maxima in the free energy can be used to describe the *secondary structure* of the protein. The secondary structure describes locally formed conformations like α-helizes and β-sheets. Large proteins do also have a *tertiary structure* that describes how α-helizes and β-sheets are arranged. Residues that form β-sheets have an extremum in the upper left sector of the Ramachandran plot. The upper right sector represents a left handed α-helix, while the lower left sector indicates a right handed α-helix.

2.3. Dimensionality reduction

If we had simulated the atom cores in Cartesian coordinates this would leave us with a $3N$ dimensional trajectory where both internal dynamics of the protein and the movement of the protein in the simulation box are included. N is here the number of atoms in the protein. As far as computational processing of the data is concerned, it highly ineffective to work on this high dimensional space. Additionally the curse of dimensionality increases the amount of data needed with each dimension drastically. Therefore our goal is to find a new set of coordinates we will call collective variables (CV) that has a low dimensionality, but still separates different metastable states without distorting the proximity relation between those states. This is likely to happen due to projection errors.
A first step to reduce the number of coordinates beforehand is the choice of a suitable coordinate system. Since we are interested in the shape of the backbone, diheadral angles are a good choice that also reduces the dynamics on internal processes of the

protein. To apply algorithms that find metastable states, the space is still of too high dimensionality. Various examples of folding small proteins have shown that $d \approx 5$ is sufficient to separate states without major projection errors. [15]
There exist different methods how to find good CVs. Common techniques are principal component analysis (PCA) (see 2.4) and time-lagged independent component analysis (TICA) (see 2.7) which are later explained in detail.

2.4. Principal component analysis (PCA)

PCA is a method to perform a dimensionality reduction. It assumes that a coordinate is more important the higher the variance along the coordinate is. The motivation behind this assumption is that the relevant motion of the protein should cover wider distances than the small fluctuations insides otherwise stable conformations. If this were not the case, it would be difficult to distinguish between different states that are close to each other.
PCA is a unitary transformation, which means that it rotates the coordinate system without stretching or squishing basis vectors and without changing angles of the basis until the variance is maximized. An example of a PCA in 2 coordinates is illustrated in figure 2.

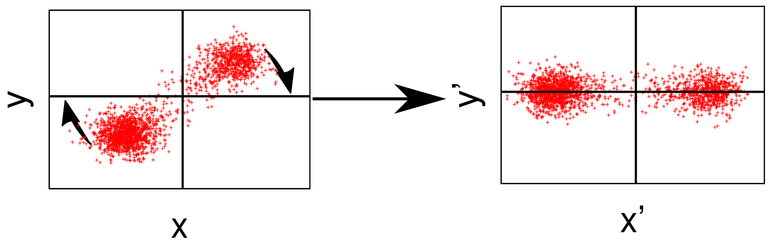


Figure 2: Visualization of a PCA in 2 coordinates

Technically this is realized by calculating the covariancematrix C of all coordinates $\vec{x}$. For a trajectory with N data points, the entries of C are given as

$$c_{i,j} = \frac{1}{N-1} \sum_{\vec{x}} (x_i - \mu_i)(x_j - \mu_j) \tag{1}$$

Diagonalizing C with eigenvalue decomposition returns its eigenbasis.

$$C = UDU^{-1} \tag{2}$$

U is an unitary matrix and D is diagonal. The column vectors of the change of basis matrices U and U^{-1} are a new set of basis vectors that is ordered by the size of their variance represented in the corresponding eigenvalue. All covariance matrices are by definition positive semidefinite. Therefore all eigenvalues are non negative. By selecting only the eigenvectors with the highest eigenvalues, the dimensionality can be reduced.

When working with cyclic coordinates, the periodic boundary constraints are problematic when calculating distances in a flat projection. To be able to perform PCA on dihedral angles as input coordinates of the MD trajectory, dPCA+ was developed. Before calculating the covariance matrix, dPCA+ performs a maximum gap shift. This shifts ensures that the highest energy barriers are on the edge where the cyclic coordinate is cut in the flat projection. That way it can be ensured that no accumulation points are cut by the edge when working directly on angles as coordinates [16]

2.5. Density based clustering

Density based clustering is a method to find metastable states in a low dimensional set of CVs. We do this by looking at the free energy landscape G in the CVs. Under the assumption of ergodicity, we can can calculate G from the density of trajectory points in the CVs.

$$G = -k_B T \ln \rho(\vec{x}) \tag{3}$$

Technically the state density $\rho(\vec{x})$ can be calculated by counting for each point how many other trajectory points are insides a hypersphere with *cluster radius* r.
We now want to identify major wells of G as states. Since G is usually very rough, our clustering algorithm must be insensitive to small unevenesses. This is done with the help of a parameter p_{min} that describes how many trajectory points are required to be in a well to form a new state. The clustering algorithm starts at the lowest value of G and counts the number of points found in each well while increasing G. Each time a new well is detected that contains at least p_{min} points, a new state is assigned to those points. If the new well merges with an already labeled state before p_{min} points are found, all new points are assigned to the state that merged with them.

2.6. Markov state model (MSM)

Markov state models (MSM) are used to estimate the long term dynamics of a systems to be able to obtain information without the need of more simulation time. This is done by approximating the dynamics as memoryless jumps between its states. Each jump evolves the system by a *lag time* of τ. The prefix "Markov" implies that the model has no memory, which means that the time evolution of a state depends only on the state itself and not previous ones. Therefore the MSM can be represented as a transition matrix T of shape $n \times n$ where n is the number of different states. Each entry $t_{i,j}$ of the Matrix represents the probability of a state j transitioning to another state i during the time τ. Mathematicians also refer to T as a stochastic matrix. [17] To get meaningful predictions of the dynamics, τ should be chosen small enough that the system is most likely to stay in its state during one time step.
Now we want to illustrate how an MSM could be build from a state labeled trajectory. The selected τ needs to be a multiple of the time step of the trajectory. We then count how often a transition between state i and j appears during the lag time τ for all possible combinations of i and j. Afterwards we normalize the counts $c_{i,j}$ for each initial state i to 1. The resulting matrix is the transition matrix T. Due to the normalization, the column vectors of T are normalized. However, the row vectors are not which means that usually transition matrices are not orthogonal.

2.6.1. Space of transition matrices

The fundamental quantity that is needed to understand VAMPnets is the implied time scale. Therefore we want to invest some time in understanding how it can be obtained from a transistion matrix and why the interpretation of being a time scale is justified for the quantity. First we have to look at the vectors, the transition matrix T operates on. Each vector has as many entries as there are different states in the system.
One interpretation is that a vector is the probability vector of one system. Therefore, after applying the transition matrix one or multiple times, the entries of the vector are the probabilities of finding the system in the corresponding state. Another Interpretation is the ensemble interpretation. We imagine having a multitude of different systems that all propagate in time independent of each other. For convenience, we normalize our vector back to 1, so that we have information about the fraction of systems that are in a given state. After applying the matrix, we get information about how many systems have changed their state. Here the transition matrix is interpreted as an operator that represents time evolution on a normalized ensemble space where systems can neither be destroyed nor generated.
Both interpretations have in common that only positive probabilities / fractions are allowed and physical meaningful. If we reduce our space to the sector where all components are positive, this leads to the slight problem our space is no longer a vector space. For each element, the inverse element is missing now. In mathematics, such a structure is called a convex cone [18] [19].
So we have to decide if we sacrifice part of our space for a better interpretation or if we work on the full vectorspace but are fine with a lack of interpretation for vectors with negative entries. Usually the second option is chosen in combination with a new interpretation for negative vectors. That's why we will still talk about vectors in the following sections.

2.6.2. Time information in transition matrices

Technically all information about the dynamics in states space are included in the transition matrix. The matrix contains for example information about the expected values how long a transition from one state to another one takes. Calculating this value turns out to be very difficult if not impossible for complex systems since the number of different pathways the system can take becomes infinite as soon as there are 3 states that can be visited during the transition. Therefore it is easier to model a markov chain with the transition matrix and calculate the times from this data.
Other information can be retrieved directly from the matrix. If a system is prepared with a randomly given vector $\vec{v}$, we expect the system to approach a stationary distribution for long time evolutions. Mathematically this can be realized by applying the matrix k times with $k \to \infty$. A stationary distribution can be characterized as a distribution where time propagation does not change anything.

$$T\vec{v} = \vec{v} \tag{4}$$

Equation (4) is an eigenvalue equation with the eigenvalue 1. The corresponding eigenvector represents the stationary distribution. When talking about equilibrium, we additionally demand the distribution $\vec{v}$ and Matrix M to fulfill the detailed balance criteria:

$$t_{i,j}v_j = t_{j,i}v_i \tag{5}$$

If a system is in detailed balance, the fluxes between two states are the same in both directions. An example for a system that has a stationary distribution that does not fulfill detailed balance is a 3 state system where the only fluxes in a circle and remaining in the state are allowed.

$$1 \to 2 \to 3 \to 1 \tag{6}$$

The other Eigenvalues and eigenvectors are more difficult to interpret. All eigenvalues despite the first one are smaller than 1 and their corresponding eigenvectors have at least one negative entry. This is due to the fact that for stochastic matizes, the sum over all components (1-norm) of an eigenvector has to be 0 if the eigenvalue is not 1. The proof can be found in the appendix in section A.1. A negative number of systems is not conceivable what makes it difficult for us to interpret those vectors as ensembles or probabilities. Furthermore an eigenvalue smaller 1 might suggest at first that systems are destroyed in during time evolution. To explain this misconception, we have to look back at how the space we operate on was defined in the previous section 2.6.1.
If we had strictly reduced our space to the positive convex cone, we would have to ignore all eigenvectors despite the first one because they are unphysical and not part of our model. Then however we would need to think about how matrices work on cones and what operations we know from vectorspaces are still allowed. What we essentially did when doing the eigenvalue decomposition, was claiming to work on the full vectorspace where we lack interpretation for $\frac{3}{4}$ of the vectors.
In the molecular modeling community the interpretation that eigenvectors of higher order represent processes rather than a distributions is deeply held [20]. A vector would then describe a process where systems flow from one state to another state or the other way around. The strength of the flow decreases over time because the eigenvalue is smaller than 1. It makes sense that the flux decreases the closer the systems approach the equilibrium distribution. Due to the eigenspace being a vectorspace, no statement about the direction of the flow can be made. The fact that all eigenvectors despite the first have 1-norm 0 supports the flux interpretation since no systems are lost or generated during the process.

2.6.3. Implied time scales

As stated above, the eigenvalues of a transition matrix T give information about how fast a process decays into the stationary distribution. The smaller an eigenvalue is, the faster a process decays. The timescale of a stationary process is infinite since the process of being in a stationary distribution never changes per definition. This is in our case associated with the eigenvalue 1. Therefore we rescale our eigenvalues and define the implied timescales t_i of an Operator $M(\tau)$ as

$$t_i(\tau) = -\frac{\tau}{\ln(\lambda_i(\tau))} \tag{7}$$

This definition is motivated from the exponential decay of the eigenvectors.

2.6.4. Chapman Kolmogorov Test

The Chapman Kolmogorov test (CK test) can be used to determine whether a system is Markovian or not. The CK equation (8) gives a condition that must be fulfilled in Markovian systems.

$$T(n\tau) = T^n(\tau) \tag{8}$$

Here $T(\tau)$ and $T(n\tau)$ are transition matrices of a Markov state model with a lag time of τ and $n\tau$. Essentially the equation says that it should not matter if the system is propagated once by a time step of $n\tau$ or n times by a time step of τ. This is only the case if the propagating matrix is independent of the point in time t when it is applied to a state. Simulations show that the equation is less accurate the smaller the lag time τ is. Larger τ reduce the information contained in the model since movements faster than τ can't be detected anymore. Therefore it is desirable model at the fastest time scale for which the CK test is still fulfilled in good approximation.
Given the definition of the implied timescales in equation (7), we now show that for Markovian systems, the implied timescale is constant as the lag time changes. We assume CK to be fulfilled. Since T can be diagonalized, a different formulation of CK equation is

$$\lambda_i(\tau)^n = \lambda_i(n\tau) \tag{9}$$

where λ_i are the eigenvalues of T

$$\begin{aligned} t_i(n\tau) &= -\frac{n\tau}{\ln(\lambda_i(n\tau))} && (10)\\ &= -\frac{n\tau}{\ln(\lambda_i(\tau)^n))} && (11)\\ &= -\frac{n\tau}{n\ln(\lambda_i(\tau)))} && (12)\\ &= -\frac{\tau}{\ln(\lambda_i(\tau)))} = t_i(\tau) && (13)\end{aligned}$$

This means that we can test the Markovianity by building transistion matrices at different lag times. When plotting the implied timescale against the lag time, we can assume Markovianity if the curve becomes horizontal.

2.7. Variational approach for Markov processes (VAMP)

The variational approach for Markov processes (VAMP) [4] has developed from time-lagged independent component analysis (TICA) [3]. TICA tries to find meaningful coordinates under the assumption that slow dynamics are also important dynamics. By selecting only the slowest coordinates, a dimensionality reduction can be done. The idea behind this approach is that a motion due to small thermal fluctuations happens on a faster timescale than a conformational change that leads to another metastable state. Therefore the main dynamics should be contained in the slowest coordinates. However, studies on alanine dipeptide have shown that this is not always the case [21]. Similar to PCA, TICA does a linear transformation when transferring to the new basis. In contrast to PCA, this transformation is not unitary since it involves a rescaling of basis vectors. Similar to PCA and TICA, VAMP projects the original coordinates of the trajectory on a new coordinate system to find meaningful coordinates. Just like TICA, VAMP is looking for a basis that maximizes timescales. While PCA used the covariance matrix and TICA uses a rescaled version of the covariance matrix, VAMP uses the so called Koopman operator to find those coordinates. This generalization should make it possible to apply VAMP to non equilibrium systems as well.

2.7.1. Koopman operator

The Koopman operator predicts the time evolution of the original trajectory coordinates for a lag time τ. Therefore the ideal Koopman operator $K(\tau)$ minimizes the average prediction error

$$\mathbb{E}_t(\vec{x}_{t+\tau} - K(\tau)\vec{x}_t) \tag{14}$$

where $\vec{x}_t$ and $\vec{x}_{t+\tau}$ are vectors of the coordinates at time t and $t+\tau$. The variational approach shown in equation (14) leads to the name of VAMP.
It can be shown [4] that equation (14) is minimzed for

$$K = C_{00}^{-\frac{1}{2}} C_{01} C_{11}^{-\frac{1}{2}} \tag{15}$$

$$C_{00} = \mathbb{E}_t((\vec{x}_t - \vec{\mu})(\vec{x}_t - \vec{\mu})^T) \approx \mathbb{E}_t((\vec{x}_{t+\tau} - \vec{\mu}_{\text{lag}})(\vec{x}_{t+\tau} - \vec{\mu}_{\text{lag}})^T) = C_{11} \tag{16}$$

$$C_{01} = \mathbb{E}_t(\vec{x}_t \vec{x}_{t+\tau}^T) \tag{17}$$

C_{00} and C_{11} are covariance matrices of the trajectory points and the time lagged trajectory points. $\vec{\mu}$ and $\vec{\mu}_{\text{lag}}$ are the mean values of $\vec{x}_t$ and $\vec{x_{t+\tau}}$. C_{01} is a cross-covariance matrix, which means that it does not share all properties of a regular covariance matrix. For example it does not have to be positive definitive. All three matrices are calculated directly from data. The equality of C_{00} and C_{11} in equation (16) is only true, if the lack of a time interval $[0,\tau]$ does not influence the expectation values of the covariances significantly.
While VAMP is designed to be applied to the coordinates of a trajectory, it can also be applied to the clustering of a trajectory. In that case, the resulting Koopman operator is exactly the transition matrix of the MSM.

2.7.2. VAMP transformation

Once the Koopman operator K is calculated from the data, VAMP uses singular value decomposition to diagonalize K. For reversible equilibrium dynamics, singular value decomposition is equivalent to the eigenvalue decomposition. Singular value decomposition is a generalized concept to bring an arbitrary matrix K of shape $n \times m$ to a diagonal-like shape where $k_{i,j} = 0$ for $i \neq j$.
For each matrix K there exists a representation

$$K = UDV^{-1} \tag{18}$$

where U and V are unitary matrices. D is a diagonal-like matrix with entries σ_i. The σ_i are called singular values of K and are distinct. If there are no degenerated σ_i, U and V

are unique. When building matrices like the Koopman operator from simulation data, the singular values can be expected to be non degenerate. U and V are the transformation matrices between the original basis and the one where the matrix has diagonal-like shape. The column vectors of U are called left singular vectors while the column vectors of V are called right singular vectors.
The interpretation of singular values differs in parts from the interpretation of eigenvalues. The largest singular value σ_1 is the largest amount a vector can be stretched by the matrix K. The corresponding singular vector $\vec{v}_1$ spans the related vector space. In contrast to an eigenvector, the transformed vector $K\vec{v}_1$ is allowed to be rotated and does not have to be part of the vector space spanned by $\vec{v}_1$ anymore.
For transition matrices, all σ_i have to be positive since otherwise the probability (or number of systems in state s) would not be conserved. Since we want to look at equilibrium dynamics, we also know that we can force detailed balance. With all those specifications, the singular value decomposition becomes equivalent to the eigenvalue decomposition. Therefore we find an orthogonal U that diagonalizes K.

$$K = U\widetilde{K}U^T \tag{19}$$

$$\widetilde{k}_{ii} := \lambda_i \tag{20}$$

$$\forall i > j : \lambda_i > \lambda_j \tag{21}$$

$\widetilde{K}$ is diagonal. U is the transformation matrix used by VAMP. The projection of $\vec{x}_t$ in the eigenbasis of K can be calculated as

$$\widetilde{\vec{x}}_t = U^T\vec{x}_t \tag{22}$$

$$\widetilde{\vec{x}}_{t+\tau} = U^T\vec{x}_{t+\tau} \tag{23}$$

Visually spoken, this eigenbasis is the set of orthogonal vectors that has the highest implied time scales. Under the assumption that a process is relevant if it is slow that the main information is contained in the first vectors of the eigenbasis.

2.7.3. CK test for VAMP

To test Markovianity, we want to check if equation (8) is fulfilled for the Koopman operator K. With K being an $k \times k$ matrix, we would have to compare k^2 values to judge if both sides of the CK equation are similar without knowing which of the matrix elements are important. Therefore it is useful to construct a score that evaluates how similar the effects of $K(n\tau)$ and $K^n(\tau)$ are on the trajectory. This is done by looking how good the predicted dynamics correlate with the real time evolution in VAMP coordinates. To do so, VAMP calculates two sets of time-lagged cross covariance matrices

$$\mathrm{Cov}_{\mathrm{est}}(n) = \langle K(n\tau)\vec{x}_t, \vec{x}_{t+n\tau}\rangle_{\rho_0} \tag{24}$$

$$\mathrm{Cov}_{\mathrm{pred}}(n) = \langle K^n(\tau)\vec{x}_t, \vec{x}_{t+n\tau}\rangle_{\rho_0} \tag{25}$$

where ρ_0 are all the points in the trajectory that have a lagged point.
$\mathrm{Cov}_{\mathrm{pred}}(2)$ for example is the cross covariance matrix of the two times propagated trajectory points and the real value of the trajectory points after 2 lag times. Since this is done in VAMP coordinates the elements on the main diagonal are the covariances of

prediction and real values for one VAMP component.
To test how similar the prediction of $K^n(\tau)$ and the estimation of $K(n\tau)$ are, VAMP suggests to compare their covariances in the main components. This means that Markovianity is given, when

$$\mathrm{Cov}_{\mathrm{est}}(n)_{i,i} = \mathrm{Cov}_{\mathrm{pred}}(n)_{i,i} \tag{26}$$

Since the covariance matrices are in VAMP basis, it is important that equation (26) is fulfilled for small i as those covariances are related to the slowest processes. For large i, we expect both matrix elements to decay to 0 for larger n, because fast processes are harder to predict on longer timescales. Since we obtain information about the Markovianity by comparing the covariances of estimates and predictions, no information can be obtained when both of them are approximately 0.

2.8. VAMPnets

Based on recent progress in the field of machine learning driven data analysis, VAMPnets [6] try to apply established software packages for building and training neural networks on MD Trajectories. Keras [22] is used to build the network while Tensorflow [23] is the most common backend that performs the training algorithms. The goal of VAMPnets is to train a network that can assign a state label to any given trajectory point without previous dimensionality reduction and clustering of the data.

2.8.1. Neural networks

Neural networks are inspired by the networks biological neurons form inside the brain. Artificial networks consist of 3 different types of neurons. Input neurons represent sensory neurons that deliver a signal to the network. Hidden neurons represent neurons of the central nervous system and are responsible for information processing. Output neurons represent motor neurons which show the reaction of the network to the signal. There are a variety of different network types that vary in the way their hidden neurons are connected. In this work, we want to focus on dense layered feed forward neural networks [24] as shown in figure 3. Here each neuron is connected with all neurons of adjacent layers. Feed forward means that each neuron can only excite neurons in its following layer but never in a previous one.

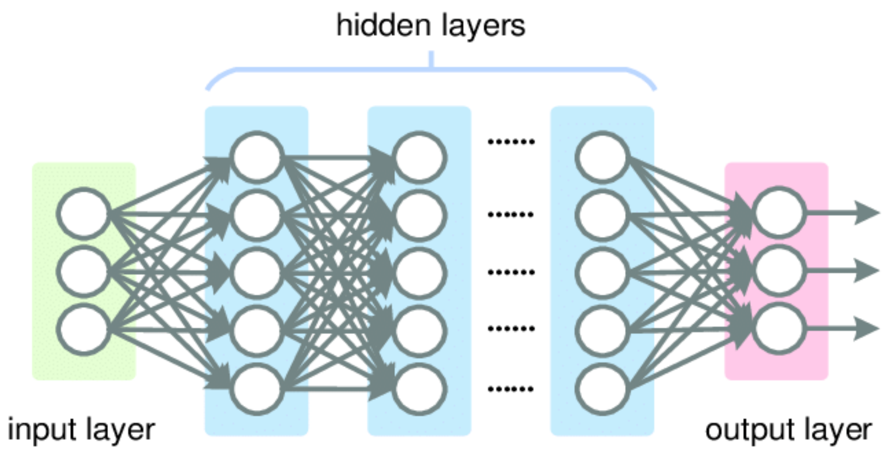


Figure 3: Schematic diagram of a feed forward neural network. Each circle is a neuron and each grey arrow is a connection between neurons. The arrow indicates the direction in which the neurons are connected. Graphic taken from [25]

Technically a layer of neurons is represented by vectors. The entries of the vector represent the excitation of the corresponding neuron. Let $\vec{v}_i$ be a set of vectors where i enumerates the layers with $\vec{v}_0$ being the input layer and $\vec{v}_d$ being the output layer. The strengths of the connections between all layers i and $i+1$ are contained in a set of matrices called M_i. The excitations of neurons in each layer $i+1$ can then be calculated from the excitatons of the previous layer i. Additionally to the standard matrix multiplication, neural networks apply a nonlinear activation function σ like the sigmoid function defined in equation (28) to the excitation that reaches the neuron in the next layer. The excitation of $\vec{v}_{i+1}$ can therefore be calculated as

$$\vec{v}_{i+1} = \sigma_i(M_i \vec{v}_i) \tag{27}$$

$$\mathrm{sig}(t) = \frac{1}{1+e^{-t}} \tag{28}$$

where σ_i takes into account that different activation functions can be chosen for different layers.

Such a network responds to a combination of excitations in the input layer with a combination of excitations in the output layer. How the response looks like depends on the matrix elements of M_i. Finding suitable M_i is the main task in ML that is achieved by training the network (see 2.8.3).
Assuming the network has n dimensional input coordinates (e.g. 10 dihedral angles) and m dimensional output coordinates (e.g. 20 states), the network can be interpreted as a function χ with

$$\chi \colon \mathbb{R}^n \to \mathbb{R}^m \tag{29}$$

The universal approximation theorem [26] states that feed forward neural networks with at least one hidden layer can approximate any continuous function with the right choices of parameters and hyper parameters in the network. Hereby, only minor restrictions are imposed on the activation functions.

2.8.2. Structure of VAMPnets

VAMPnets consist of 2 architectural identical network lobes. An illustration can be seen in figure 4. The input of both lobes are the coordinates from the MD. The output of each lobe is interpreted as a state clustering. This means that the dimension of the output layer is also the maximum number of states VAMPnets can find in the data. Network lobe 1 is used to transform the trajectory from coordinate space to state space while network lobe 2 transforms the time lagged trajectory.

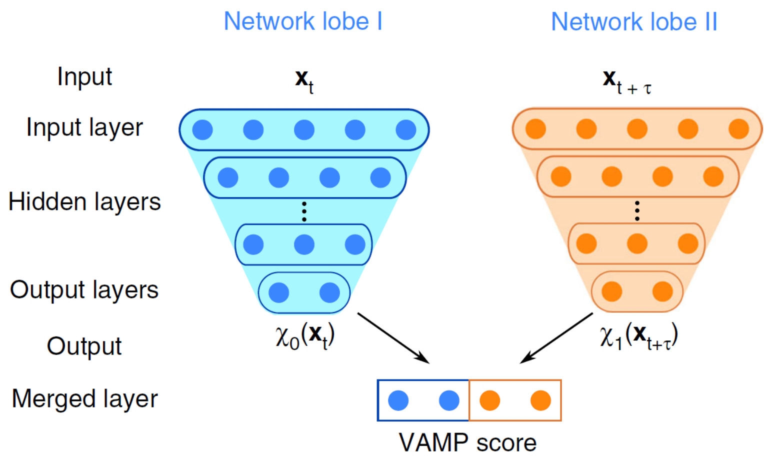


Figure 4: Architecture of a VAMPnet. Taken from [6]

As mentioned above, the outputs of the network lobes should represent state clusterings. We want each of the m output nodes to display the probability to be in the state the output node represents. This can automatically be achieved by an elegant choice in the activation function of the output layer. The softmax function, which is defined as follows:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}} \qquad \text{for } i = 1, \dots, m \qquad \text{and } \vec{z} = (z_1, \dots, z_m) \in \mathbb{R}^m \tag{30}$$

normalizes the sum over all components of the output vector to 1 and rescales all values to the interval $[0, 1]$.Taking softmax as the activation function of the output layer makes χ map into

$$\{x \in \mathbb{R}^m \mid 0 < x_i < 1, |x| < 1\} \tag{31}$$

which reduces the amount of functions that can be approximated to all functions that look like a state clustering. For all other activation functions, the sigmoid function is chosen. We will call state clusterings that assign sets of probabilities for belonging to each possible state a *fuzzy clustering*. In contrast, we will call clusterings that assign one state label for each trajectory point a *hard clustering*. A hard clustering can be obtained in different ways. The intuitive way is to select the state with the highest probability in the fuzzy clustering. It is important to mention that probabilities returned from a fuzzy clustering are artificially crafted quantities that are designed to have the properties of probabilities. It is not necessarily given that they represent real quantities of the system.

2.8.3. Training VAMPnets

To train a neural network, the network needs both data to train with and a *cost function* that evaluates how good its response to a given input was. The cost function of choice must return a scalar value that is called the *loss*. The network then tries to minimize its loss. Here the backpropagation algorithm is a useful tools to calculate a gradient of the cost function [27]
Common networks are trained by the supervised learning method. Here, the modeler must provide training data as input and the desired output of the network in combination with a loss function L. This function could for example calculate the deviation between prediction $\chi(x)$ and the desired result $r(x)$. E.g. if we wanted to learn the function $\chi(x) = 2x$, we would give the inputs $\{0, 1, 2, 3, 4\}$ with the labels $R = \{0, 2, 4, 6, 8\}$. Our loss function could be

$$L = \sum (\chi(x) - r(x))^2 \tag{32}$$

where r takes the label of x from R. Here we can see that we have to be very careful to avoid overfitting. With networks being universal function approximators, we have to build a network that is on the one hand complex enough to identify the relation we are looking for and on the other hand not too complex to be sensitive to statistical fluctuations. If there was additional noise in the data of the example above, a complex network could always fit a polynomial of 5th order to reduce its loss to 0. The complexity of a network is defined by its architecture. Therefore we must be careful when selecting hyperparameters like the number of nodes in each layer.

In contrast to supervized learning, VAMPnets use a self supervised learning method. Here the network does not need the correct answers to the inputs of the training data. Instead it calculates the loss directly from its own result by looking at the size of the implied timescales are after the transformation. This is done by applying VAMP on a batch of transformed data points $\chi(\vec{x})$. Afterwards the VAMP2 score S of the Koopman operator K is calculated. The VAMP2 score is definded as

$$S = \sum_i \sigma_i^2 \tag{33}$$

where σ_i are the singular values of K. The negative VAMP2 score is taken as loss. Additionally VAMPnets have the option to directly calculate the gradients on the loss for both network lobes. In the latest version (0.1.4), VAMPnets support tensorflows automatic gradient calculation. Therefore VAMPnets gradient calculation was not needed in this thesis. A common way to solve the minimization problem of the cost function is stochastic gradient descent since analytic methods are impracticable in high dimensional spaces [28]. Due to this numerical approach to learn χ, the training data usually needs to be shown multiple times to the network to ensure convergence. *Adam* [29] is a improved version of stochastic gradient descent that is used as an optimization algorithm for VAMPnets. After the network was trained, it can be applied on the whole trajectory to perform a fuzzy clustering.

2.8.4. Assumptions

Finally we want to talk about the assumptions that are made when claiming that VAMP-nets find an optimal state clustering when maximizing the VAMP2 score.
First, we claim that we can identify the importance of a process by its timescale. Similar to TICA [3] and VAMP [4] (which VAMPnets use), we assume that the slower a transition proceeds, the more important it is. Second, we claim that an ideal state clustering maximizes time scales of all processes. VAMPnets assume that the ideal state mapping is found, when the implied time scales of the Koopman operator in the fuzzy state variables become maximal. This can be a reasonable assumption since the highest barriers in the free energy landscape of all conformations should be in between states. The higher a barrier is, the longer it takes the system to pass it. This means that the clustering with the highest implied timescales should separate the coordinate space at the highest barriers in the free energy landscape, which matches with our intuition of a good clustering.

3. Model system AIB_9

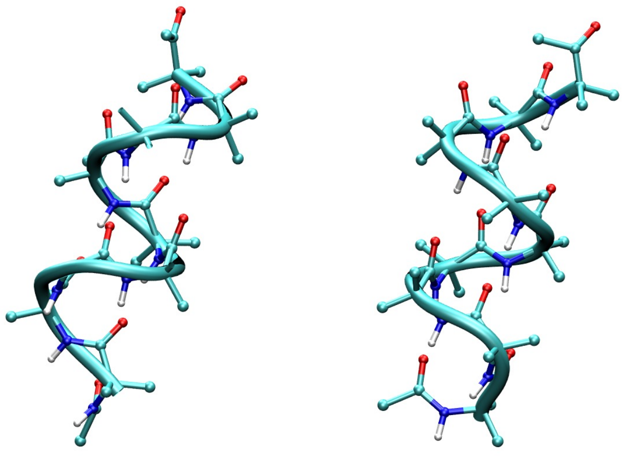


Figure 5: Structural representation of the the 2 main states of AIB_9. The image on the left shows the left handed and the image on the right the right handed α helix.

3.1. Characteristics of AIB_9

AIB_9 is a peptide that consists of 9 alanine amino acids. Due to its simplicity, AIB_9 is well understood and it is therefore suitable as first test system for a new method like VAMPnets. In contrast to most larger proteins, AIB_9 is an achiral system. In most cases each residue has a clear left (L) or right handed (R) orientation as can be seen in the Ramachandranplot (figure 8. In the two main conformations of AIB_9, the protein either forms a fully left handed or fully right handed α helix. Those can be seen in figure 5. What makes AIB_9 a non trivial test system is that the transitions between the two main states shown in figure 5 include different timescales. On short timescales, hydrogen bonds are breaking which can result for medium timescales in the swap of the chiral orientation of one residue. For long timescales, this can lead to a change of chiral orientation of the full peptide. In main pathways between a fully left and fully right handed orientation, the chiral orientation changes at one end of the protein and step by step swaps over all orientation of the proximate residues until the other end of the protein is reached.

3.2. AIB_9 Meld data set

The data used in this work was generated with the "Modeling Employing Limited Data" (MELD) method. MELD is an enhanced sampling scheme that can accelerate MD simulation with the use of information the modeler already has about the system. It is a Bayesian approach that tries to build a physical correct model from limited data obtained e.g. in an experiment. MELD increases the sampling rate of rare events which results in shorter simulation times while the resulting free energy landscape remains qualitatively similar [10] Detailed balance is preserved. The simulation produces a multitude of short trajectories. In our the data contains 5.5 million trajectory points which is represents a total simulation time of 221 μs with a time step of 40 ps between two frames. The length distribution of trajectories is shown in figure 6. We see that most trajectories have 350 - 450 points. Even though AIB$_9$ has 9 residues, we ignore the outer 2 on both sides what reduces the number of relevant residues to 5 and the number of corresponding dihedral angels to 10. This is done because the dynamics of the outer residues underlie strong fluctuations.

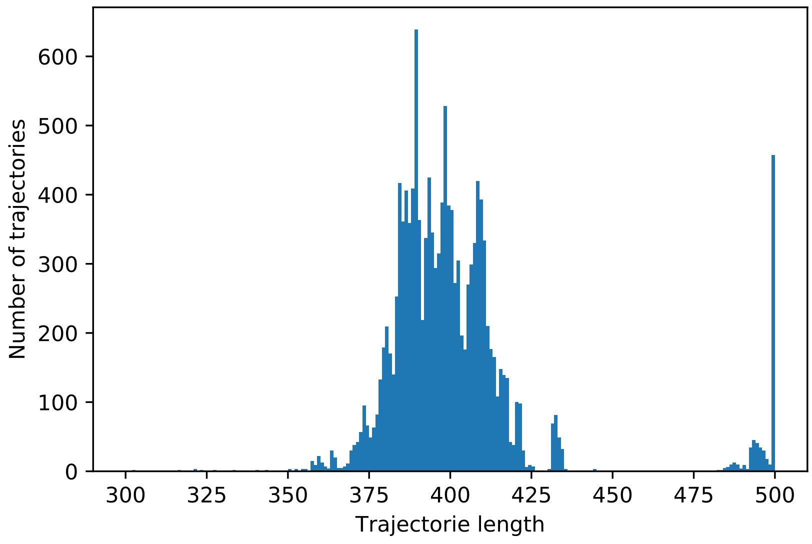


Figure 6: Distribution of trajectory lengths in the MELD data of AIB$_9$. There are 13805 trajectories in total. The shortest has 35 frames (1.4 ns) while the longest has 500 (20 ns) frames. Only few trajectories have less than 300 frames and are omitted in the histogram.

3.2.1. Density based clustering

To evaluate how good the projection of VAMP and the clustering done with VAMPnets are, we need to know how to identify a good result. Therefore we want to use established methods as benchmarks for VAMP and VAMPnets.
Previous study have shown that PCA yields good results for AIB$_9$. [10]. Figure 7 shows the free energy landscape of AIB$_9$ projected to the main two components found with PCA. Additionally, a density based clustering was done to find the main states and their populations. In the plot, the location of the 10 main states is labeled with their molecular representation. The clustering was done on the first 5 PCs with a clustering radius of 0.34 and multiple different values of p_{min}. For the clustering shown in figure 7, $p_{min} = 200$ was chosen.

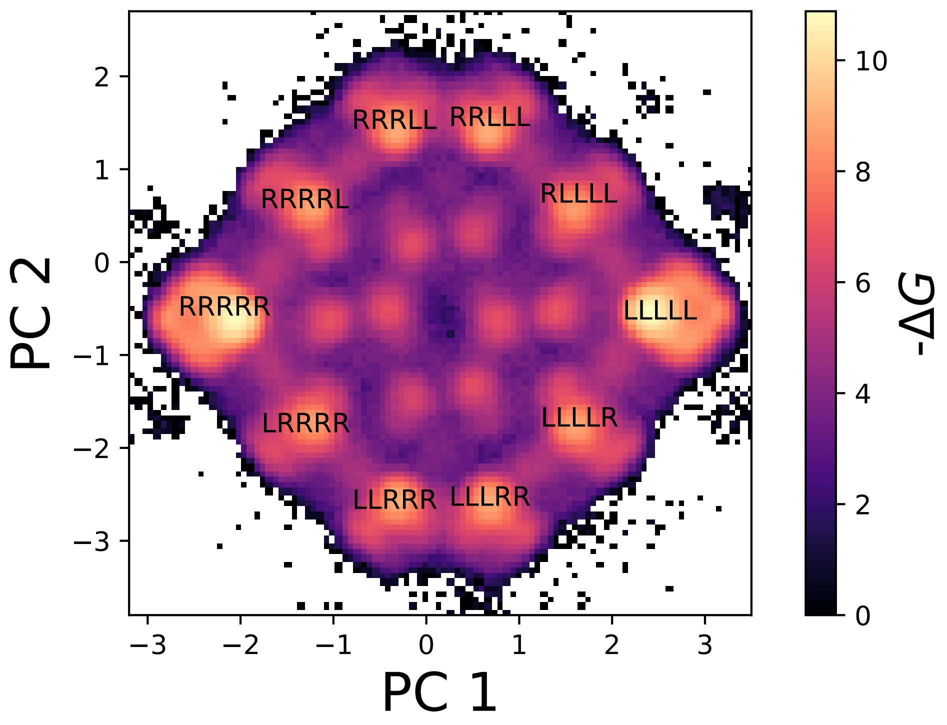


Figure 7: Free energy landcape of AIB$_9$ in the first two principal components. The location of the 10 most populated states found with density based clustering is labeled with their molecular representation (LLLLL,...). The representations are based on the average location of states in the Ramachandran plots of the inner 5 residues.

3.2.2. Ramachandran clustering

Another method to find different metastable states uses the Ramachandran plots of the 5 residues. We will call this approach Ramachandran clustering. Here we use the knowledge that in most cases, the residues of AIB$_9$ have a clear right or left handed orientation as can be seen in figure 8.
Points in the top right α_l-region have a left handed orientation (L) while points in the bottom left α_r-region have a right handed orientation (R). As can be seen, the top left β-region is very low populated. This means that for each residue, we can assign the attribute left (L) or right (R). This is done by looking whether the ψ value of the point is above or below the line

$$\psi = -0.2 \cdot \phi \tag{34}$$

This separation is designed in a way that it cuts through the highest potential barriers in the free energy landscape of the Ramachandran plot. As stated above, for each of the 5 residues either L or R is assigned. Following the laws of combinatorics, this results in $2^5 = 32$ different possible states.

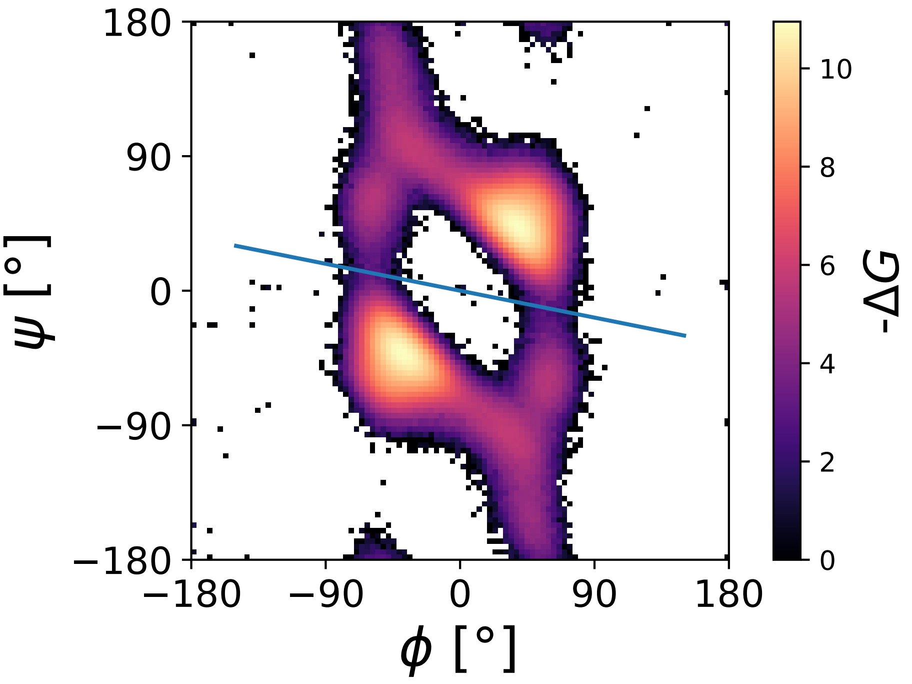


Figure 8: Ramachandran plot of the first residue of AIB_9. The blue line shows the separation made by the Ramachandran clustering

Table 1 shows the results of a Ramachandran clustering provided by Matthias Post. The table also shows the population of the states.

Both density based clustering and Ramachandran clustering show that the two most populated states of AIB_9 contain over 60% of all trajectory points. Due to the interpretation coming from the Ramachandran plots, they can be identified as states where the peptide forms a helical structure. It is either fully left- or fully right-handed. We will refer to the Ramachandran states 1 (LLLLL) and 2 (RRRRR) from table 1 as the main states. The next 8 most populated states represent conformations where only one end of the helix is twisted in the other orientation (e.g. LLLRR). We will refer to those states as major states. In the projection on the first 2 PCs (see fig 7) they can be found as outer ring in the free energy landscape. The least populated states are those where the orientation in the middle of the protein differs from the orientation at the ends (e.g. LRLLR). We will refer to those states as minor states. They can be found in the center of the projection to the first 2 PCs.

state	interpretation	population	%
1	LLLLL	1716645	31.03
2	RRRRR	1701847	30.76
3	LLLRR	249536	4.51
4	LLRRR	246444	4.45
5	RRRLL	240263	4.34
6	RRLLL	235933	4.26
7	LLLLR	222508	4.02
8	RRRRL	219050	3.96
9	LRRRR	188590	3.41
10	RLLLL	182983	3.31
11	LLRLL	32730	0.59
12	RRLRR	30835	0.56
13	RRRLR	29139	0.53
14	LLLRL	27599	0.50
15	RLLLR	26837	0.49
16	LRRRL	26169	0.47
17	LRLLL	24979	0.45
18	RLRRR	24808	0.45
19	LLRRL	22956	0.41
20	RRLLR	22206	0.40
21	LRRLL	18156	0.33
22	RLLRR	17711	0.32
23	RRLRL	4182	0.08
24	LLRLR	4125	0.07
25	LRLRR	3072	0.06
26	RLRLL	2996	0.05
27	LRRLR	2635	0.05
28	LRLLR	2529	0.05
29	RLRRL	2333	0.04
30	RLLRL	2237	0.04
31	RLRLR	356	0.01
32	LRLRL	344	0.01

Table 1: Populations of states found with the Ramachandran clustering. The interpretation represents the chiral orientation of the inner 5 residues.

4. VAMP analysis of AIB$_9$

Before inspecting how VAMPnets cluster AIB$_9$, we want to investigate if VAMP finds a reasonable projection of the data. Furthermore we are interested in finding a suitable lag time τ that can be used by VAMPnets as well, assuming both approaches behave similar.

4.1. VAMP on dihedral angles

VAMP was applied on the MELD data with different lag times. The input coordinates for VAMP were the usual 10 dihedral angles. To visualize the transformation that VAMP found, we plot the data along the PCs on the x axis and along the VAMP coordinates on the y axis for the 5 main coordinates. In that way, correlations between the PCs and the VAMP coordinates should become visible. For an exemplary lag time of $\tau = 1.2$ ns, this is shown in figure 9. To see if points belonging to the same state are projected in the same areas, the states are colored differently. The state labels are based on the density based clustering discussed in section 3.2.1. Figures 29 and 30 in the appendix show the same plots for very short ($\tau = 0.04$ ns $= 1$ frame) and long lag times ($\tau = 6.4$ ns $= 160$ frames).

The diagonal elements of figure 9 show that there is a strong correlation between the main components of VAMP and PCA. This shows that vectors in the direction of the highest variance are also the vectors that represent the slowest processes. So, VAMP performs well for AIB$_9$.
The correlation becomes stronger for lower lag times as can be seen in figure 29 in the appendix. Here the 4th and 5th component of VAMP is multiplied by -1. Since we operate in a vectorspace, the orientation of the basis does not matter. Figure 30 in the appendix shows that the correlation decreases with higher lag times. Here the correlation of the first 2 components is still significant. Higher components do not correlate strongly anymore with their correlation decreasing as the index of the components increases. The cross correlation between different components is also reduced. This aligns well with the idea that dynamics on timescales closer to the lag time are more difficult to resolve than dynamics that are magnitudes slower than the lag time. Furthermore, the fact that the MELD data is a combination of many short trajectories leads to problems with high lag times. To calculate the Koopman operator, we need to calculate $C_{01} = \mathbb{E}_t(x_t x_{t+\tau}^T)$ which includes pairs of data points and lagged data points.
The number of pairs N_C used to calculate C_{01} is

$$N_C = L - \tau \tag{35}$$

where L is the length of the trajectory and τ is the lag time in frames. If τ reaches the order of L, we construct C_{01} with very limited information since we can not make a prediction about the time evolution of most data points anymore. For example, for a lag time of $\tau = 7$ ns (175 frames), the calculation of C_{01} would only use half the data it would for the smallest lag time of $\tau = 0.04$ ns (1 frame) as can be seen in figure 6. Additionally, since the trajectories have different lengths, we ignore all points from trajectories shorter than the lag time. Summing up, it can be said that we want to choose the lag time as short as possible.

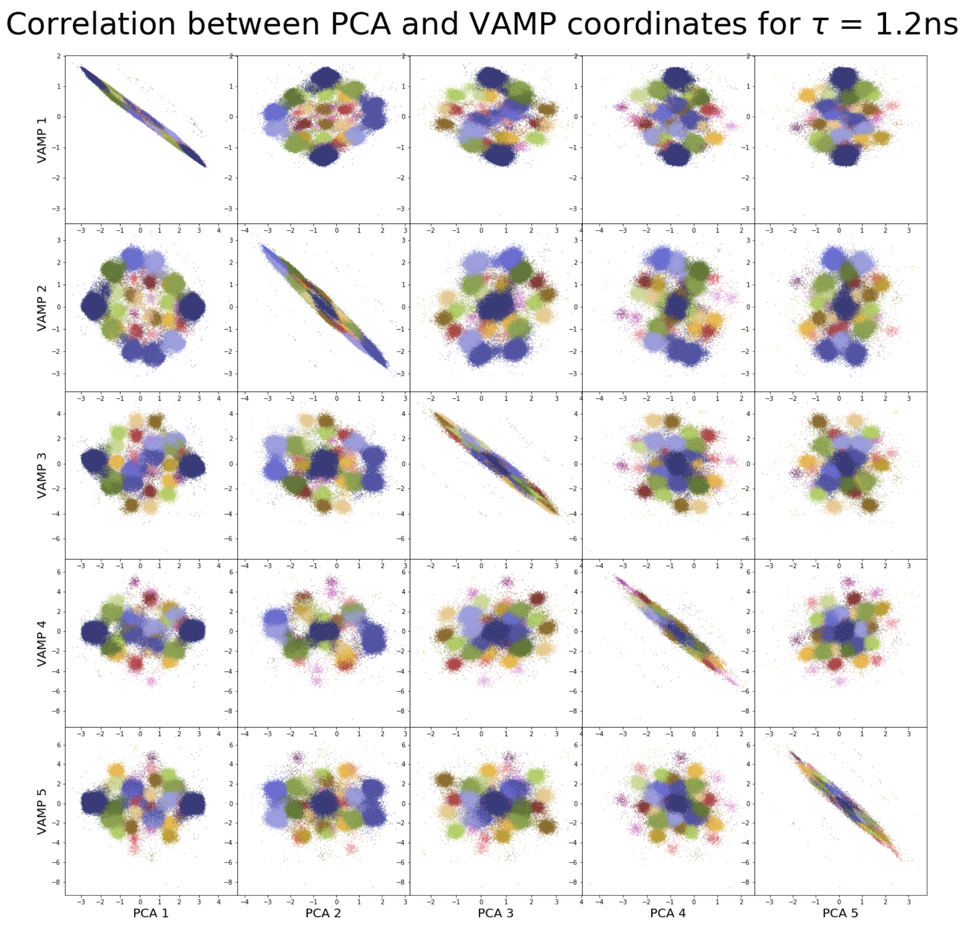


Figure 9: Projections by VAMP (y-axis) and PCA (x-axis) of the AIB$_9$ MELD data points. Every 10th point is plotted. For VAMP a lag time of $\tau = 1.2$ ns (30 frames) was chosen. The color encodes the state label found with density based clustering.

4.2. Chapman Kolmogorov test

Even though short lag times result in a better data basis, we still want to ensure that the underlying dynamics VAMP observes are Markovian. Therefore we make a CK test as described in section 2.7.3. Figure 10 shows how the covariances of estimates ($K(n\tau)$) and predictions ($K^n(\tau)$) change with increasing n. Only the covariances of the 4 main VAMP components are plotted as columns. Each row shows the CK test for a different lag time.
As can be seen, the CK test of the first component yields high deviations between estimates and predictions for short lag times. The higher the lag time becomes, the closer are the the covariances of estimates and predictions. This gives us a lower border on the possible lag times. Based on the plot we decide that the CK criteria is sufficiently fulfilled at a lag time of $\tau = 1.2$ ns.
Figure 10 also shows that the covariances of higher components decay faster to 0. This is logical since those components represent faster processes. The plot shows that the 4th component already decreases to 0 for small n. Therefore we base our choice for a Markovian lag time only on the first 3 components. Nevertheless we have seen in figure 9 that the fourth and fifth VAMP components are very reasonable as well for $\tau = 1.2$ ns.

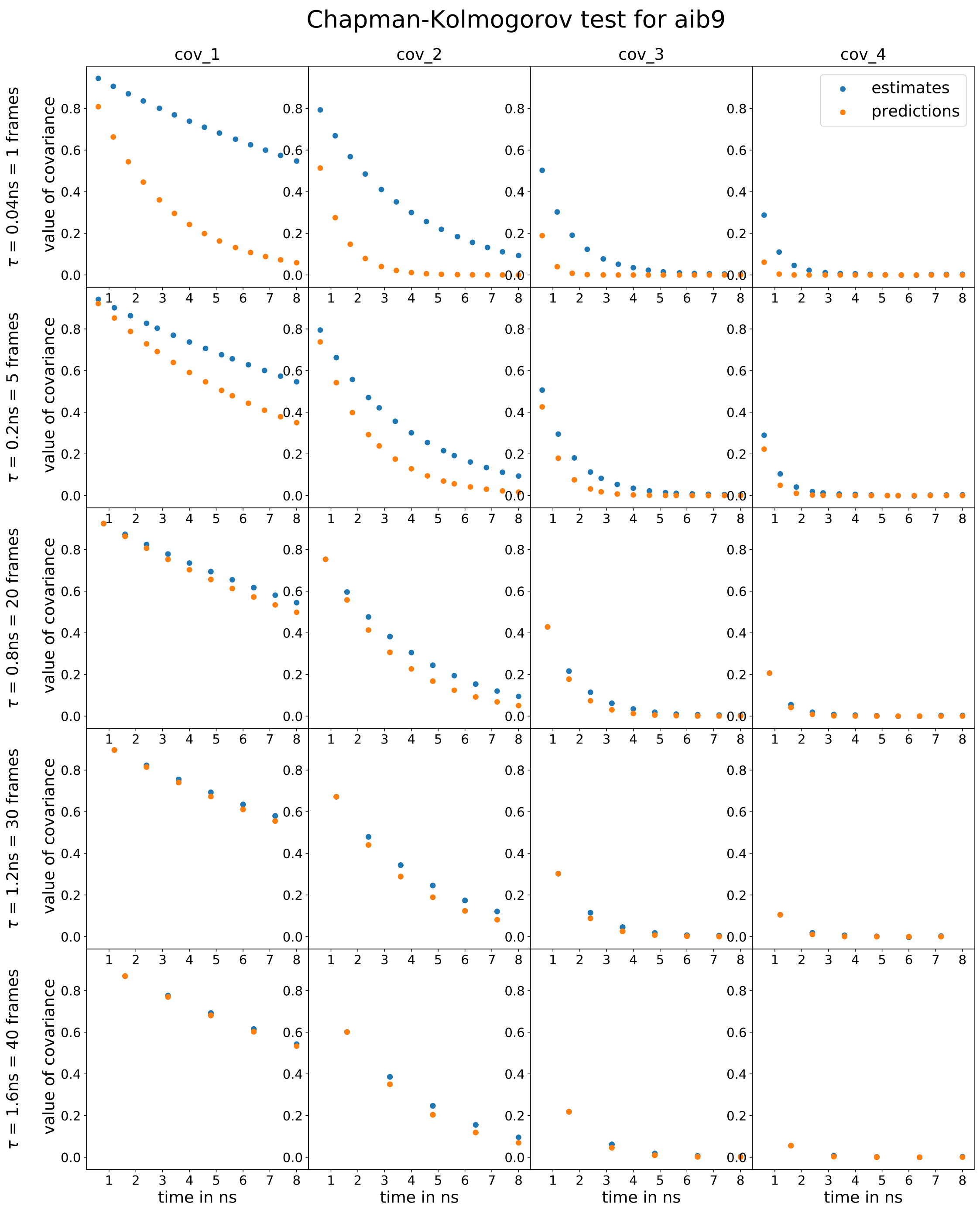


Figure 10: CK test of VAMP models build on the AIB_9 MELD data with different lag times. Each column show the covariances of a VAMP coordinate at different lag times. Each row shows the covariances of the 4 main VAMP coordinates at the same lag time.

4.3. VAMP analysis of the Ramachandran clustering

Now we want to apply VAMP on the Ramachandran clustering to check if the assumption (see 2.8.4 of VAMPnets that timescales are increased when going from space-like coordinates to state-like coordinates is fulfilled for AIB$_9$. This can be done by comparing the singular values of the Koopman operators and the VAMP2 scores from VAMP on dihedral angles with VAMP on the Ramachandran clustering. The Ramachandran clustering is a hard clustering. VAMP can only be applied to multidimensional trajectories. Therefore we vectorize the Ramachandran clustering by labeling each trajectory point with a 32 dimensional array where all entries are 0 despite the entry of the actual state which is set to 1. So the entries of the array can be interpreted as probabilities just as the output of VAMPnets. On this new fuzzy-like clustering we apply VAMP with the same lag time as used for the dihedral angles.
Figure 11 shows the singular values of both the Ramachandran clustering and the dihedrals. Even though the largest singular value is expected to be 1 for the Ramachandran clustering, it can not be found in figure 11. This is because VAMP and VAMPnets remove it and add it in hindsight again to ensure that the largest singular value is really set to 1 after decorrelating the basis [4].

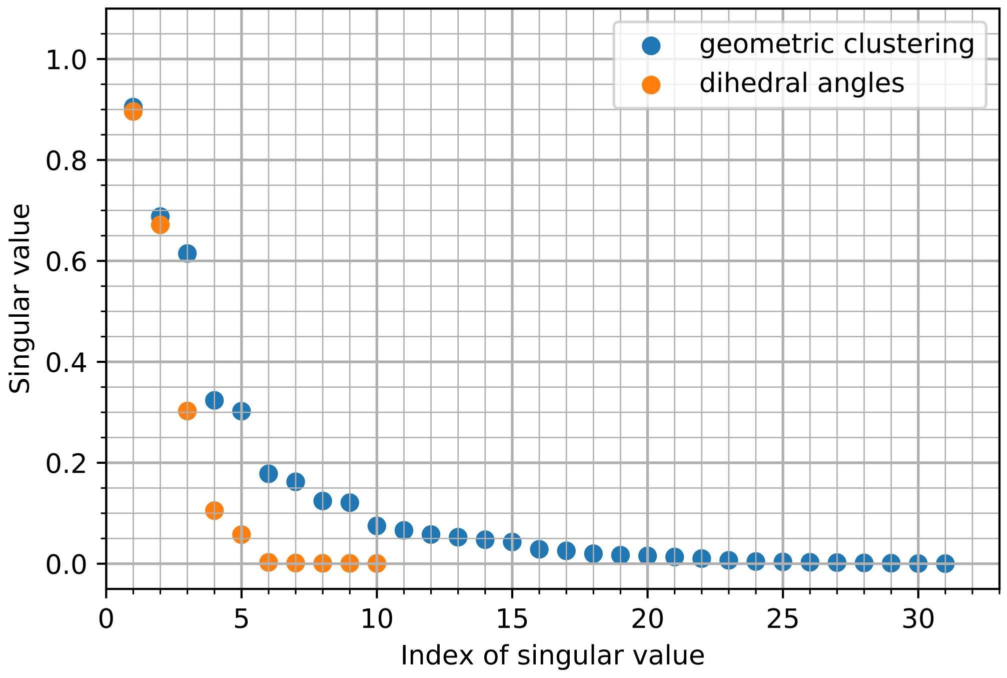


Figure 11: Singular values of Koopman operators on the dihedral angles and the fuzzy like Ramachandran clustering

It can be seen that the first 2 singular values are very similar for both choices of coordinates. Those two could represent main transitions of states in the outer ring of the

projection to the 2 main VAMP coordinates (see figure 7). For all other singular values, those of the Ramachandran clustering are significantly larger than the ones of the dihedral angles. This means that the timescales of non main processes are increased when transitioning from space- to state-like coordinates for AIB$_9$. It is also worth noting that the smallest 5 singular values of the dihedrals are close to 0. This indicates that all relevant processes can be kept when reducing the dimensionality to a 5 dimensional space. So, VAMP can give a hint how many collective variables should be kept for clustering algorithms.
In comparison, the decay of the singular values on the Ramachandran clustering is much smoother and there are ≈ 80 of them close to 0. We also see that there are only 31 singular values plotted in figure 11 even though the Koopman operator of the Ramachandran clustering is a 32 dimensional matrix. This could be due to a numerical issue when calculating the singular values which made VAMP ignore one value.
Figure 12 shows the VAMP2 scores in relation to the number of singular values used to calculate the score. Since VAMP2 is the sum over the squared singular values, the impact of small singular values is negligible. We see that the VAMP2 score of the Ramachandran clustering is more than 0.6 larger than the VAMP2 score of the original trajectory in dihedral angles. This validates the idea that the timescales are increased in state-like coordinates. Assuming that the Ramachandran clustering is close to the best clustering we can get for AIB$_9$, this leads to the expectation that a VAMP2 score of 2 (or 3 if we take the 1 into account that VAMP adds in hindsight) is an upper border for all possible clusterings of AIB$_9$. For VAMPnets, this means that no matter what feature transformation χ the network learns, the VAMP2 score should not exceed 2. VAMPnets ignore the additional 1 for the calculation of the VAMP2 score in the training.
Figure 12 also shows that the VAMP2 score only increases marginally when using more than 9 singular values. This means that for AIB$_9$ it is possible to find a clustering to 10 states (1 more than important singular values) that leads to the same VAMP2 score as the Ramachandran clustering to 32 states.
Looking back at the state distribution of AIB$_9$ in the Ramachandran clustering, the 10 most populated states are exactly the main and major states. Still, it is not given that a clustering to 10 states with a VAMP2 score of 2 finds the 10 most populated Ramachandran states. Figure 12 only showed that there are 9 important processes in the clustering to 32 states. It did not show that those processes are in the 10 most populated states.

VAMPnets only possibility to evaluate the quality of a clustering is the VAMP2 score. This is problematic since we saw that a So, it is expected that VAMPnets have problems in finding more than 10 states while struggling to distinguish minor states.

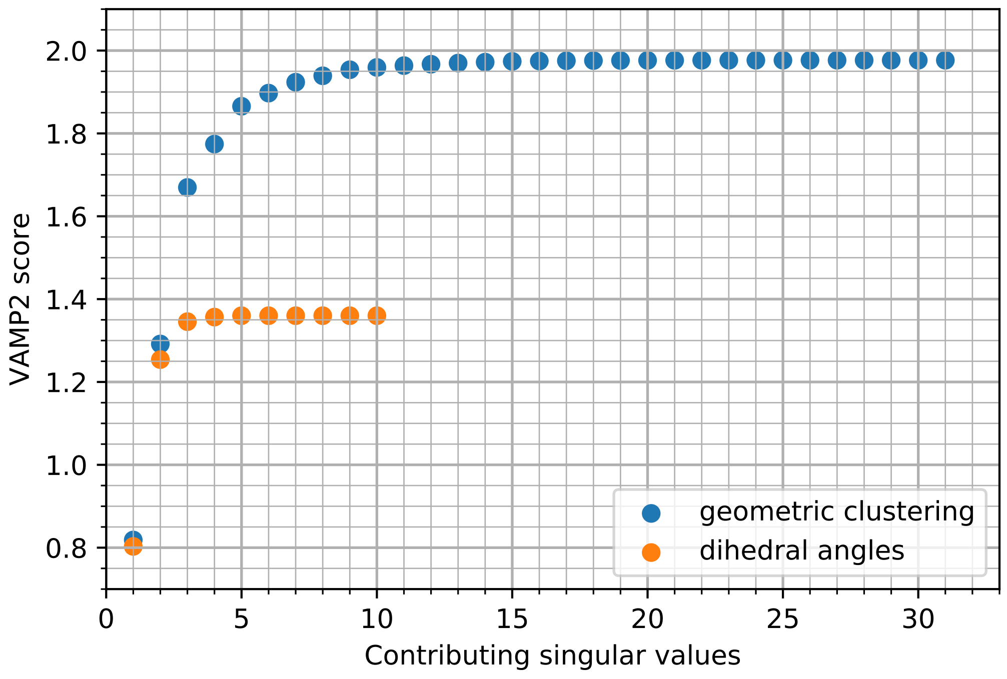


Figure 12: VAMP2 score of fuzzy-like Ramachandran clustering and dihedrals based on how many singular values are used.

5. Applying VAMPnets to AIB$_9$

Even though the claim of VAMPnets is that the clustering results rely less "on substantial technical expertise of the modeler" [6], VAMPnets still require the modeler to select various parameters. In the following, the meaning of the different parameters is explained together with their choice in this work. At the time of this work, there exists no user friendly "plug and play" implementation of VAMPnets. The user has to build VAMPnets manually with Keras [22]. This is why many parameters like the optimization method must be specified explicitly by the user.

5.1. Parameters of VAMPnets

Lag time τ: The user has to select a lag time that is used by VAMP when calculating the Koopman operator. τ is therefore the time lag between the points in both network lobes. As was shown in figure 10, $\tau = 1.2$ ns is a good choice for VAMP on AIB$_9$. We assume in the following that this lag time works on VAMPnets as well.

Maximum number of states m: The user has to choose the size m of the output layer. This means the user has to specify what the maximum number of states found in the data should be. On one hand we already saw based on the Ramachandran clustering (see 3.2.2) that we expect a maximum of 32 states in AIB$_9$. On the other hand, we saw based on the analysis with VAMP (see 4.3) that VAMPnets are expected to resolve only up to 10 states. For the first analysis we will set $m = 20$ since we think that this is sufficient. Later we will do clusterings with $m = 32$ as well.

Cutoff ϵ: When calculating the VAMP2 score, ϵ is a cutoff that can be used to reduce computational time. All singular values smaller than ϵ will be ignored. This can come in handy, when m becomes large and computational time increases with no additional information obtained from small singular values. In the case of AIB$_9$ this is irrelevant because we never want more than 32 states and always want to use all singular values. ϵ was set to 10^{-5}. Therefore no significant error was made due to ϵ.

Batch size: Due to a limited memory capacity and runtime optimisation, computers can not process the whole trajectory at once to calculate gradients of χ. Therefore, the data is split up to batches and a stochastic gradient descent (SGD) is performed. The network calculates a gradient for one batch and slightly modifies its parameters. Afterwards it does the same thing with the next batch until all batches are done. Undersized batches can result in numerical errors while training VAMPnets. Oversized batches make the algorithm more stable at the cost of longer computation time. Additionally a high batch size results in less gradient steps, which means that the network does improve its parameters less often when iterating over the training data.
In this thesis, a batch size of 3000 was used for all networks since this appeared to be a good compromise.

Number of epochs: The number of epochs represents the number of iterations over the whole training data set that is used to train the network. In this work, the networks were trained for 30 epochs, which have proven to be sufficient since the networks showed converging results.

Optimization method: The user has to select the optimization method that should be used for the SGD. In this work we used ADAM [29] since it was recommended for VAMPnets [30].

Network architecture: How many hidden layers should the network have? How many nodes should be in each layer? The answer to those two questions is totally up to the user. Nonetheless it is a very important question to answer since it massively changes the complexity of the network. The VAMPnets paper [6] proposes to use cone-shaped networks, where the number of nodes monotonically increases/ decreases by one for each hidden layer until the number of nodes in the output layer is reached. In the following we will also inspect the influence of the network architecture on the results of VAMPnets.

Data preparation: To detect overfitting, we want to split our data into a training and a validation data set. To ensure that both data sets contain trajectory points from all trajectories, all possible pairs of points and lagged points were generated for each trajectory. Afterwards, 90% of pairs of each trajectory were assigned as training data and 10% as validation data. This validation data pairs were taken in equal distances from the original trajectory so that the validation points were not concentrated at one location of the trajectory. All networks in this work were trained with the exactly same data splitting. This is important because differences in the clusterings can not be explained by different training data sets.

5.2. Visualisation

VAMPnets do not provide their own coordinates to visualize the clustering. Given that the assumptions of VAMPnets are valid, the VAMP2 score can be used to measure how successful a clustering was. Still, we want to see visually how well VAMPnets clusterings match with our understanding of a reasonable state clustering for AIB_9.
Since we already did a Ramachandran clustering in section 3.2.2 and showed with the Ramachandran plots that it is well justified and precise for AIB_9, we can assume to know what the ideal clustering should look like. Therefore we can visualize VAMPnets clustering by plotting the differences in state assignments between VAMPnets and the Ramachandran clustering. We will call this a state mapping plot. An example is shown below in figure 13a. Figure 13 shows the best hard clustering result out of 10 independent trainings achieved by a cone-shaped network with 4 hidden layers. The maximum number of states was set to 20, but only 16 were populated. On the upper half of the circle in figure 13a the states found with VAMPnets are listed. On the lower half are the states found with the Ramachandran clustering are shown. Those are indexed clockwise with decreasing population. The translation from index to molecular representation is shown in table 2. The connections between Ramachandran and VAMP states represent the fraction of points that map to an other state. The small number below the state labels in figure 13a indicate how many trajectory points belong to the state. This is also represented by the radian measure of the state.

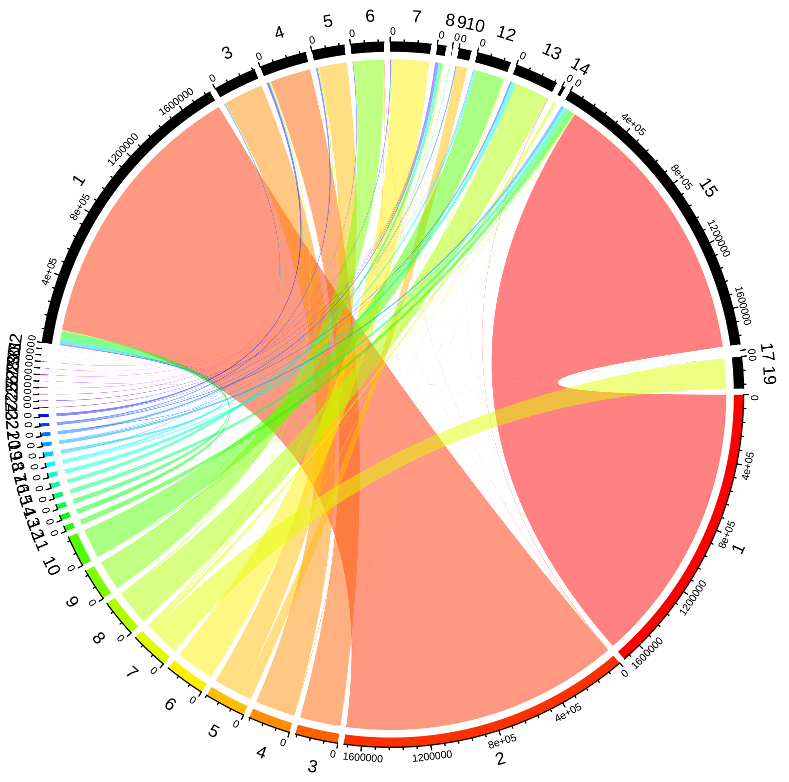
In case we want to analyze a new system for which no reference clustering is known yet, we can plot the trajectory points in the first 2 components of VAMP obtained from the original trajectory. An example can be seen in figure 13b. The clustering itself is visualised by color.

The advantage of the representation in figure 13a is that it clearly shows if VAMP merges or splits up states from the Ramachandran clustering. Since the Ramachandran states have a meaningful molecular interpretation, we can directly see from table 1 which molecular conformations VAMP struggles to distinguish. For example we see in figure 13a that the Ramachandran state 5 (RRRLL) is split up to both VAMPnets state 10 and 5.
The advantage of the representation in figure 13b is that we can distinguish trajectory points that are clustered to the same state. We get information about where the states are in VAMP coordinates which allows us to see which states are close. For example we see that minor states close to the two main states in red (LLLLL) and blue (RRRRR) are clustered to the main state as well. Additionally we would be able to see if states were not localized in case the clustering had gone wrong and the points of a state were spread across the coordinate space.

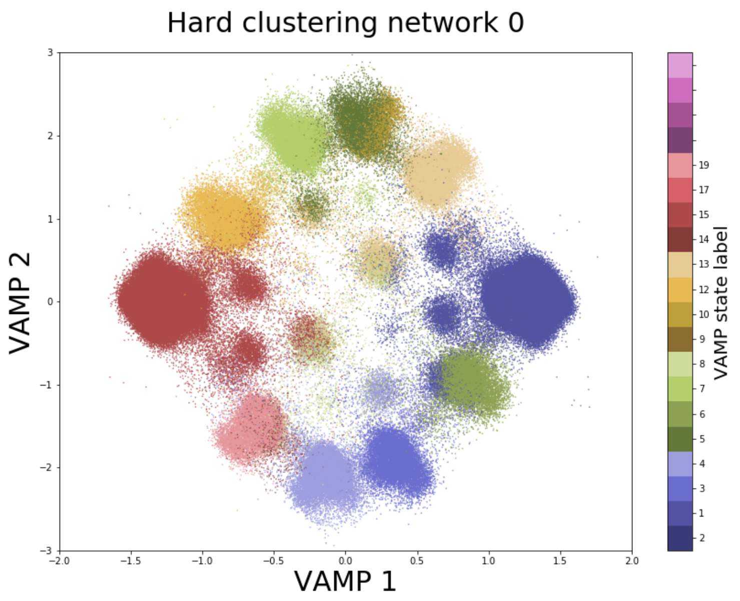
For the sake of clarity and because the essential information can mostly be obtained from the state mapping plot (see figure 13a, we will only show this one for further clusterings. The other representation can be found in the appendix. Table 2 shows the translation of the 10 most populated states in the Ramachandran clustering.

state index	interpretation	population	%
1	LLLLL	1716645	31.03
2	RRRRR	1701847	30.76
3	LLLRR	249536	4.51
4	LLRRR	246444	4.45
5	RRRLL	240263	4.34
6	RRLLL	235933	4.26
7	LLLLR	222508	4.02
8	RRRRL	219050	3.96
9	LRRRR	188590	3.41
10	RLLLL	182983	3.31

Table 2: Translation from state index to structural interpretation for the 10 most populated states of the Ramachandran clustering.



(a) Example of the state mapping plot



(b) Example of the visualization in the 2 main VAMP coordinates. The states are encoded in the color.

Figure 13: Best example of a Clustering done with cone-shaped VAMPnets with 4 hidden layers. The lag time was set to $\tau = 1.2$ ns and the maximum number of output states to 20.

5.3. Cone-shaped networks with varying depth

As stated in section 5.1, the original VAMPnets paper [6] proposed cone-shaped networks, where the number of nodes monotonically increases/ decreases by one for each hidden layer until the number of nodes in the output layer is reached. The freedom of architecture can therefore be reduced to the number of states the network should cluster to. However this is just one possible network architecture and it is not given that this is an intuitive or optimal network choice as the following example illustrates:

- In case we had 10 input coordinates and had tried to cluster on 7 states, we would have 17 hidden nodes distributed in 2 layers.
- In case we had 10 input coordinates and had tried to cluster on 2 states, we would have 42 hidden nodes distributed in 7 layers.

The second network is far more complex, even though it seems to performs a simpler clustering which should require a less complex network. Because of this consideration we construct our networks differently. While maintaining the cone shape, we allow to choose the number of hidden layers as well. The difference in the number of nodes between two consecutive layers is approximately constant. In our case, this means that for a network with 10 input nodes, 20 output nodes and 1 hidden layer, this layer has 15 nodes. If we were to choose 2 hidden layers, those layers would have 13 and 17 nodes and so on.
To check the influence of the number of hidden layers on the clustering results, a set of networks with 0 to 10 hidden layers was trained. Examples of the clusterings are shown in the figures 14 to 17. Since the input coordinates are 10 dimensional and we cluster to a maximum of 20 states, the network with 10 hidden layers has the architecture proposed by the VAMPnets paper.
In comparison to original VAMPnets where the network shape narrowes, the shape of our networks widens up. This is because we previously reduced the dimensionality of our data by working with dihedral angles. With the idea of reducing the number of steps in the workflow when analyzing MD data, VAMPnets are designed to work on high dimensional input coordinates. Still we have seen that 10 dimensions are enough to clearly encode 32 different states. Therefore it should be even easier for VAMPnets to find those states, since the input data contains less irrelevant information.
Since the networks try to cluster by optimizing the VAMP2 score, we will also compare the scores of the networks when evaluating the quality of the clusterings.

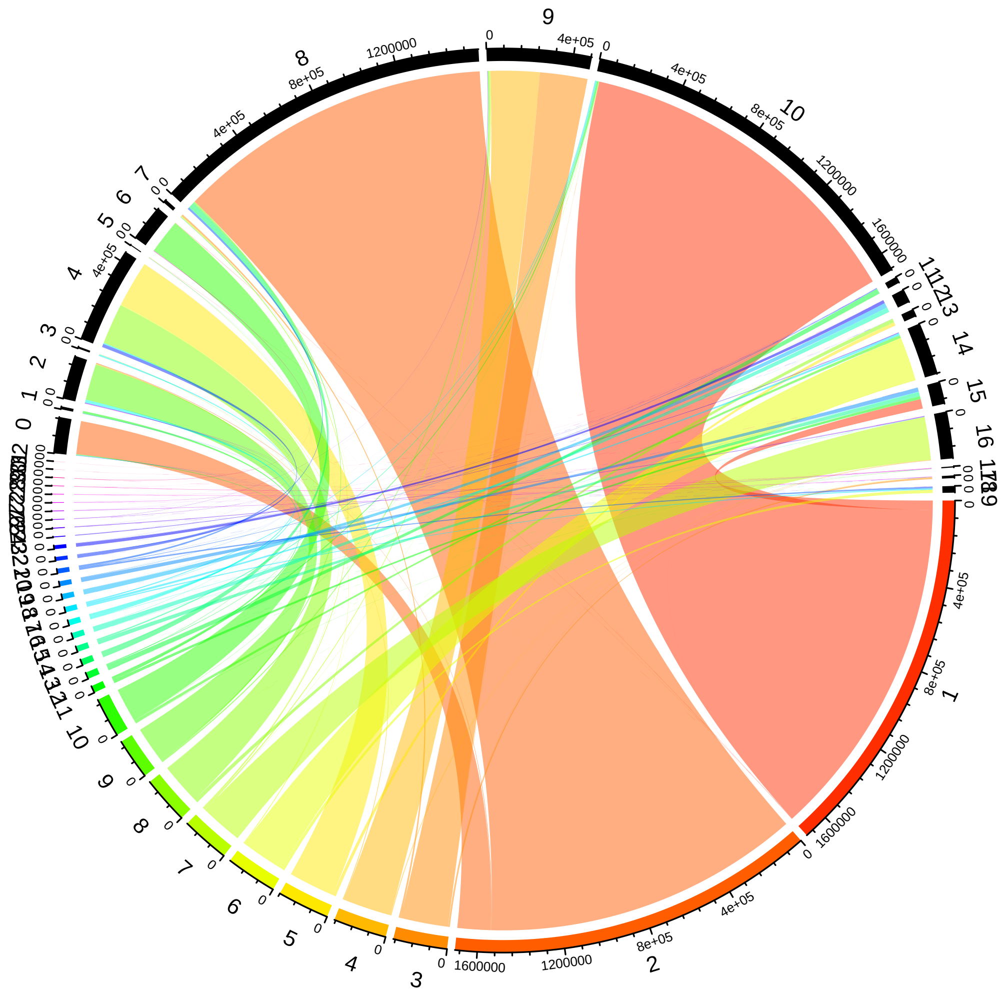


Figure 14: State mapping plot of a network without hidden layer. 6th best VAMP2 score of all 11 cone-shaped networks with different depths. For visualization in VAMP coordinates see figure 32.

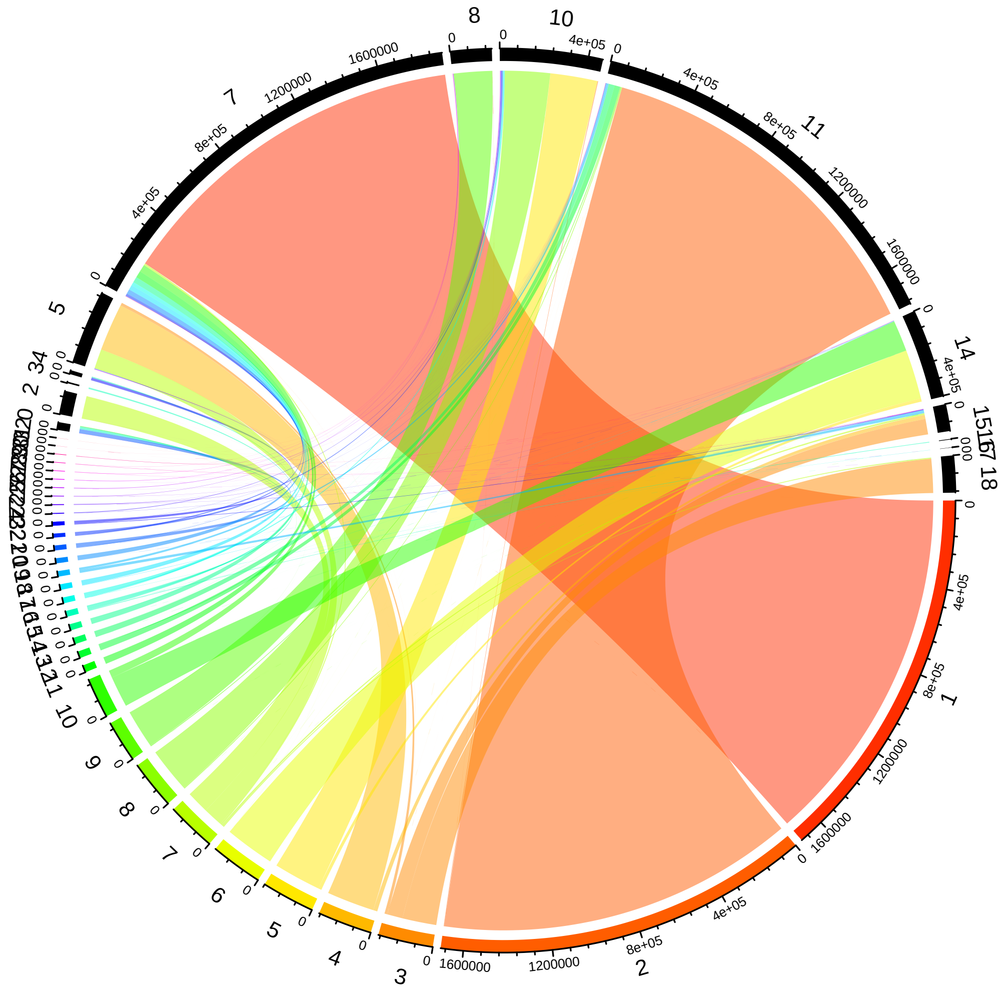


Figure 15: State mapping plot of a cone-shaped network with 4 hidden layers. Best VAMP2 score of all 11 cone-shaped networks with different depths. For visualization in VAMP coordinates see figure 33.

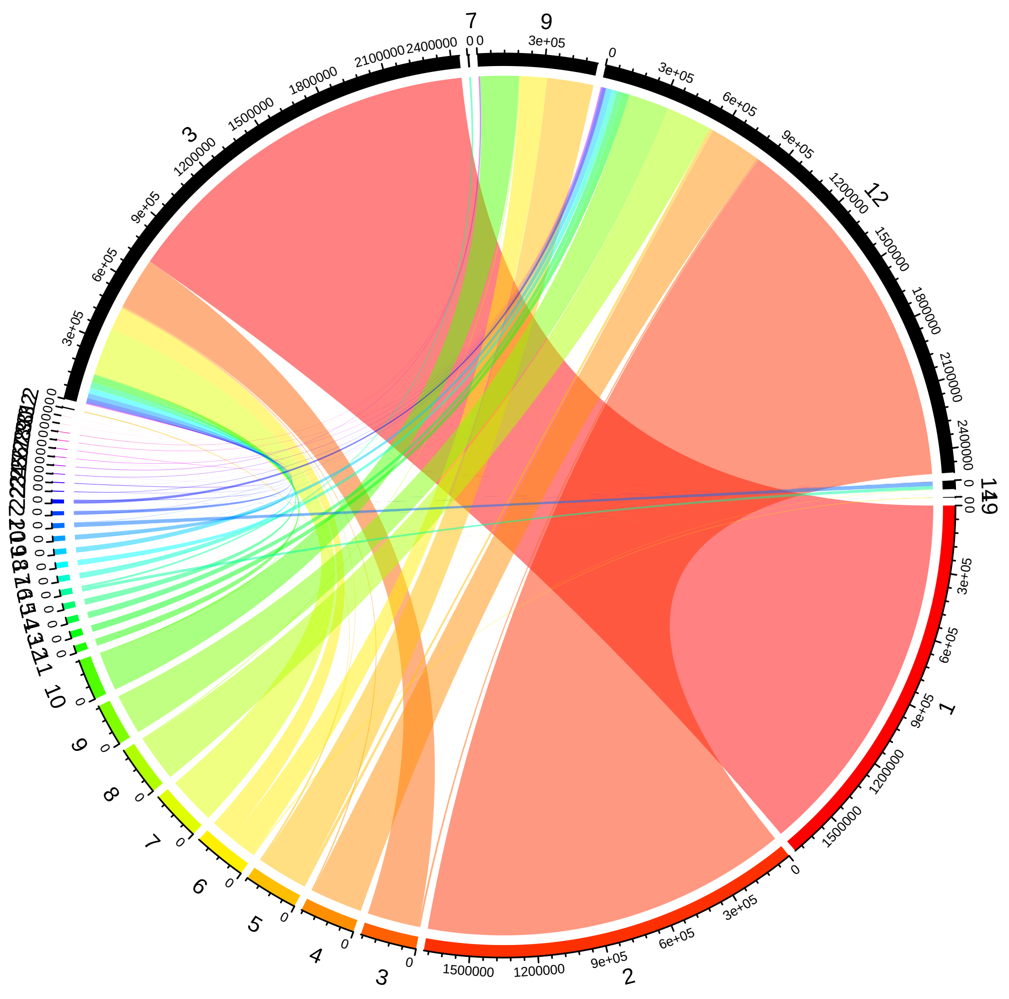


Figure 16: State mapping plot of a cone-shaped network with 5 hidden layers. Worst VAMP2 score of all 11 cone-shaped networks with different depths. For visualization in VAMP coordinates see figure 34.

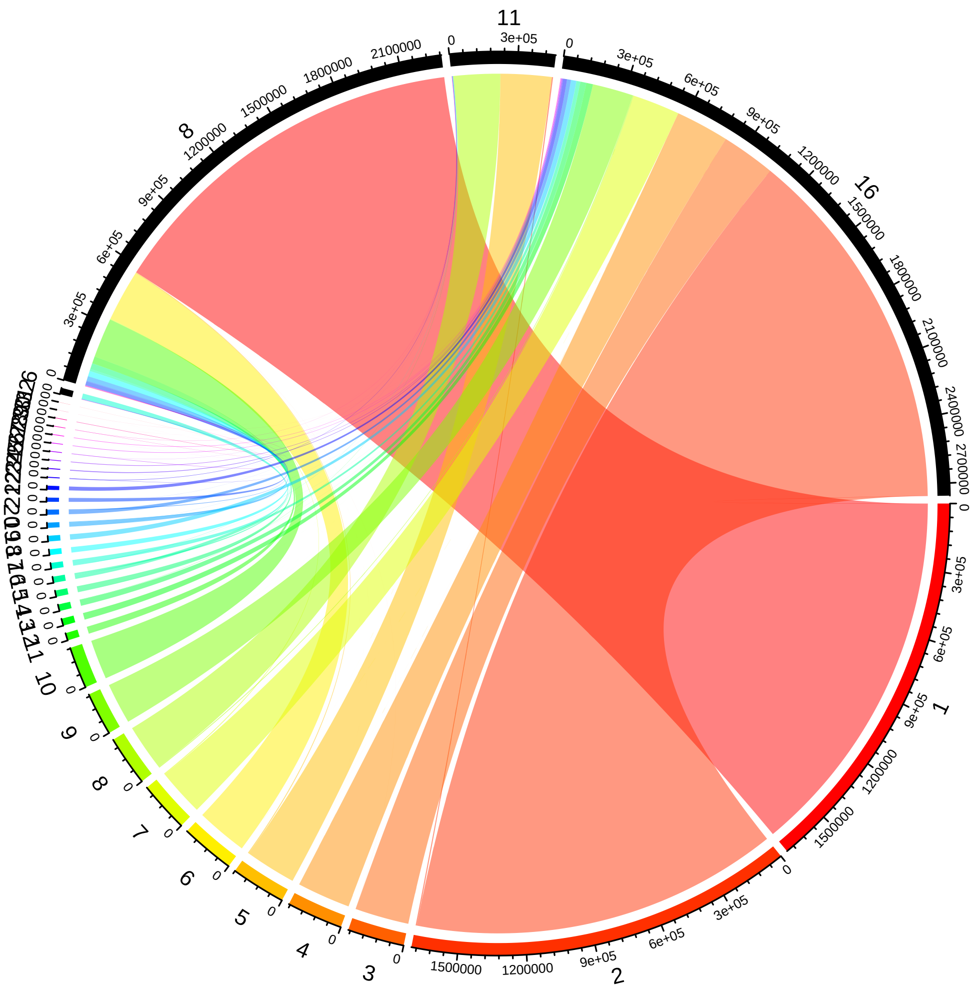


Figure 17: State mapping plot of a cone-shaped network with 9 hidden layers. 5th best VAMP2 score of all 11 cone-shaped networks with different depths. For visualization in VAMP coordinates see figure 35.

5.3.1. Description of the clusterings

The clustering without hidden layers in figure 14 is able to identify the two main states. However, the second Ramachandran state (RRRRR) is separated into state 8 and 0. Since state 8 and 0 do not contain any other major states of the Ramachandran clustering, this is not too disruptive. The Ramachandran clustering did only seperate between left and right handed residues, but the Ramachandran plots have shown that there are secondary minima in the free energy landscape of the residue at (-50, 50) and (50, -50) as can be seen in figure 8. So it could be that the VAMPnet identifies one or several of those minima as its own state.
More disturbing is that the fifth (RRRLL) and eighth (RRRRL) Ramachandran state are merged in state 4. Similarly, the third (LLLRR) and fourth (LLRRR) Ramachandran state are merged in state 9. Those states are close to each other as can be seen in figure 32 in the appendix.

The clustering with four hidden layers seen in figure 15 has the best VAMP2 score as can be seen in figure 18b. Therefore the VAMPnets claim that this clustering is the best one. However, it merges more states than the clustering without hidden layers. Similar to the clustering without hidden layers, the Ramachandran state 5 (RRRLL) and 8 (RRRRLL) are merged. This time only a small fraction of the Ramachandran state 3 (LLLRR) is merged with 4 (LLRRR). Instead, the Ramachandran state 3 is split up to 3 different states (5, 15, 18). Additionally the Ramachandran state 4 (LLRRR) is merged with a fraction of the Ramachandran state 7 (LLLLR) into state 5. Thus states are merged that are separated by more than the flip of a single residue. Moreover, the Ramachandran states 10 (RLLLL) and 6 (RRLLL) are merged to state 14.
This leads to the conclusion that this clustering is worse than the one without hidden layer even though it has the better VAMP2 score.

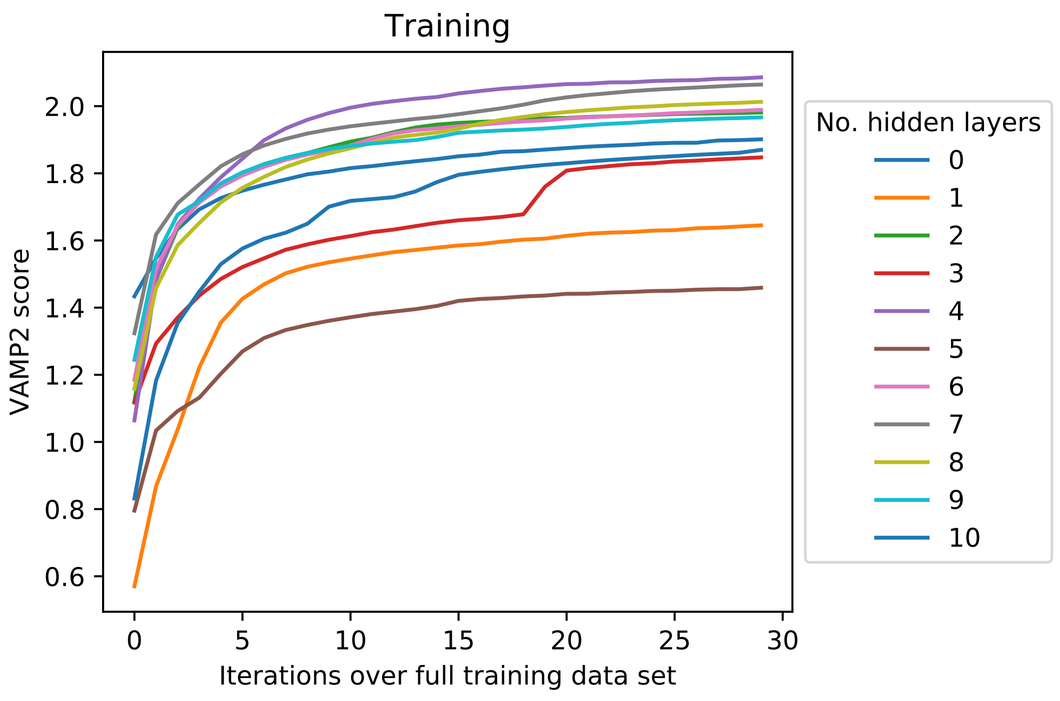
The clustering with 5 hidden layers seen in figure 16 has the worst VAMP2 score and yields corresponding results. The VAMPnet is only able to detect 6 different states even though it had the possibility to detect up to 20. A lot of states, including the main states are merged. VAMPnets state 3 contains LLLLL, LLLLR, LLLRR, a fraction of RRLLL and minor states. VAMPnets state 12 contains RRRRR, RRRRL, LLRRR, LRRRR and minor states. Only one more significant state was found. This state 9 contains RLLLL, RRRLL and the rest of RRLLL. The 3 other states only contain minor Ramachandran states. This clustering was only able to distinguish whether a state is rather right handed or left handed. Luckily the VAMP2 score tells us already that this clustering didn't turn out so well.

A further increase of hidden layers to 9 (see figure 17 does not improve or even deteriorates the clusterings. The network with 9 hidden layers only detects 4 states, whereby one state only contains a minor Ramachandran state. VAMPnets state 16 contains RRRRR, LRRRR, LLRRR, LLLRR, LLLLR and minor states. Those are all major states that have an R at the right end. VAMPnets state 8 contains LLLLL, RRLLL, RLLLL and minor states. The last state 11 contains RRRRL and RRRLL.
It can be said that here even a separation between mostly left handed and mostly right

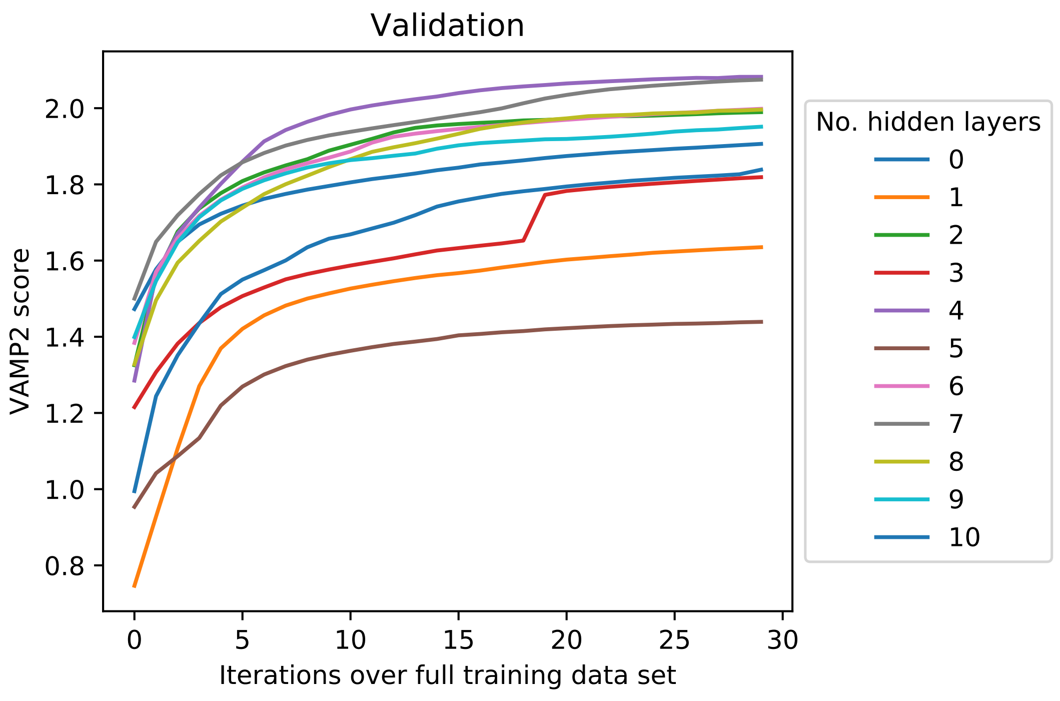
handed fails. Still, the VAMP2 score of this clustering is better than the one of the clustering without hidden layers. This can nicely be seen in figure 35 in the appendix as well.

5.3.2. Conclusion for cone-shaped networks with different depths

In conclusion, the networks with few hidden layers merge less states and produce a more detailed clustering than the networks with many hidden layers. The deeper the network, the worse the clustering becomes. This is counterintuitive since more complex networks perform simpler clusterings. Moreover, the universal approximation theorem is not valid for networks without hidden layer. For this network, it is mathematically not guaranteed that it is complex enough to produce the desired clustering. Nevertheless, our results show that this clustering is the best one, VAMPnets produced in this set of clusterings. One explanation for our observation could be that more complex systems overfit the problem. This is a common problem in machine learning. Complex networks are sensitive to noise in the training data and start interpreting meaning into it. To validate this explanation, we can look at the VAMP2 scores of the training and validation data set. During the training, we tracked the progression of the score on both data sets. The VAMP2 score was calculated after each epoch and is plotted for the training data in figure 18a and for the validation data in figure 18b. If the network was overfitting, one would expect the VAMP2 score of the validation data to decrease with increasing complexity which in our case means network depth. The VAMP2 score of the training data would still increase and become much larger than the validation score. This can not be observed. Both scores are mostly similar as can be seen in figure 31 in the appendix as well where we plotted the differences between them. Since the curves of the VAMP2 score of the training data flatten out for late epochs, we can assume that the networks are well trained. In the curve for 3 hidden layers one can see that at 18 epochs, the score increases quickly in one epoch. This happens when the network was able to jump out of a local minimum in its parameter space while minimizing the cost function.
We were also able to verify our assumption from section 4.3 that VAMPnets have problems in separating more than 10 states.



(a) VAMP2 score of training data after each epoch of training



(b) VAMP2 score of validation data after each epoch of training

Figure 18: VAMP2 scores during training. Figure 31 in the appendix shows the differences between training and validation score

Figure 19 shows the final VAMP2 score of the fully trained networks in relation to the network depth. As can be seen there is no real correlation between VAMP2 score and network depth visible. This is astounding since we could observe that the clusterings became worse for deeper networks. During the training, the network tries to optimize the VAMP2 score, which obviously did improve the clustering. Otherwise the networks would assign a random state to each trajectory point. This means that there is a correlation between VAMP2 score and the quality of the clustering. However the VAMP2 score does not directly measure the quality of the clustering even though this is one assumption of VAMPnets.
Figure 19 suggests that the VAMP2 score of a fully trained network underlies a high statistical variance. Otherwise it would be hard to explain, why the VAMP2 score of a network with depth 4 is the best and the one with a depth of 5 is the worst one.

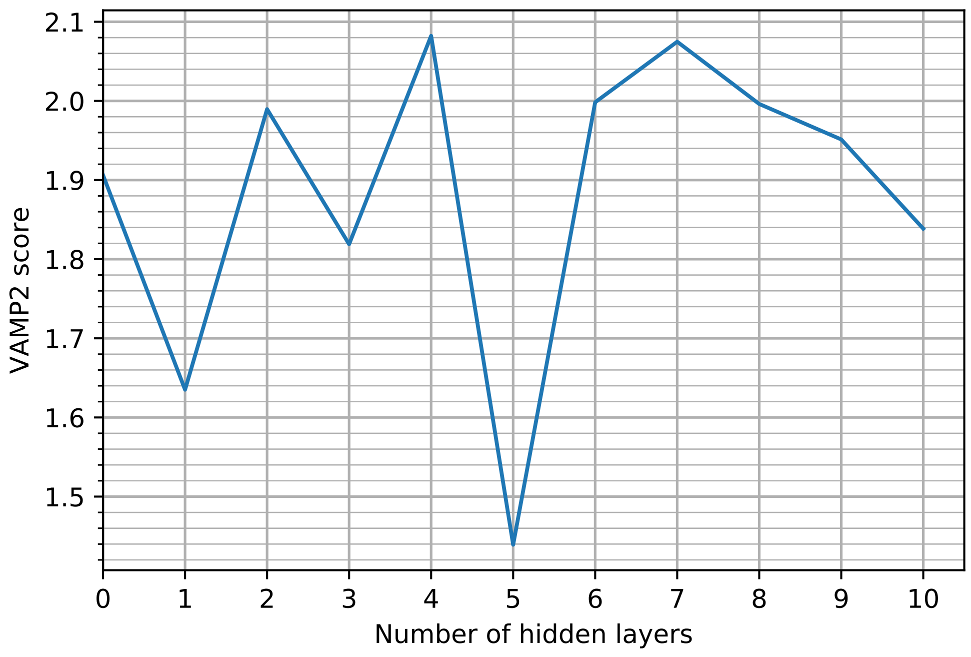


Figure 19: VAMP2 score in relation to the network depth

5.4. Fuzzy clusterings of cone-shaped networks

When comparing the clustering results to the Ramachandran clustering or plotting them as color code in VAMP coordinates, we always look at the hard clustering. After hard clustering, all information about how certain the network assigns a state to a trajectory point is lost. We now want to have a closer look at the fuzzy clusterings to see whether worse clusterings also come with a higher uncertainty in state assignments. Figure 20 shows the histograms of the highest assigned probability for all the fuzzy clusterings in the trajectory for different network depths. This means that they show how many state labels were assigned with which certainty. These 6 histograms illustrate the variation when going to deeper networks.

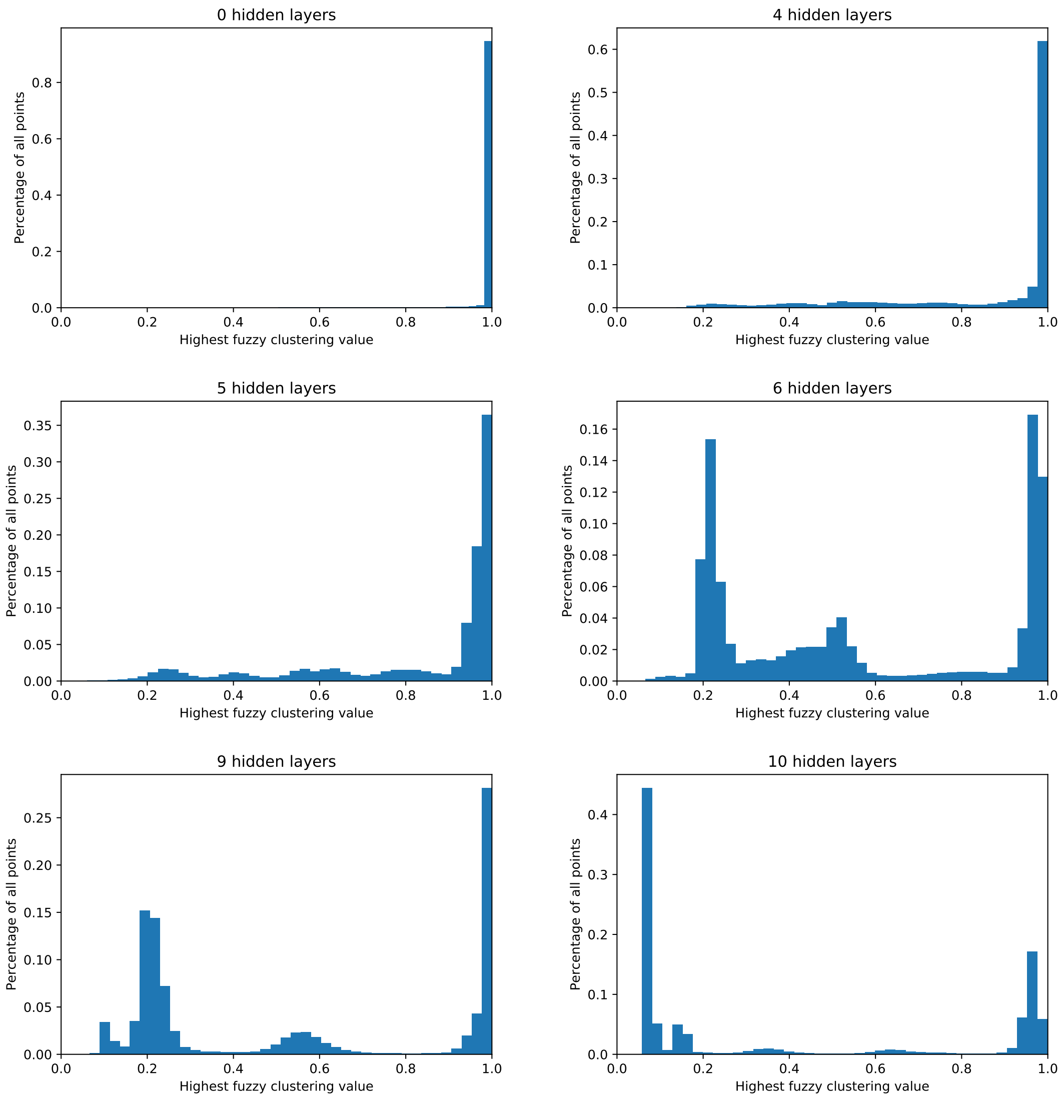
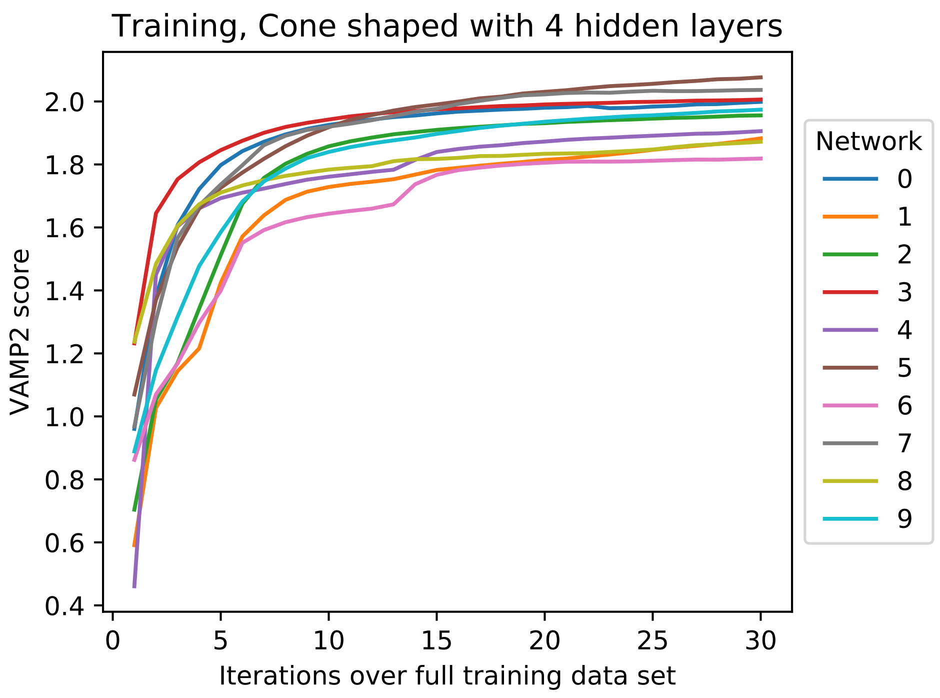


Figure 20: Histograms of the highest probabilities of the fuzzy clusterings for cone-shaped networks of different depths

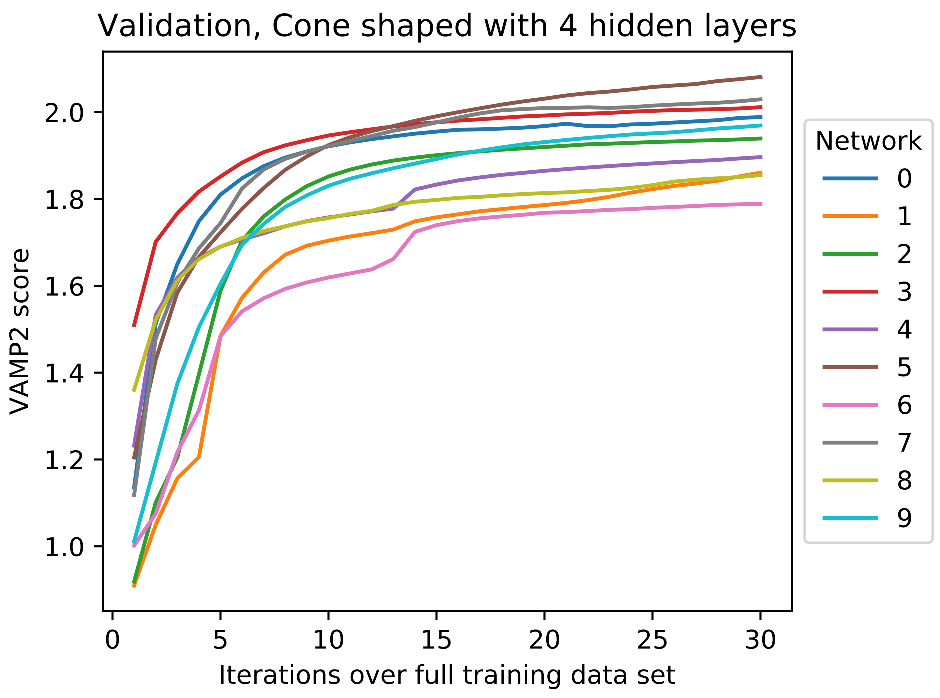
It can be seen that the certainty of predictions decreases with increasing network depth. This aligns well with the obeservation that the clustering becomes worse. The network without hidden layer assigns almost all state labels with a probability higher than 98%. In contrast, the deepest network with 10 hidden layers assigns almost half of the state labels a probability of 10% that the state label is correct. This means that the network is unable to unambiguously select a state for the majority of trajectory points. The clustering with 4 hidden layers, which had the best VAMP2 score finds state labels with a high degree of certainty. However, the worst clustering with 5 hidden layers also labels the majority of points with a certainty larger than 90%. Therefore we can not say that a bad clustering implies a low degree of certainty in the state assignments. For this set of networks, at 6 hidden layers, a sharp peak at 0.2 appears. The deeper the networks become, the more points are found in this peak instead of the one on the very right side. It is notable that there are certain values that are more likely to appear than others. The peaks often have a constant distance from each other. This means that the network is more likely to assign a multitude of a specific value in the fuzzy clustering. The reason behind this is not clear.

5.5. Reproducibility of clustering results for cone-shaped networks

The training of a neural network uses stochastic gradient descent. Furthermore, the strengths of connections are initialized randomly when the network is created. To see how those statistical fluctuations influence the results of the clustering, we train multiple networks with the same architecture and same choice of hyperparameters on the same data set and compare their results. The networks were identical to the networks with 4 hidden layers from section 5.3. This network depth was chosen because it generated the best VAMP2 score out of all network depths Figure 21 shows how the VAMP2 score of the training and validation data changes during training. All of the curves seem to converge, which is why we can assume that we trained long enough. We can already see that the VAMP2 scores vary by 0.3 in on both data sets. This is half of the variation observed in figure 18a when comparing networks with different depths. Looking back at figure 12 it is also half of the difference between the VAMP2 score of the Ramachandran clustering and the VAMP2 score of the dihedral angles which did not represent a clustering at all.
The examples in figure 22 - 24 show how different the clustering can be when comparing them to the Ramachandran clustering.



(a) VAMP2 score of training data after each epoch of training



(b) VAMP2 score of validation data after each epoch of training

Figure 21: VAMP2 scores during training. Difference of validation and training score can be found in figure 36 in the appendix

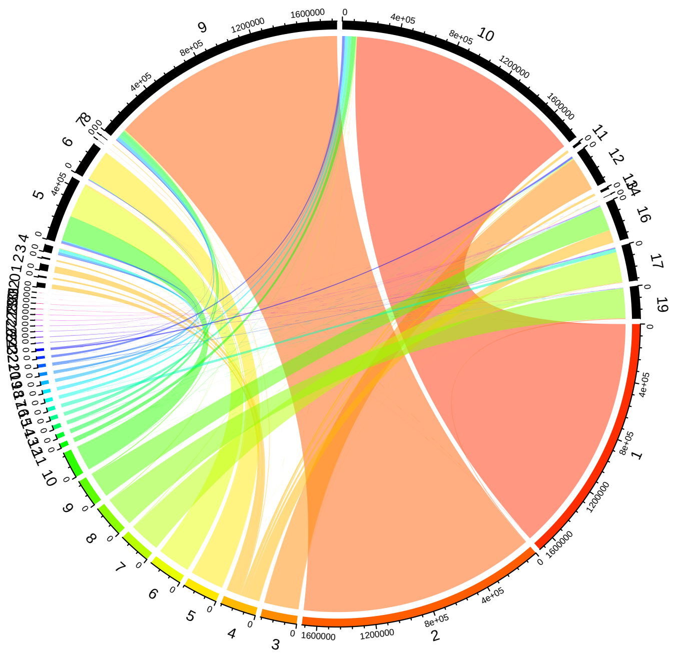


Figure 22: Network 5, best VAMP2 score of all 10 identical cone-shaped networks

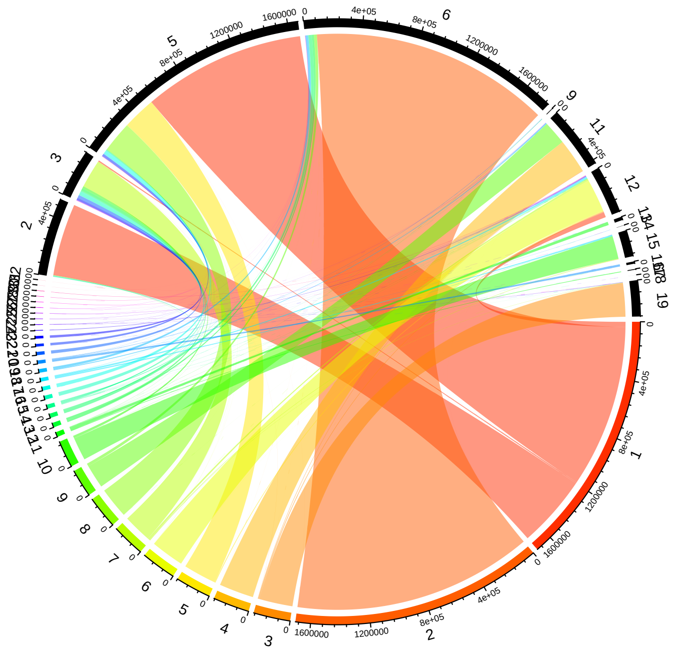


Figure 23: Network 7, 2nd best VAMP2 score of all 10 identical cone-shaped networks

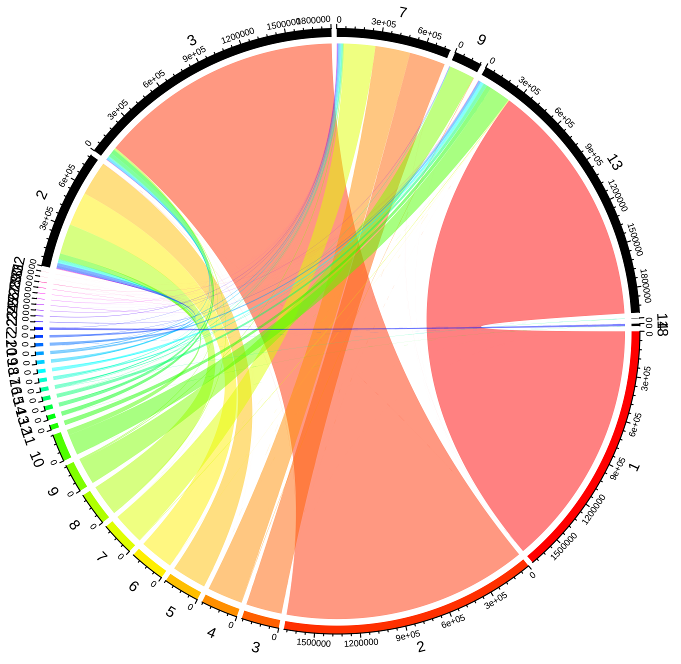


Figure 24: Network 6, worst VAMP2 score of all 10 identical cone-shaped networks

The clustering with the best VAMP2 score (Network 5) is able to separates the main states and most major states. It is shown in figure 22. The Ramachandran state 4, which represents LLRRR as can be seen in table 2 gets split up in many different states on the top left side of the plot (state 0 - 3). Furthermore VAMPnets assign a part of LLRRR to state 16 where the Ramachandran state 9 (LRRRR) can be found as well. Low populated states are mostly assigned to a main but also to major states. This is not because we did only cluster on a maximum of 20 states. The contribution of low populated states to the VAMP2 score is negligible as was shown in figure 12. We can see that 4 of the 20 states VAMPnets have available are used only for LLRRR even though the total number of states that can be found in AIB$_9$ is higher than 20. Furthermore only for 18 of the 20 possible states states were assigned.
Figure 23 shows the clustering with the second best VAMP score (Network 7). Here the Ramachandran state 1 (LLLLL) is split up to 2 states. Unfortunately in one of those states (state 5) two other major states are included. Those are the Ramachandran states 5 (RRRLL) and 8 (RRRRL) which are quite different to LLLLL. This can nicely be seen in figure 23 in the appendix. The problem here is that the VAMP2 score suggests that the clustering was successful even though we would say that being able to identify the main states LLLLL and RRRRR is the most fundamental requirement of a good clustering.
The example used in figure 13a when illustrating different visualization methods comes from this set of networks. It shows the clustering found by network 0 which had the 4th best VAMP2 score of all 10 networks. Given that it represented a clustering that was able to identify both main and all major states correctly, it is a matter of concern that

its VAMP2 score is smaller than the one of network 7 in figure 23 which was unable to detect the main states correctly. We would expect the clustering of network 0 to have the best VAMP score since it reproduced the Ramachandran clustering the best for which we have shown that it represents a more or less ideal clustering. Still, this is not what the VAMP2 score measures.
Figure 24 shows the result with the worst VAMP2 score. Here the main states are separated but the Ramachandran main state 1 (LLLLL) is merged with the Ramachandran major state 10 (RLLLLL). Furthermore only 3 more relevant states are found. State 2 contains the Ramachandran states 5 (RRRLL), 6 (RRLLL), 8 (LRRRR) and minor states. State 7 contains Ramachandran states 3 (LLLRR), 4 (LLRRR), 7 (LLLLR) and minor states. VAMPnets correcly identified the Ramachandran state 10 (RLLLL) and merged it with no other states.

Summing up, we could observe that when applying VAMPnets as a clustering algorithm, they do not return reproducible clusterings. The quality of the clusterings did vary from identifiying all main and major states correctly to not being able to find the two main states. While bad VAMP2 scores below 1.8 hint that many states are merged, a high VAMP2 score is not a sufficient criteria for a good clustering as could be seen for network 7. Meanwhile a better VAMP2 score is not a necessary criteria for a better clustering as network 0 showed in comparison to network 7.

5.6. Rectangular shaped network architectures

As seen in section 5.3, the architecture of a network can radically change the results it delivers. All networks that were inspected until now had a cone shape. This means that their architecture was narrow (few nodes per layer) and deep (many hidden layers). We now want to see whether short networks with many nodes per layer behave differently. Therefore we choose an examplary network architecture with 4 hidden layers and 300 nodes per layer. For the sake of consistency we also cluster to a maximum of 20 states. All other parameters were chosen as described in section 5.1.
Figure 27 shows the train and validation scores during training. Figure 26 shows the comparison of the best and the worst clustering based on the VAMP2 score to the Ramachandran clustering. Alike the clusterings done with cone-shaped networks, the clustering with the best VAMP2 score was not the one that reproduced the Ramachandran clustering best. Network 5 did only have the 5th best VAMP2 score out of all 8 networks but was able to separate both main and all major states best.
When looking at the progression of the VAMP2 score during training, it is very astounding that it decreases for some networks with increasing epochs. Usually we would expect that the score increases monotonic with the epochs as was for the cone-shaped networks shown in figure 21a. To except an error of the user, we double checked that the network in fact optimized and measured both times the VAMP2 score of the training and validation data. Even though stochastic gradient descent can temporarily worsen the results during training on single batches, stochastic optimization algorithms are usually designed to keep their last best result after one iteration of a training process if they are unable to improve. However the optimization algorithm is included in the prebuild tensorflow software package, which is widely used and well optimized. Therefore we

expect the tensorflow software to perform well and lack an explanation for the strange behaviour measured during training. However the final networks seem to be well trained and the clusterings look very similar to the ones done with cone-shaped networks. Even though they appear to be more stable, they share the same problems. Still, when training on identical data, they produce varying clusterings. The main problem remains that the VAMP2 score can not be used to identify the best clustering. The conclusion that rectangular shaped VAMPnets produce more stable results is on one hand based on the lower variation of the VAMP2 score which fluctuates only in a range of ≈ 0.13 for the trained networks. On the other hand even the worst clusterings did only merge few major states together. We also trained a second set of 10 rectangular shaped VAMPnets that tried to cluster to a maximum of 32 states. The results were similar to the ones shown in figure 27 to 25. In the set with 32 states were also no networks that clustered the majority of states together like the cone-shaped network in figure 24. This supports the conclusion that rectangular VAMPnets are more stable.

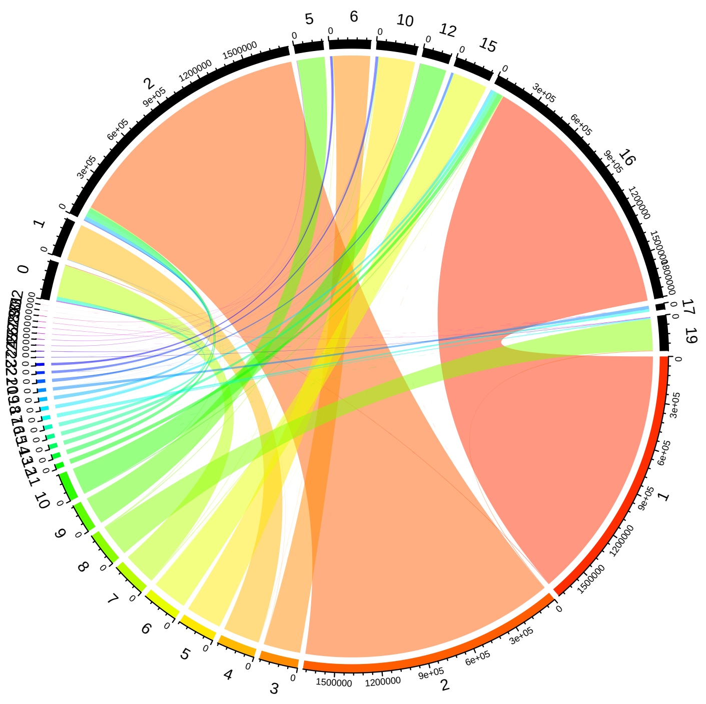
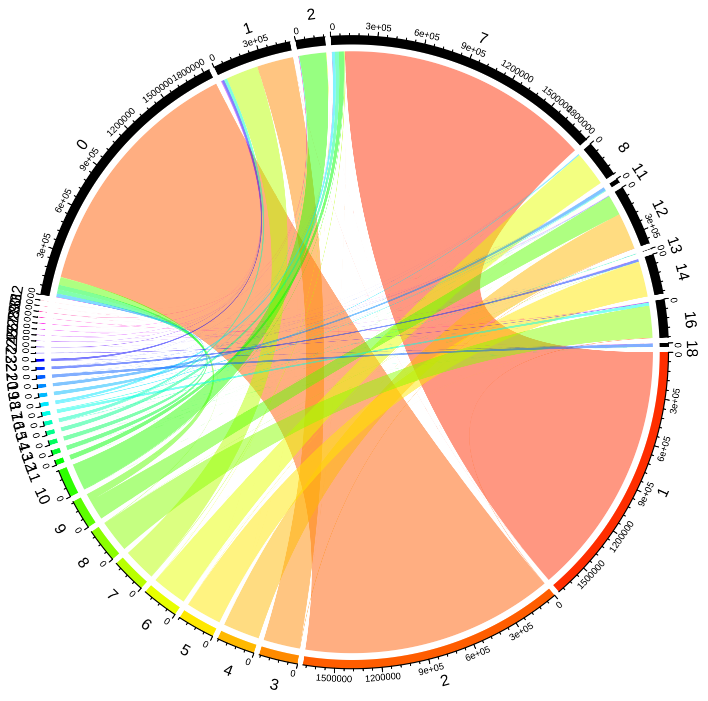
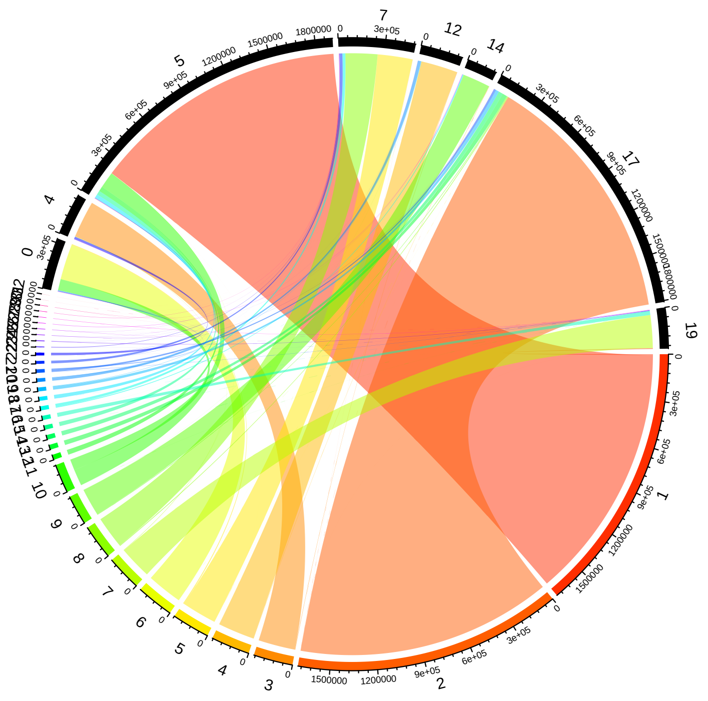


Figure 25: Network 5, 5th best VAMP2 score of 8 identical rectangular shaped networks.

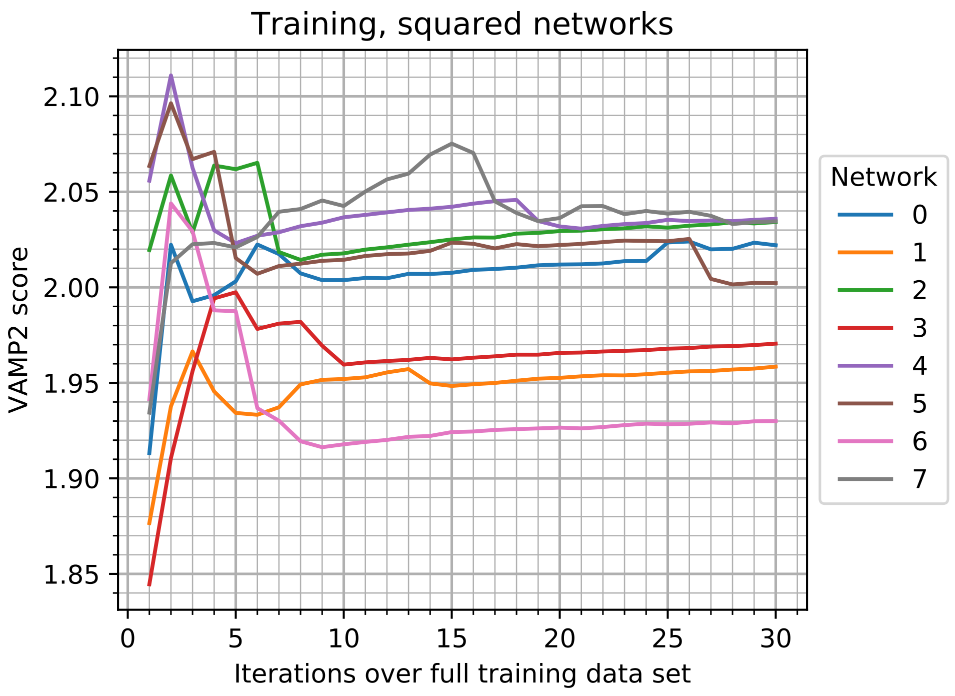


(a) Network 7, best VAMP score

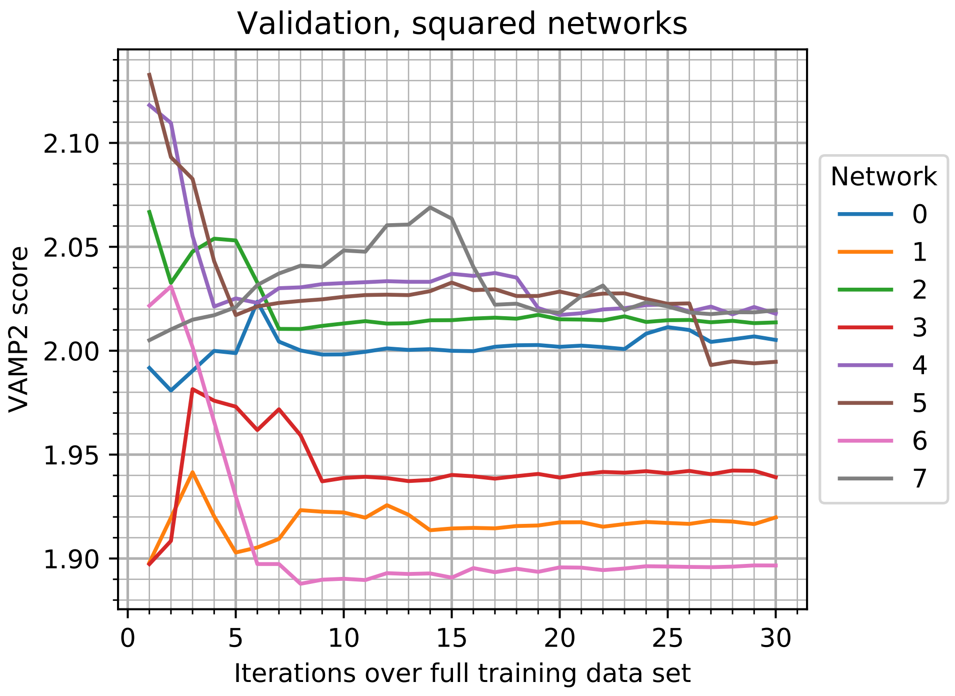


(b) Network 6, worst VAMP2 score

Figure 26: Best and worst examples of clustering results of 8 identical rectangular shaped networks based on VAMP2 score.



(a) VAMP2 score of training data after each epoch of training



(b) VAMP2 score of validation data after each epoch of training

Figure 27: VAMP2 of rectangular shaped networks scores during training. Difference of validation and training score can be found in figure 37 in the appendix

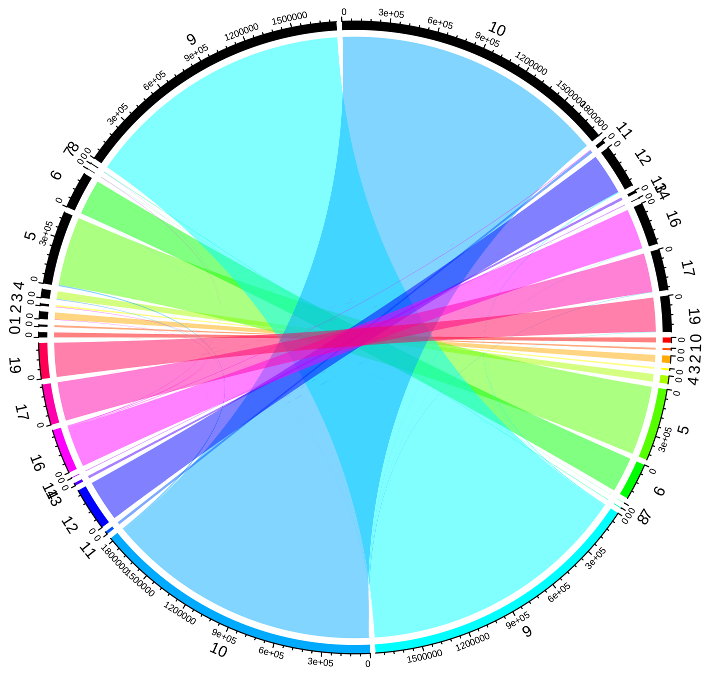
5.7. Different interpretations of the fuzzy clustering

As hinted in section 2.8.2, the hard clustering can be derived in different ways from the fuzzy clustering:

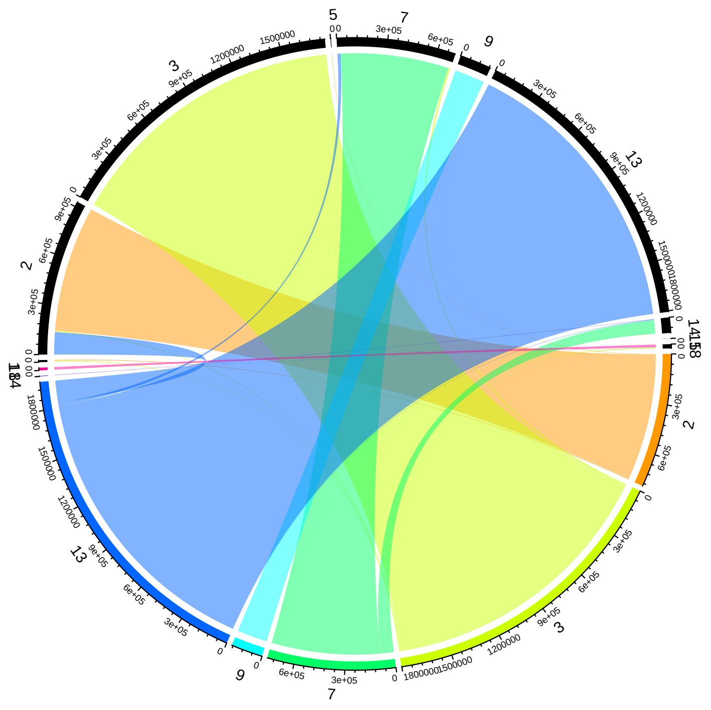
1. The intuitive way is to accept the state with the highest probability in the fuzzy clustering as the state in the hard clustering.

2. The tutorials included in the official VAMPnets repository [30] calculated a meanfree fuzzy clustering from the original one. In each component is a likelihood that represented the meanfree probabilities of the original fuzzy clustering. Afterwards, the hard clustering was done by classifying each point with the state that had the highest likelihood in the meanfree fuzzy clustering. Probably this is done to reduce the number points in minor states that are merged into a main states. However, calculating meanfree probabilities undermines the concept of probabilities being positive and normalized to one. Given that the activation function of the output layer was specifically constructed to give the output layer the appearence of a probability, this approach seems counter intuitive.

We now want to compare the two different hard clusterings that are build on the same fuzzy clusterings by comparing which states are labeled differently. Figure 28a shows an example for a cone-shaped network that clustered AIB$_9$ well and figure 28b one that was unable to find a good clustering. The top half of the state mapping plot represents the meanfree clustering while the lower half represents the original hard clustering.
As expected, in the meanfree clustering some of the points that previously belonged to the main state are now distributed among other states. However the changes are only minor and don't change the clustering fundamentally. In fact, for the best performing network in figure 28a, the differences are almost not visible. Here a minimal fraction of one one main state (10) in the original clustering is assigned to the meanfree state 5. For the worst network in figure 28b the differences are more significant. A fraction of a main state (13) in the original clustering is assigned to the major state 2. Additionally it can be seen that the meanfree clustering finds more states in this case. It divides state 7 so that a fraction of state 7 is now the new state 14. Still the clustering in figure 28b is only slightly finer and the new states are too low populated to represent major states. Taking into concern that the motivation of the meanfree clustering seems arbitrary, the changes in the clustering do not jusitify this approach as being better that the original hard clustering.
We could observe that the good clustering changed less than the bad one. This seems reasonable since states are lower populated when finding more states. This corresponds with lower probabilities. Therefore the sum over all probabilities for one states is lower as well and the probabilities in one trajectory point are changed less when calculating the meanfree version of the clustering.



(a) Network 5 (see figure 22), best VAMP score.



(b) Network 6 (see figure 24), worst VAMP2 score.

Figure 28: Differences between meanfree clustering (top half circle) and original hard clustering (bottom half circle) for cone-shaped networks

6. Summary

Finally we want to summarize the results of our studys on our model system AIB$_9$ with VAMP and VAMPnets. A more detailed deduction of our conclusions can be found at the end of the corresponding chapters in section 4 and 5.

6.1. Ramachandran clustering

The Ramachandran plots of the inner 5 residues have shown that most conformations of AIB$_9$ clearly have a right or left handed orientation at each residue. Therefore a clustering that separates states by the chiral orientations of the residues is physically meaningful and covers the system dynamics well. Such a combinatoric Ramachandran clustering was therefore used as a reference for the clusterings done with VAMPnets.

6.2. VAMP

We used VAMP with dihedral angles of the inner 5 residues as input coordinates to find the slowest processes and to do a dimensionality reduction afterwards. The dimensionality reduction done with VAMP produces similar results as Principal component analysis (PCA) on AIB$_9$. We saw a a strong correlation between the first 5 PCs and the first 5 VAMP coordinates. This was expected for AIB$_9$ since TICA finds the same correlation to PCA. VAMP is expected to behave similar since it also rates the importance of coordinates by their timescale. The similarity of coordinates with high variance to coordinates with high implied timescales is due to the choice of our test system and is not generally given as previous studies on alanine dipeptide have shown [11]. Similar to TICA, the choice of the lag time τ influences the collective variables found by VAMP. For large τ, the correlation between PCs and VAMP coordinates is reduced (see 4) while for small τ, Markovianity of the dynamics is no longer given (see 4.2).
The analysis of the singular values of the Koopman operator can give information about the relevant number of collective variables that should be kept for clustering algorithms. As seen in figure 11, only 5 dimensions are required to represent the slowest processes of AIB$_9$. This matches with experience from PCA, where 5 dimensions were deemed to be sufficient for AIB$_9$ as well [21].
In contrast to TICA, VAMP provides a Chapman Kolmogorov test to check if the observed dynamics are Markovian. We have seen in section 4.2 that for AIB$_9$, the approximation of the dynamics as a Markov process should only be made with $\tau > 1$ ns. The timescales of dynamics in the ϕ and ψ angle of each residue of AIB$_9$ differ by orders of magnitude. Lag times > 1 ns hide the fast dynamics in the ψ angle, so that only the changes of chiral orientation implied by transitions in the ϕ angle are resolved.
VAMP can also be used to build an Markov state model. When applying VAMP on vectorized state trajectories, the resulting Koopman operator is equivalent to the transition matrix of a Markov state model.
The analysis of the Ramachandran clustering with VAMP has shown that the VAMP2 score (see equation (33)) is mainly influenced by the 9 largest eigenvalues of the Koopman operator. This leads to the conclusion that it is possible to find a clustering to only

10 states that has a similar VAMP2 score as the Ramachandran clustering with 32 well defined states.

6.3. VAMPnets

Clusterings with $\tau = 1.2$ ns and various different architectures of VAMPnets have shown that it is in principle possible to detect the 10 most populated states of AIB_9 (see figure 13). The lag time τ was chosen based on the previous study with VAMP. Those states also contain the main dynamics. Low populated states were rarely detected. The conclusion that VAMPnets struggle to find more than 10 meaningful states could be confirmed in all clusterings. Even though our Ramachandran clustering showed that there are 32 well defined states in AIB_9 and we gave VAMPnets the possibility to find up to 20 states, most of the time less states were found. In most clusterings VAMPnets merged highly populated states that differed only by the chiral orientation at one residue. When looking at VAMPnets as a ML algorithm that tries to find patterns, it is reasonable that states that only differ in one detail can be merged by accident.
Even though VAMPnets reduce the number of different steps for the analysis of MD data, the claim that it reduces the influence of the modelers expertise can not be confirmed. In the original workflow via PCA and density based clustering, the user had to select the number of collective variables d, a cluster radius r and the minimal population p_{min}. All of those parameters have a clear physical meaning and knowledge of the system helps for choosing suitable values. VAMPnets require many more parameters as described in section 5.1. While some of them, like the singular value cutoff ϵ, do not influence the clustering results if not chosen too poorly, others like the network architecture have a critical influence.
In the paper of Noe et al. [6], VAMPnets are designed to transform a high dimensional trajectory to a low dimensional state clustering. Therefore their cone-shaped architecture narrows with increasing network depth. In our case, the choice of dihedral angles as input coordinates previously reduced the dimensionality of the data. Since we want to find more states than we have input dimensions, our networks have to widen up with increasing depth. This change is not expected to influence the functionality of VAMPnets badly since it only reduces the amount of unimportant information provided to the networks. It is rather expected that VAMPnets can more easily detect the relevant states when there is less noise in the data.
We have seen that the choice of network architecture has a huge influence on the clustering results. The adaption of the original architecture of deep cone-shaped networks proposed in the VAMPnets paper [6] produced clearly worse results than networks with less hidden layers.
The choice of suitable parameter is compounded by the fact that many of them like the network architecture lack a clear physical interpretation. While having a general intuition that more complex systems should require more complex networks, we do not have an intuition how to achieve this complexity best. The multitude of possibilities how many hidden layers should be combined with which number of nodes in each layer leaves many combinations that appear to be equally complex.
Our analysis has shown that deep cone-shaped networks are often unable to separate the

10 major states of the system correctly (see 5.3). Simpler networks represented by cone-shaped networks with 4 hidden layers and 10 - 20 nodes per layer were able to detect the main 10 states more often and produced better results. An approach to increase the complexity of the network by constructing wider networks with 300 nodes in 4 hidden layers each (see 5.6) produced similar results.
While we used VAMPnets on the well understood AIB_9, the method is designed to be a scientific analysis tool to acquire knowledge about unfamiliar peptides. In our study, applying VAMPnets multiple times with identical parameters on the same data set has yielded highly varying results. While some clusterings separated the 10 most populated states correctly, others even failed to isolate the 2 main states. A scientific method of analysis is expected to be yield reproducible results. Due to the inevitable use of stochastic methods in machine learning on large data sets, exact reproducibility can not be given by design. Ideally, we would expect converging results when the networks are trained long enough. We have seen that parameter wise identical copies of VAMPnets converge to different clusterings.
Since no general convergence is given, we could hope that the VAMP2 score can be used to identify the best result out of a set of trained networks. We saw that the best clusterings did not have the best VAMP2 score in both sets of cone-shaped and rectangular shaped networks. If we had not known the expected states from the Ramachandran clustering beforehand, we would not be able to detect the best result via VAMP2 score. This is unfortunate since some copies of the network were able to find the 10 most populated states correctly but were not rated best.
Recent studies by the group of Ferguson encountered the problem of highly varying clustering results as well [31]. The new state-like coordinates VAMPnets find are slow since optimizing the VAMP2 score does implicitely increase timescales. The strongly nonlinear nature of neural networks allows to find a more arbitrary feature transformation that algorithms like TICA. Therefore the group of Ferguson tried to use VAMPnets as dimensionality reduction algorithm, where they interpreted the output of VAMPnets not as clustering but a set of slow collective variables that could be used by a different clustering algorithm to find states. While producing better clusterings than the original VAMPnets, the intention of those so called "statefree VAMPnets" contradicts the idea to simplify the analysis pipeline. Here, VAMPnets become just another dimensionality reduction algorithm in the common workflow.

We now want to reflect why the VAMP2 score appears to be an inaccurate measure for the quality of a clustering. Our main criterion for a good clustering is that the states are separated in coordinate space. Ideally, the state borders are at at the highest free energy barriers. Since the VAMP2 score depends on the timescales resulting from the clustering of the trajectory, VAMPnets are expected to do a separation at the highest potential barriers. However, the eigenvalues of the Koopman operator are squared when calculating the VAMP2 score. Therefore the total score is dominated by the slowest processes. An additional fast processes does not increase the VAMP2 score much. We have seen that it is possible to find a clustering to 10 states that already has an optimal VAMP2 score of ≈ 2. Since the addition of another fast processes that would need the isolation of minor states does not change the VAMP2 score significantly, the network does not see a necessity in finding minor states. This is why VAMPnets are only able to

resolve a minority of the 32 states in AIB_9.
Another problematic aspect is that the VAMP2 score is directly calculated from the Koopman operator of the outputs of VAMPnets. Those outputs are vectors with probability like entries that describe the certainty of a point being assigned to a state (fuzzy clustering). We are interested in a hard clustering where each point of the trajectory has only one state label. A transition from a fuzzy to a hard clustering removes all information except the location of the maximum from the probability distribution. Therefore all information about the evolution of the probabilities that were not the highest entries is lost in the process. This is problematic, since VAMPnets optimize the timescales of processes in all vector components. In the hard clustering, processes in non maximum components do not exist anymore. As seen in figure 20, there are networks, where 90% of the probability is distributed among other states for almost half of the trajectory points. Therefore the implied timescales in the fuzzy clustering are expected to be higher than the timescales in the Markov state model that is build from the hard clustering. To estimate the influence of this effect, further study is required. It is expected that some network architectures are more vulnerable since the distribution of probabilities in the fuzzy clustering varies highly as figure 20 has shown.

In the end, we want to think about two possible improvements to VAMPnets and discuss if they could provide solutions for the discovered problems.
A possible improvement regarding the dynamics in the fuzzy clustering could be an additional layer behind the original output layer. In this layer, the hard clustering of the fuzzy clustering is calculated. When building VAMPnets with this new output layer, the VAMP2 score would be calculated directly from the processes in the hard and not the fuzzy clustering which ensures that only timescales of processes visible in an Markov state model are taken into account.
Another improvement was suggested in a tutorial of the official VAMPnets repository [30]. They proposed a meanfree interpretation of the fuzzy clustering. In the cases we studies, we saw that this approach generates only minor improvements for networks that performed badly beforehand. The clusterings of networks that already performed relatively well in did not change significantly. However, when using meanfree fuzzy clusterings, the interpretation of fuzzy clustering entries as probabilities is lost, what makes this modification problematic.
While modifications may improve the quality of clusterings slightly, the main problem remains that the VAMP2 score does not measure the quality of a clustering in terms of geometric separation precise enough and is unable to resolve minor states. AIB_9 is a relatively simple system. If we were to move on to larger peptides where even more states are to be detected, we expect VAMPnets to perform worse. Additionally we expect to encouter major problems when analyzing systems where processes with large timescales are not necessarily important processes. To really fix the problems of VAMPnets we would need another metric than the VAMP2 score that better represents our understanding of a good clustering.

A. Appendix

A.1. Proof that 1-norm of eigenvectors with eigenvalue $\neq 1$ is 0

Let M be an $n \times n$ transitionmatrix with normalized rows. V is an orthogonal matrix.

$$\sum_j M_{i,j} = 1 \qquad M_{i,j} \geq 0 \tag{36}$$

$$M = V^{-1}\Lambda V \qquad \text{with } \Lambda = \text{diag}(\lambda_1, ..., \lambda_n) \tag{37}$$

Proof:

$$\sum_j M_{i,j} = 1 \tag{38}$$

$$\sum_j \sum_k \lambda_k (V^{-1})_{i,k} V_{k,j} = 1 \qquad | \cdot \sum_i V_{n,i} \tag{39}$$

$$\sum_{j,k} \lambda_k V_{k,j} \sum_i V_{n,i} (V^{-1})_{i,k} = \sum_i V_{n,i} \tag{40}$$

$$\sum_{j,k} \lambda_k V_{k,j} \delta_{n,k} = \sum_i V_{n,i} \tag{41}$$

$$\lambda_n \sum_j V_{n,j} = \sum_i V_{n,i} \tag{42}$$

This equation is true for

$$\lambda_n = 1 \qquad \vee \qquad \sum_i V_{n,i} = 0 \tag{43}$$

$$\square$$

Therefore the 1 norm of all vectors with and eigenvalue different than 1 has to be 0
This Proof was provided by Matthias Post.

A.2. Supplementary images

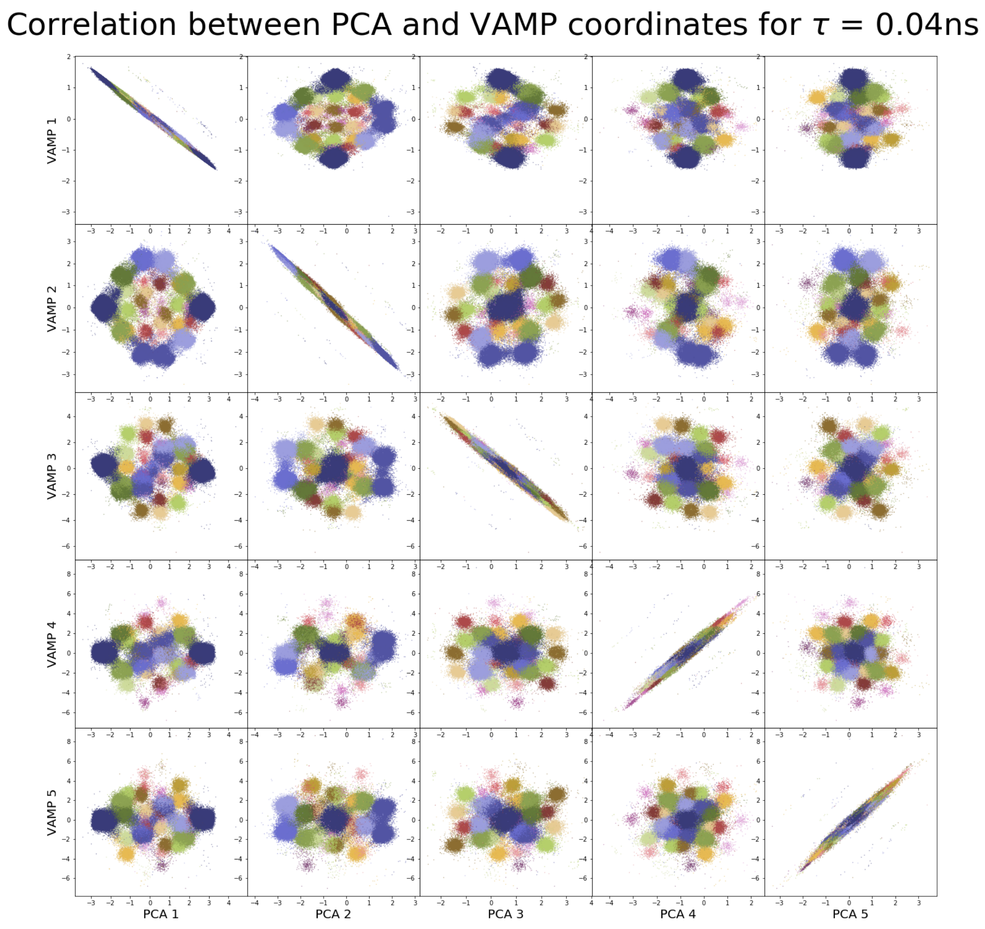


Figure 29: Projections from VAMP (y-axis) and PCA (x-axis) of the AIB$_9$ MELD data points. Every 10th point is plotted. For VAMP a lag time of $\tau = 0.04$ ns (1 frame) was chosen. The color code is a state label found with density based clustering.

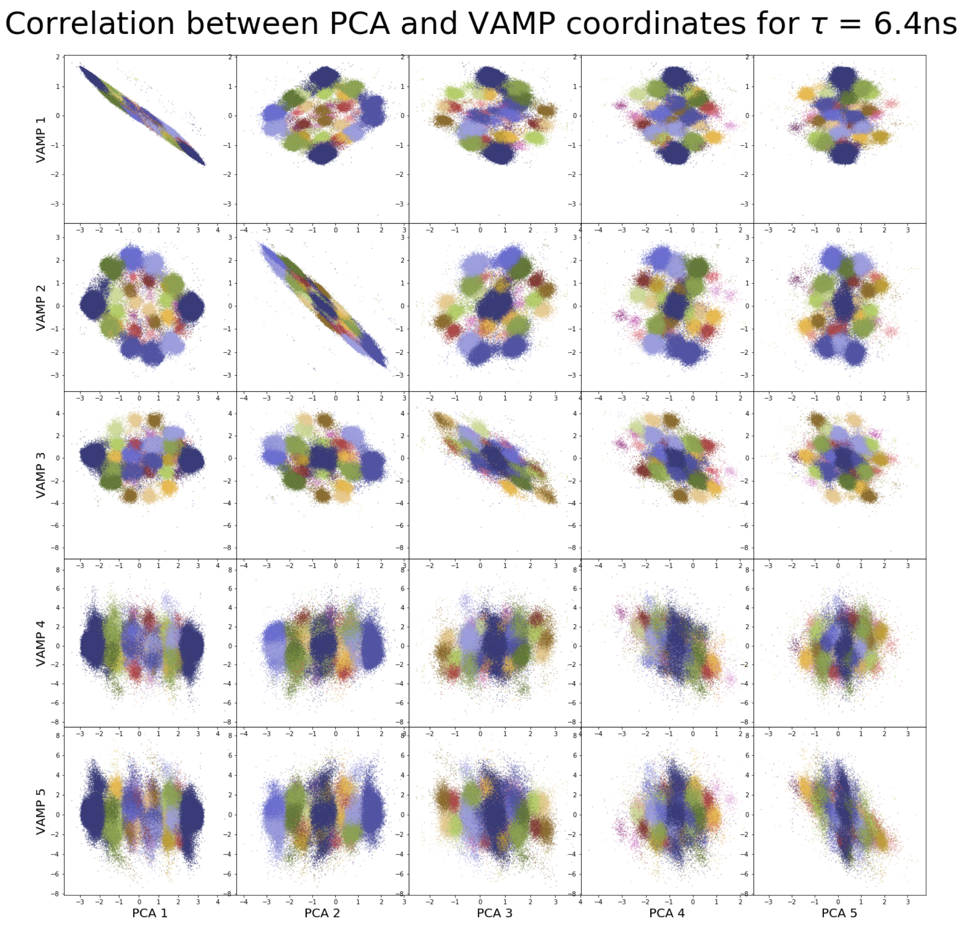


Figure 30: Projections from VAMP (y-axis) and PCA (x-axis) of the AIB$_9$ MELD data points. Every 10th point is plotted. For VAMP a lag time of $\tau = 6.4$ ns (160 frames) was chosen. The color code is a state label found with density based clustering.

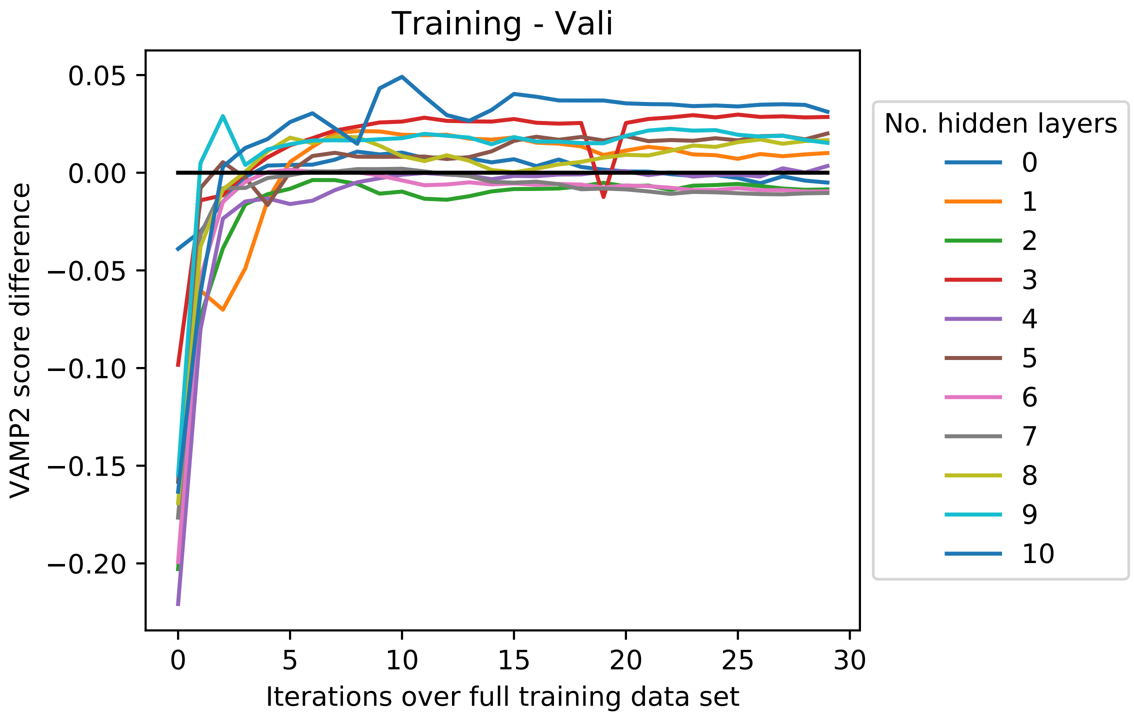


Figure 31: Deviation of VAMP2 scores of training data and validation data after each epoch of training for cone-shaped networks with different depths

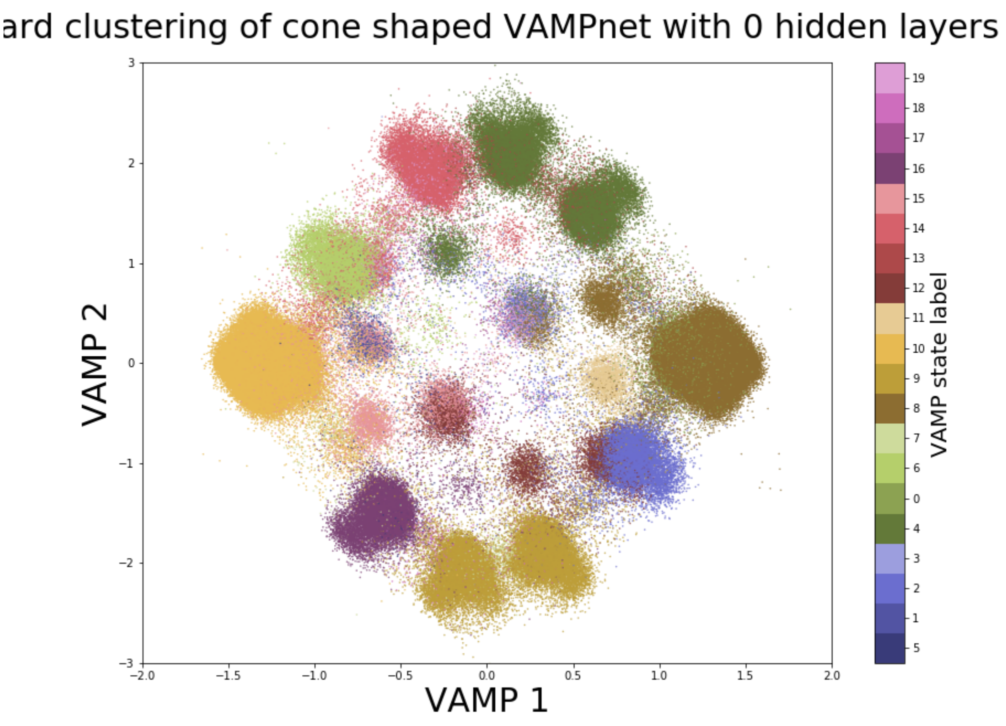


Figure 32: Network without hidden layer. 6th best VAMP2 score of all 11 cone-shaped networks with different depths.

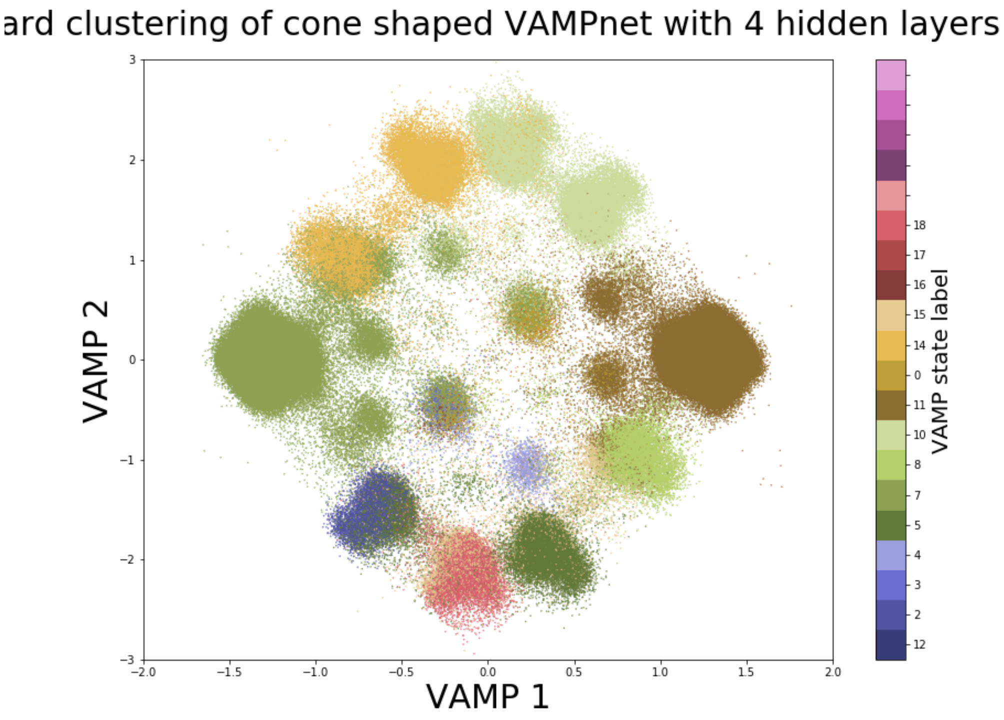


Figure 33: Cone-shaped network with 4 hidden layers. Best VAMP2 score of all 11 cone-shaped networks with different depths.

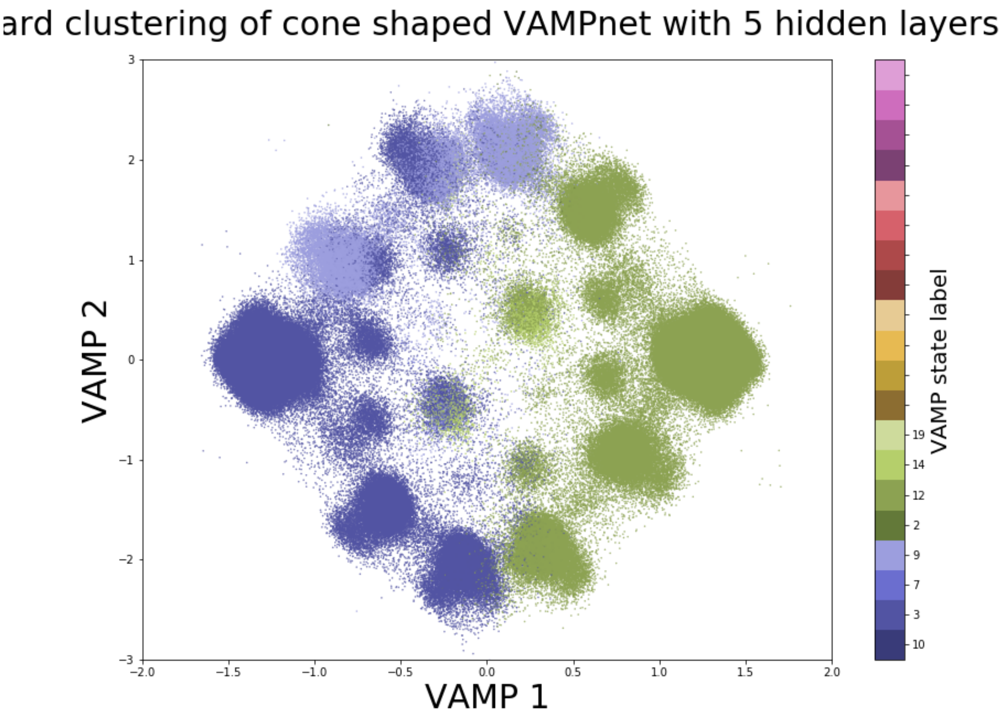


Figure 34: Cone-shaped network with 5 hidden layers. Worst VAMP2 score of all 11 cone-shaped networks with different depths.

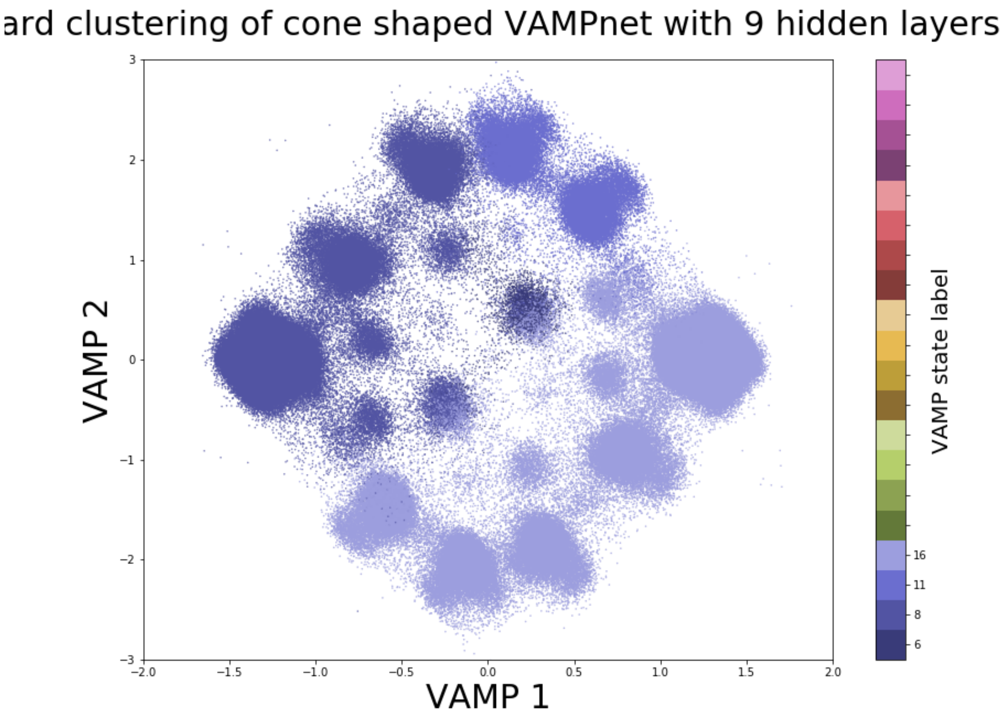


Figure 35: Cone-shaped network with 9 hidden layers. 5th best VAMP2 score of all 11 cone-shaped networks with different depths.

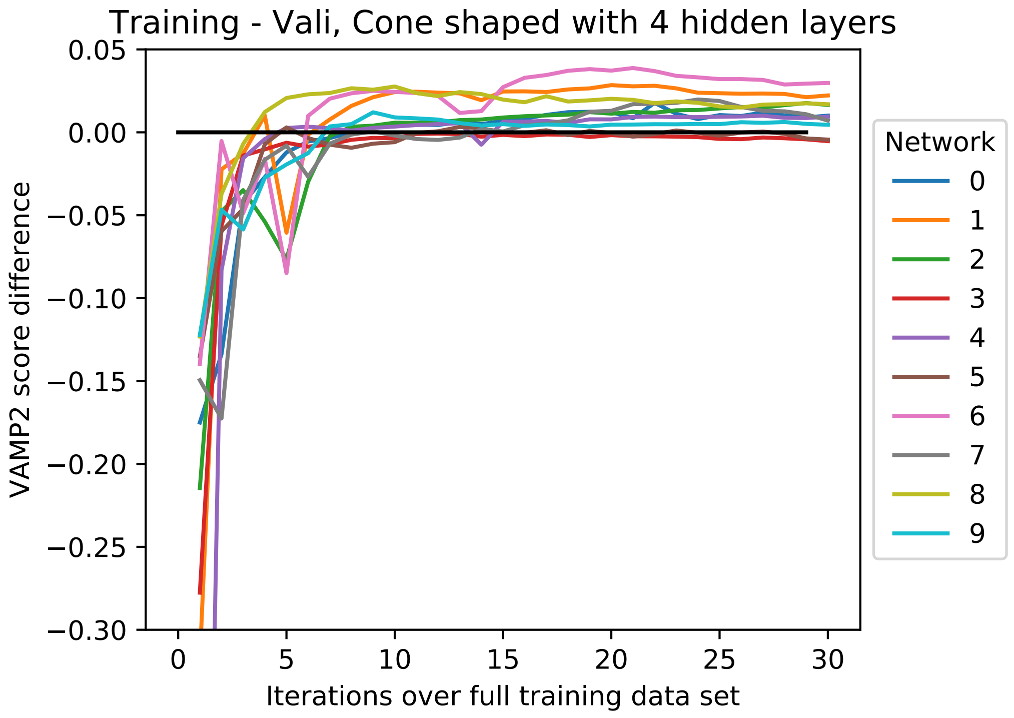


Figure 36: Deviation of VAMP2 scores of training data and validation data after each epoch of training for identical copies of cone-shaped networks with 4 hidden layers.

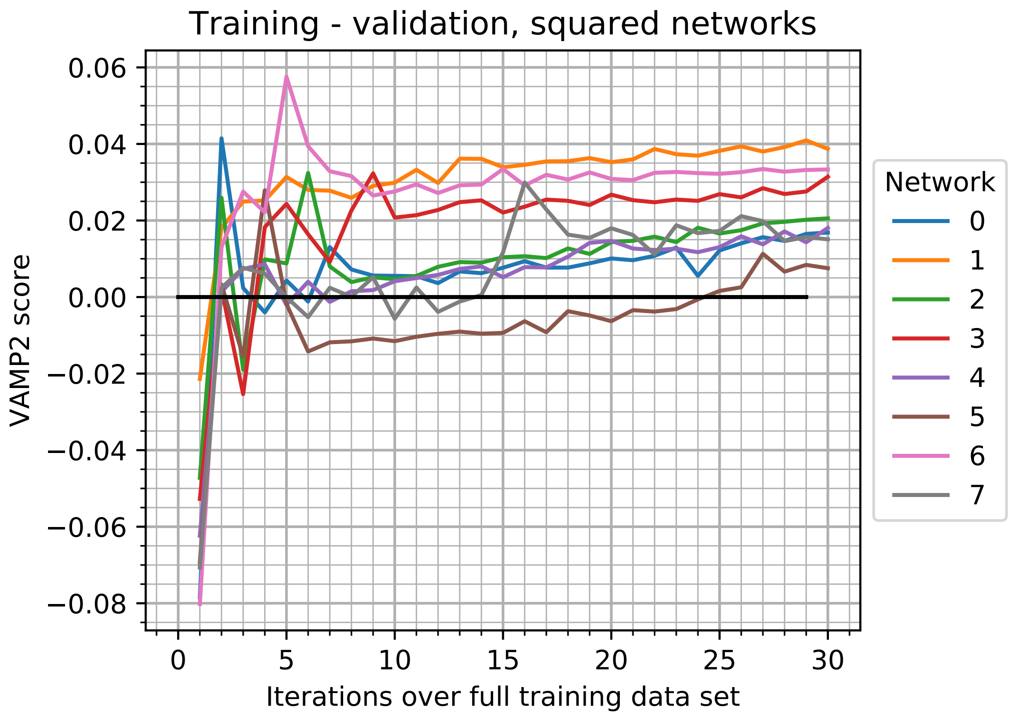


Figure 37: Deviation of VAMP2 scores of training data and validation data after each epoch of training for rectangular shaped networks with 4 hidden layers.

References

[1] Florian Sittel and Gerhard Stock. Perspective: Identification of collective variables and metastable states of protein dynamics. *The Journal of Chemical Physics*, 149(15):150901, 2018.

[2] I. T. Jolliffe. *Principal Component Analysis.* Springer-Verlag New York, 2002.

[3] Guillermo Pérez-Hernández, Fabian Paul, Toni Giorgino, Gianni De Fabritiis, and Frank Noé. Identification of slow molecular order parameters for Markov model construction. *The Journal of Chemical Physics*, 139(1):015102, 2013.

[4] Hao Wu and Frank Noe. Variational approach for learning Markov processes from time series data. *arXiv:1707.04659v1*, 2017.

[5] Florian Sittel and Gerhard Stock. Robust density-based clustering to identify metastable conformational states of proteins. *Journal of Chemical Theory and Computation*, 12(5):2426–2435, May 2016.

[6] Mardt Andreas, Pasquali Luca, Wu Hao, and Noé Frank. VAMPnets for deep learning of molecular kinetics. *Nature Communications*, 9(1):5, 2018.

[7] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Univ. Helsinki*, 1970. Master's Thesis (in Finnish).

[8] Alexander Radovic, Mike Williams, David Rousseau, Michael Kagan, Daniele Bonacorsi, Alexander Himmel, Adam Aurisano, Kazuhiro Terao, and Taritree Wongjirad. Machine learning at the energy and intensity frontiers of particle physics. *Nature*, 560(7716):41–48, 2018.

[9] Ashok Chandrashekar, Fernando Amat, Justin Basilico, and Tony Jebara. Artwork personalization at netflix. `https://netflixtechblog.com/artwork-personalization-c589f074ad76`, 12 2017.

[10] Alberto Perez, Florian Sittel, Gerhard Stock, and Ken Dill. Meld-path efficiently computes conformational transitions, including multiple and diverse paths. *Journal of Chemical Theory and Computation*, 14(4):2109–2116, 2018. PMID: 29547695.

[11] Daniel Nagel, Anna Weber, Benjamin Lickert, and Gerhard Stock. Dynamical coring of Markov state models. *The Journal of Chemical Physics*, 150(9):094111, 2019.

[12] David A. Case, Thomas E. Cheatham III, Tom Darden, Holger Gohlke, Ray Luo, Kenneth M. Merz Jr., Alexey Onufriev, Carlos Simmerling, Bing Wang, and Robert J. Woods. The amber biomolecular simulation programs. *Journal of Computational Chemistry*, 26(16):1668–1688, 2005.

[13] Sander Pronk, Szilárd Páll, Roland Schulz, Per Larsson, Pär Bjelkmar, Rossen Apostolov, Michael R. Shirts, Jeremy C. Smith, Peter M. Kasson, David van der Spoel, Berk Hess, and Erik Lindahl. GROMACS 4.5: a high-throughput and highly

parallel open source molecular simulation toolkit. *Bioinformatics*, 29(7):845–854, 02 2013.

[14] Nanjiang Shu and Tuping Zhou. Describing and comparing protein structures using shape strings. *Current protein & peptide science*, 9:310–24, 09 2008.

[15] Rainer Hegger, Alexandros Altis, Phuong H. Nguyen, and Gerhard Stock. How complex is the dynamics of peptide folding? *Phys. Rev. Lett.*, 98:028102, Jan 2007.

[16] Florian Sittel, Thomas Filk, and Gerhard Stock. Principal component analysis on a torus: Theory and application to protein dynamics. *The Journal of Chemical Physics*, 147(24):244101, 2017.

[17] Paul A. Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation.* USA, NJ: John Wiley & Sons, 2017. pp. 9–11.

[18] Dennis S. Bernstein. *Matrix Mathematics: Theory, Facts, and Formulas (Second Edition).* Princeton University Press, 26. 7. 2009. p. 97.

[19] Walter Nef. *Linear Algebra.* Courier Corporation, 01. 01. 1988. p. 35.

[20] Jan-Hendrik Prinz, Hao Wu, Marco Sarich, Bettina Keller, Martin Senne, Martin Held, John D. Chodera, Christof Schütte, and Frank Noé. Markov models of molecular kinetics: Generation and validation. *The Journal of Chemical Physics*, 134(17):174105, 2011.

[21] Andrzej J. Rzepiela, Norbert Schaudinnus, Sebastian Buchenberg, Rainer Hegger, and Gerhard Stock. Communication: Microsecond peptide dynamics from nanosecond trajectories: A langevin approach. *The Journal of Chemical Physics*, 141(24):241102, 2014.

[22] François Chollet et al. Keras. `https://keras.io`, 2015.

[23] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[24] Murat Sazli. A brief review of feed-forward neural networks. *Communications, Faculty Of Science, University of Ankara*, 50:11–17, 01 2006.

[25] Tianqi Wang, Chao-Kai Wen, Hanqing Wang, Feifei Gao, Tao Jiang, and Shi Jin. Deep learning for wireless physical layer: Opportunities and challenges. *China Communications*, 14:92–111, 10 2017.

[26] Balázs Csanád Csáji. Approximation with artificial neural networks. *MSc Thesis, Eötvös Loránd University (ELTE), Budapest, Hungary*, 2001.

[27] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[28] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 116, New York, NY, USA, 2004. Association for Computing Machinery.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[30] Bernhard Reuter (msmdev). Vampnets github repository. `https://github.com/msmdev/deeptime-vampnet-tensorflow-gpu-install`, 11 2018.

[31] Wei Chen, Hythem Sidky, and Andrew L. Ferguson. Nonlinear discovery of slow molecular modes using state-free reversible vampnets. *The Journal of Chemical Physics*, 150(21):214114, 2019.