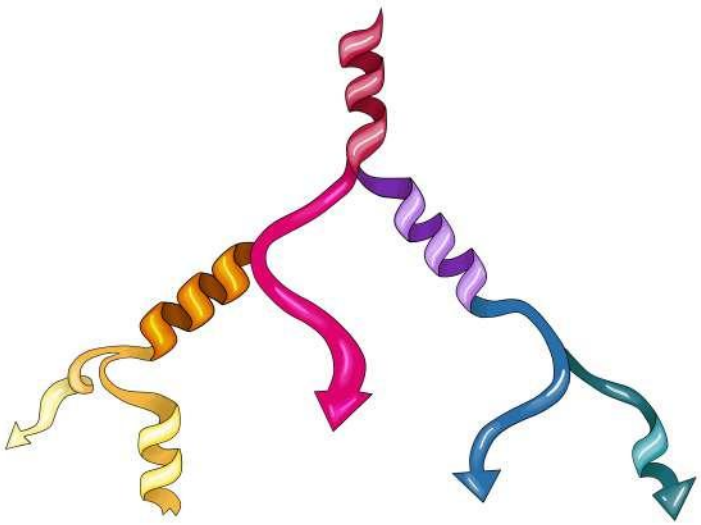
# Deep probabilistic programming applied protein superposition, protein structure prediction and ancestral sequence resurrection.

Lys Sanz Moreta



UNIVERSITY OF
COPENHAGEN

# 1 Acknowledgements

I would like to thank my supervisor Professor Thomas Hamelryck for his supervision and tutelage during the course of my Ph.D. degree. My gratitude extends to the Independent Research Fund of Denmark for the funding opportunity to undertake my studies at the Department of Computer Science, University of Copenhagen. Additionally, I would like to express gratitude to my research group co-workers for their mentorship and patience. My appreciation also goes out to my family and friends for their encouragement, patience, and support all through my studies.

# 2 Preface

Here, I present my thesis compendium of three years and 3 months of work, from 1st November 2018 until 31st January 2022, at the Department of Computer Science, at the University of Copenhagen, in Copenhagen, Denmark, under the supervision of Professor Hamelryck.

Lys Sanz Moreta

# Contents

# 3 Abstract

## 3.1 English

The content of this thesis covers several concepts associated with structural bioinformatics, molecular evolution, and probabilistic programming. It includes new methods for performing protein superposition, protein structure prediction, and ancestral sequence resurrection.

The first manuscript embarks into protein superposition by presenting Theseus-PP [1]. This new method uses a Bayesian approach, instead of the Maximum Likelihood method implemented in the original Theseus [2], which allows introducing relevant priors over the model's parameters. The superposition model is contemplated as a new type of error loss function that will assist during protein structure inference.

The second manuscript extends the previous Theseus-PP into Theseus-HMC [3], this method uses Hamiltonian Monte Carlo inference, concretely the No-U turns sampler [4], to allow the computation of uncertainty over the parameters needed for the superposition problem.

The third manuscript implements an adaptation of the generative Deep Markov Model [5] for the prediction of protein fragments libraries [6]. Deep Markov Models are an extension of classical Hidden Markov Models that instead use both amortized inference and gated neural networks (such as recurrent neural networks [7] ) over the emission and transition probabilities to preserve long-range dependencies across the sequences. This new variation of the DMM benefits from Bayesian inference to compute uncertainty over the fragment's predictions.

The last manuscript proposes a unique approach to Ancestral Protein Resurrection that overcomes factorized evolution and encodes sequence evolution using a tree-structured Ornstein–Uhlenbeck latent process [8].

## 3.2 Dansk

Denne afhandling dækker koncepter fra strukturel bioinformatik, molekylær evolution og probabilistisk programmering. Den omfatter nye metoder til at udføre proteinsuperposition, forudsigelse af proteinstruktur og gendannelse af forfædres gensekvenser.

Det første manuskript går i gang med protein superposition ved at præsentere Theseus-PP [1]. Denne nye metode anvender en Bayesiansk tilgang i stedet for Maximum Likelihood-metoden, der er implementeret i den oprindelige Theseus [2], hvilket gør det muligt at indføre relevante priors over modellens parametre. Superpositionsmodellen er tænkt som en ny type tabsfunktion, der vil være til hjælp i forbindelse med proteinstrukturinferens.

Det andet manuskript udvider den tidligere Theseus-PP til Theseus-HMC [**?**], denne metode anvender Hamiltonian Monte Carlo-inferens, konkret No-U turns sampler [4], til at muliggøre beregning af usikkerhed over de parametre, der er nødvendige for superpositionsproblemet.

Det tredje manuskript implementerer en tilpasning af den generative Deep Markov Model [5] til forudsigelse af biblioteker af proteinfragmenter [6]. Deep Markov Models er en udvidelse af klassiske Hidden Markov Models, der i stedet anvender både amortiseret inferens og gated neurale netværk (såsom recurrent neural networks [7] ) over emissions- og overgangssandsynlighederne for at bevare langtrækkende afhængigheder på tværs af sekvenserne. Denne nye variant af DMM drager fordel af Bayesian-inferens til beregning af usikkerheden over fragmentets forudsigelser.

Det sidste manuskript foreslår en unik tilgang til Ancestral Protein Resurrection, der overvinder faktoriseret evolution og koder sekvensudvikling ved hjælp af en træstruktureret Ornstein-Uhlenbeck latent proces [8].

# 4 Introduction

Here I introduce the guideline theoretical concepts that have directed the development of this thesis. The statistical models implemented in the papers have been developed under the probabilistic programming libraries of Pyro [9] and Numpyro [10] that offer optimized automatic Bayesian inference.

# 5 Protein folding problem

## 5.1 What is the protein folding problem?

Resolving the 3D structure of proteins brings us a step closer to defining their functional role. Investigating the protein structure leads to their potential usage, on the early stages of research, as drug targets or templates for biotechnological tools. For example, synthesis of antibodies or peptides which yield multiple therapeutic applications [11].
The protein folding mechanism encountered a major breakthrough in 1950-1962 thanks to Christian Anfinsen and his *Thermodynamic Hypothesis* [12]. His hypothesis postulated that the native or natural conformation of the protein is that one which generates the most thermodynamically stable structure in the intracellular aqueous medium. Leading to the conclusion that the folding of the structure does not depend on the action of the ribosome or the chaperones during the protein synthesis. Most of them can be simply refolded in a test tube, with the exception of insulin, the $\alpha$-lytic protease and the serpin. This meant that the folding process is a result over the spatial constraints of the peptide bonds determined by the chemical and physical properties of the amino acids. The hypothesis was tested by denaturalizing the RNase enzyme and observing that the enzyme's amino acid structure refolded spontaneously back into its original form when it was returned to a medium with similar conditions to the intracellular medium. In 1972, Anfinsen summarized the phenomena as "The native conformation is determined by the totality of inter-atomic interactions and hence by the amino acid sequence, in a given environment"[12].

## 5.2 The protein structure

The protein sequence is delineated by the covalent union of amino acids via the peptide bond, see Figure 1. The peptide bond is formed by the union of the carboxyl group of the first amino acid with the amino group of the second. Meaning that the protein sequence starts at the C-terminal (carboxyl) and ends at the N-terminal. The nature of the protein sequence is dependent on the combination of the 20 different types of existing amino acids (See Figure 18). The differences among them rely on the composition of the side chains, which confer different chemico-physical properties to the molecule. The type of amino acid is essential to configure the disposition of the amino acids in an aqueous medium. Hydrophobic chains will be located in the inside of the structure, minimizing the contact with water, and the hydrophillic elements will be oriented towards the exterior [13, 14].
The 2D protein sequence is turn into a 3D structure were the backbone is folded according to the dihedral or torsion angles. The dihedral angles are designated as $\omega$, $\phi$ and $\psi$. The $\phi$ and $\psi$ angles are formed between the hyperplanes originates among the 2 different sets of 4 atoms marked in light blue in Figure 1. The $\omega$ angle is restricted to values 0 or 180 and it's prediction is trivial compared to the $\phi$ and $\psi$ angles, therefore, often ignored [15, 16, 17].

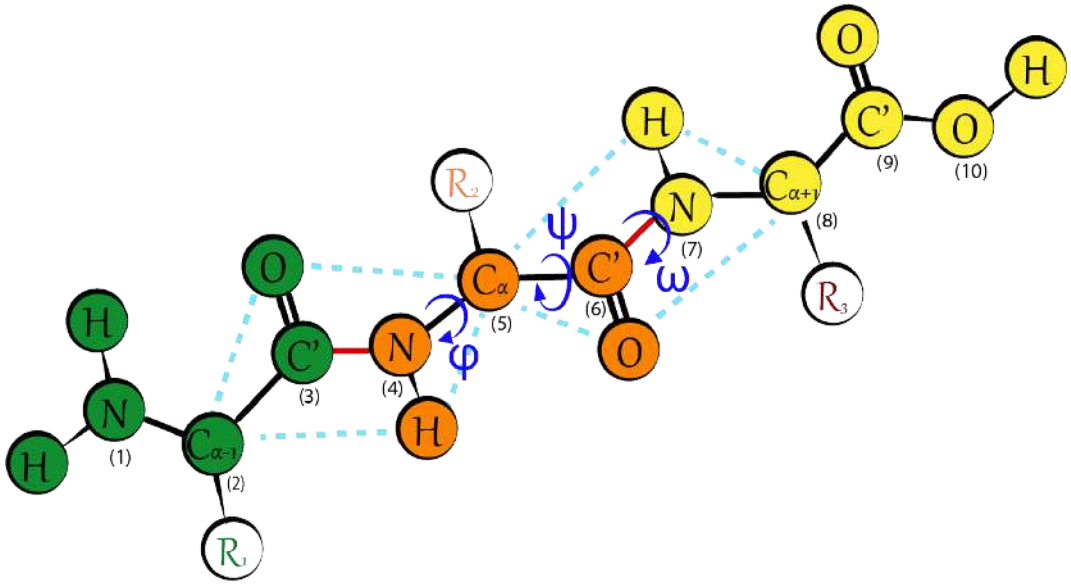



**Figure 1:** Peptide Bond. The picture represents 3 amino acids connected by 2 peptide bonds (highlighted in red). It shows how the $\phi$ and $\psi$ angles, correspondent to the center amino acid (orange), are formed by 2 different hyperplanes. The $\phi$ angle hyperplane is formed by atoms $C_{\alpha-1}(2)$,$O(3)$,$H(4)$ and $C_{\alpha}(5)$. The $\psi$ hyperplane is formed by atoms $C_{\alpha+1}(8)$,$O(6)$, $H(7)$ and $C_{\alpha}(5)$. The 3 side chains are indicated with $R$.

## 5.3 Predicting protein structures

The human proteome is composed of a large number of proteins, proposed to range between 10,000 and several billion different types [18]. The experimental determination of their 3D structure is carried out via different crystallographic techniques which, despite their recent amelioration and automatization, they still consist of lengthy, uncertain and costly procedures. The cost of generating the structure can range from 140.000 up to 2.5 million dollars [19, 14].

These drawbacks motivated the community to pursue solving protein folding and protein structure prediction using statistical models. Hence, the CASP (Critical Assessment of Techniques for Protein Structure Prediction) competition was established in 1994 (University of Maryland) in order to offer fair assessment and guarantee generalization and reproducibility of the protein structure methods presented by different research groups [20].

The CASP competition relies on measuring accuracy of the predictions via the Global distance test total score (GDT-TS). Using this measurement overcomes the issues given by the Root Mean Squared Error (RMSD) which is not able to capture partially correct fragments achieved by the predictions. GDT-TS superimposes the $C_{\alpha}$ atoms within four distance cutoffs 1Å, 2Å, 4Å, and 8Å. The GDT-TS i) counts the number of residues falling within each of those distances cutoffs ii) reports the average of the residues within each cutoff, iii) adds the averages iv) divides them by four. This sustains that scores $\sim 20$ are considered random predictions, $\sim 50$ present an overall correct topology, $\sim 70$, present an accurate global and local topology, $> 80$ have high structural detail and $> 95$ is equal to the accuracy obtained from experimental data [21, 20].

## 5.4 The progression in protein structure prediction

The community has evolved from solving protein folding using mainly template-based modelling, meaning that homologous structures are used as a reference to make a prediction, to tackle more challenging protein targets and performing a more generalized free modelling (*ab initio*) without the use of templates [22]. The most re-known protein prediction framework is Rosetta, a computational method that relies on Monte Carlo simulated annealing search to find the proteins with the most stable energy score function [23, 24, 25].

The first breakthrough that improved predictions was modelling inter-residues contact maps. Contact maps model how mutations throughout the Multiple Sequence Alignment (MSA) are correlated and therefore how residues co-variate. This meant that they could capture if two residues were in physical contact in the structure (even though they were not close in the sequence) and could, consequently, influence each others mutations.

Contact maps research evolved into Direct coupling analysis (DCA) [26, 27, 28], a technique that measures the strength of the correlation between 2 amino acid sites in the sequence, even if there is not a *direct* correlation between them. DCA developed a covariance analysis calculated simultaneously between all residues in a protein sequence.

In 2017 [29] the first deep learning method to be able to learn high quality distance contact maps from a MSA was developed. This new methodology was able to make predictions even with the smallest datasets of structure homologs and allowed to generate a high quality geometric fingerprint of the underlying fold.

In 2018, during CASP13, AlphaFold, a branch from the research group DeepMind, highly refined the mapping of the sequence to a 3-dimensional structure. They developed a framework consisting of Convolutional Neural Networks for prediction of the $C_\alpha$ residue contact map from the MSA and the corresponding torsion angles. First, they estimated a statistical scoring potential from the negative log likelihoods of the distances. Secondly, they calculated a scoring potential for the sampled torsion angles. Finally, they introduced an already existing term utilized within Rosetta for measuring Van der Waals forces to prevent steric clashes. Their optimization problem reduced to minimizing those potentials, which can be considered as *pseudo-energies*, to improve their prediction accuracy score [30].

Lastly, during CASP14, in 2020, once more AlphaFold2 remarkably achieved the highest GDT-TS scores for the mapping of the sequence to the structure of the protein, particularly in the case of monomers [20]. Due to their success, currently they are ambitiously predicting all the available protein sequences across all protein databases and aiming to change the research field with their protein structure predictions [31]. Their new model network inputs a residue pairwise distance matrix and a MSA divided by clusters of similar proteins whose PDB structure is not necessarily know to enhance the evolutionary information that the model intakes. These inputs are given to the Evoformer, a new neural network structure that benefits from attention and the transformer [32] mechanisms to deal with large blocks of sequences inputs. The Evoformer transforms the input into a MSA and pairwise distance matrix embedded representations. The embedded pairwise distance map is interpreted as a directed graph whose edges represent the distances between the nodes (residues). In order to interpret the estimated pairwise distances as 3D coordinates, they need to be corrected using attention mechanisms and the so called “multiplicative triangle update". Combining these techniques with multiple updates and corrections the model reaches unforeseen prediction accuracies [33].

Alphafold2 has undoubtedly achieved a milestone within the protein folding field, however many aspects still remain unsolved, specially the underlying physico-chemical phenomena that allows such precise folding in nature, protein stability [11], intrinsically disordered proteins [34], the extension to higher protein complexes [35], the exact modelling of interactions with binding agents (which typically requires ultra low resolution to guarantee exact binding) and overall the required flexibility of the protein conformations that allow interactions with the medium, which are currently absent within their static prediction approach [20, 34].

# 6 Ancestral Sequence Reconstruction (ASR)

Ancestral sequence reconstruction tackles the process of inferring ancestral sequences relying on the current observed ones. These procedures require the usage of evolutionary trees in order to resurrect the desired protein sequence at a given point in the evolutionary tree [36].

## 6.1 The nature of protein evolution

Protein sequences are composed by amino acids susceptible to change throughout time. Those changes are mutations which are responsible for changes in the functionality and stability of a protein. Mutations take place in the DNA sequence and are translated to the protein, following the semantics of the codon to amino acid table, see Figure 18. The codon to amino acid table establishes that each amino acid (there are 20 most representative amino acids) is represented by at least 1 codon (referred as the degeneracy of the codon table), a three letter code composed by any of the 4 nucleotide types (A, T, G, C). There are different types of mutations that can occur and affect the protein differently [37].

Substitutions are a type of mutation where the DNA sequence suffers a replacement of one nucleotide by another. This change can result in a) neutral/synonymous substitution where the amino acid does not change b) missense substitution where the amino acid changes c) nonsense, where the mutation translates to a stop codon and disrupts the sequence by truncating it [38].

Deletions are a type of mutations where a) 1 or more nucleotides are lost from the DNA sequence, this causes a frameshift that leads into a complete dysfunctional protein b) 3 contiguous nucleotides (codon) are lost, therefore only 1 amino acid is deleted from the sequence but the protein may still remain functional. Insertions are the opposite of deletions but the repercussions are similar, meaning that they can also result in frameshifts or the addition of amino acids that might appear deleterious or not [38].

## 6.2 Evolution through a phylogenetic tree

The phylogenetic tree embodies the occurrence of mutations and the correlations among the leaves through them. It reflects the evolutionary time passed from the most recent common ancestor until the currently observed sequences. Branch lengths or patristic distances are numerical representations of those evolutionary times and therefore capture the accumulation of mutations in a sequence. [36].

Current evolutionary models use mostly binary trees, where each node is only allowed to have 2 children. This assumption is not limiting since any other type of tree distribution, with more children, can be approximated by a binary tree where some of the tree branches are very short [39].

The currently observed sequences are placed in the leaves nodes (*taxa*), whilst the unseen ancestral sequences are located at the internal nodes, see Figure 6.2. True trees are rooted, they place a global most recent common ancestor (the root) in the phylogeny. Unrooted trees lack a concrete most recent ancestor and the internal nodes are shared across all the leaves. Consequently, we cannot know which 2 leaves share the same ancestor. The latter ones are more commonly used in phylogeny inference, due to their lower amount of tree topology combinations, in comparison to rooted trees. A rooted tree with $n$ leaves there are $(2n-3)!!$ number of trees and $(2n-5)!!$ for the unrooted case [39, 40].

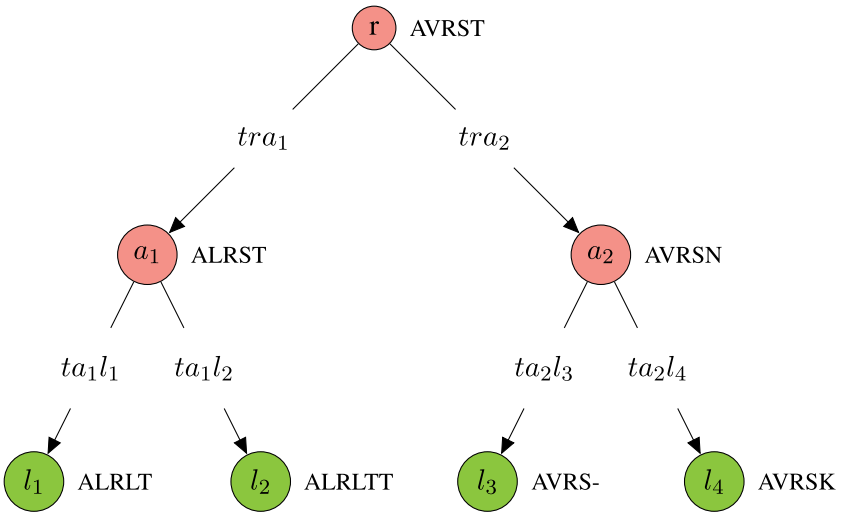



**Figure 2:** Evolution of a sequence through a binary phylogenetic tree. Leaves are labelled as $l$, ancestors as $a$ and the root as $r$. The phylogenetic or evolutionary distance among the nodes (branch lengths or patristic distances) is labelled as $t$.

## 6.3 Methods in Phylogenetics and ASR

Phylogenetics optimizes the topology of the phylogeny under certain objective functions. Those objective functions or criteria set a series of assumptions that attempt to approach the true phylogeny as the number of data increases.

### 6.3.1 Maximum Parsimony methods

The first criterion to be explored was the *parsimony criterion*. Maximum Parsimony (weighted [41] or traditional [42]) attempts to construct a tree that minimizes the tree length. The tree length is described as the minimum number of mutations over all sites achieved when comparing between different taxa arrangements. The optimization problem is described as follows,

$$\text{len(tree)} = \sum_{b=1}^{B} \sum_{j=1}^{\mathrm{n}_L} w_j cost(n_{s1,j}, n_{s2,j}) \tag{1}$$

where $B$ is the number of branches, $\mathrm{n}_L$ is the number of sites (nucleotide or amino acid positions in the alignment), $n_{s1}$ and $n_{s2}$ are the 2 nodes connected by branch $b$ and $n_{s1,j}$ and $n_{s2,j}$ are their corresponding sites [36]. The *cost* function specifies the cost of a mutation and it's assigned a weight $w_j$ per site. Solving for the most parsimonious phylogeny is an *NP*-hard problem [43] and the criterion might be statistically inconsistent [44], therefore alternative objective functions were proposed subsequently [45].

### 6.3.2 Distance-based methods

The second criterion is the *distance-based* objective. This approach finds the best phylogeny that fits a given $n \times n$ matrix of evolutionary distances $D$ among the observed sequences $X$. This evolutionary distance matrix must follow certain characteristics in order to be feasible within phylogenetic inference. These characteristics say that it must be a: i) *dissimilarity matrix* ii) *metric* matrix: follows the triangle inequality iii) *additive* matrix: follows the Buneman's four point condition [46]. Alternatively, it can also be constructed as an ultrametric additive matrix if the distances are ultrametric (all leaves have the same distance to the most common recent ancestor) [45].

One of the most relevant distance-based criteria is the "minimum evolution". The methods that follow this criteria find a phylogeny whose sum of branch lengths is minimal. They are found to be generally statistically consistent and the computation of the distance matrix is fairly cheap [45]. It is important to note that these methods tend to perform their computations in unrooted trees, however this issue can be overcome with an added "dummy vertex" as a root. The general minimimun evolution problem (MEP) can be written as,

$$\begin{aligned} \min_{(T,t)} & L(T,t) \\ \text{s.t } & f(D,T,t) = 0 \\ & T \in \mathcal{T}, t \in \mathbb{R}^{(2n-3)} \end{aligned} \tag{2}$$

where $T$ is the topology, $t$ are the branch lengths, $D$ is the evolutionary distance matrix and $f(D, X, t)$ is the function that translates $D$ into a phylogeny. The function $f(D, X, t)$ can be subdivided into 2 subproblems : i) The determination of the optimal phylogeny $T$ ii) The determination of the optimal branch lengths $t$. The latter ones and referred as *edge weight estimation models* and can be subdivided into the Least-Squares Models and the Linear programming models [45].

The first Least-Squares method proposed was the Ordinary Least-Squares (OLS) method [47]. This model assumed that the evolutionary distances $d_{ij}$ (entries of $D$) were uniformly distributed accumulated mutations. This assumption was thought to be unrealistic and the model was revised to the Weighted Least-Squares (WLS) [48] which introduces some weights or variances $w_e$ over the distances $d_{ij}$. Later, in the Generalised Least-Squares (GLS) algorithm, the weights were replaced by co-variances, modelled by a Poisson process, to account for sites dependencies [49, 50]. These algorithms were demonstrated to be computationally infeasible and lack of means to deal with non-additive distance matrices generated by Markov Processes [51]. The field approached this challenge by building methods that enforced a positive constraint over the distance matrix $D$. These algorithms transformed the distance matrix $D$ to achieve additivity or ultrametricity. Among the most successful ones we can find the Balanced Least-Squares (BLS) edge-weight estimation model [52, 53], an algorithm that proved to be more biologically plausible and hold lower computational complexity [52, 53].

Solving MEP has been approached both with exact algorithms [54], those that guarantee to reach the correct solution by exhaustive enumeration of all phylogenies, and non-exact (approximation) algorithms, which use heuristics to provide reasonably good solutions in a quick fashion. There are 3 main types of heuristics, i) clustering: they exploit the *star decomposition algorithm* to iteratively merge the edges of a star formed by $n$ leaves ii) constructive: iteratively insert nodes to a partial phylogeny iii) clustering/constructive: a combination of the previous ones [45].

*Unweighted Pair Group Method using arithmetic Averages* or UPGMA [55] is a well-known and practical heuristic clustering method based on the OLS method. The method relies on progressive clustering of the sequences based on the distance $dij$. The distance $dij$ is set to the average distance between the pairs of sequences allocated in each of the clusters to be compared. In the first iteration, each sequence forms its own cluster. Each cluster is merged successively and compared in a paired form. The method requires an ultrametric additive matrix, meaning that the resulting branch lengths from the UPGMA trees are ultrametric. This transcribes to stating that distances from all the leaves to the root are identical (see Molecular Clock Theory [56]). This translates to establishing a constant rate of mutation at every point in the tree, meaning that it does not take into account that different types of proteins might mutate at different rates or that within the same protein, some parts might evolve faster than others [57]. Despite those disadvantages, this algorithm reaches a beneficially lower computational complexity of $\mathcal{O}(n^2)$ [45, 55].

Lastly, the Neighbour-Joining (NJ) algorithm, which is the most notable algorithm in phylogenetics, is also based in a clustering heuristic and the BLS model. The algorithm starts computing an initial evolutionary additive distance matrix which is pruned and updated as the clusters among nodes are formed [58].

### 6.3.3 Maximum-Likelihood methods

Alternatively, we can find the algorithms that maximize the *likelihood criterion*. The ML objective is to maximize the joint probability of the sequence alignment across independent sites $P(x\,T, t, Q)$ [59]. Maximum likelihood algorithms require an evolutionary model, in this case, a substitution model. The substitution model is a stationary Markov process whose substitution probability matrix is defined as $P(t)$. The substitution model assumes that the equilibrium frequencies and the transition probabilities remain constant through time.

The joint probability is the product of transition probabilities or probability of substitution from character $i$ to character $j$ in time $t$. The individual site probabilities are calculated from the transition rate matrix $Q$ and the branch lengths $t$, see Equation 3. At the same time, the rate matrix $Q$ is the adjusted product of the relative rate matrix $R$ (usually computed using distance-based methods) and the equilibrium frequencies $\Pi$. The aforementioned matrices have dimensionality $n \times n$, where $n$ is the number of available characters (4 in the case of nucleotides and 20 in the case of amino acids) [60]. The transition probability for an individual site can be expressed as,

$$P(t)_{ij} = e^{Qt} \tag{3}$$

The joint probability across sites is calculated using the log-likelihood summation for computation ease. The likelihood of the sequences is maximized by optimizing the branch lengths and the topology.

There are different types of substitution models based on the types of substitution matrices utilized. From the most simple uniform mutation rates and equal equilibrium frequencies [61] to the more flexible schemes that offer more biologically plausible solutions [62, 63, 64, 65].

### 6.3.4 Bayesian methods

Bayesian Inference methods enforced a Bayesian approach over the traditional Maximum Likelihood methods [66]. They included priors distribution over the tree topologies, the branch lengths and the

parameter of the substitution model. This allowed computing the posterior probability of the possible sets of model parameters $\theta$ given the sequence alignment $x$ [67],

$$P(\theta|x) = \frac{P(x|\theta) \times P(\theta)}{p(x)} \tag{4}$$

where the likelihood of the data given the tree is calculated using the traditional ML substitution models with incorporated priors over the model parameters. These algorithms use a Markov Chain Monte Carlo (MCMC) sampling method to estimate the posterior probability. The method starts by i) initializing the topology of the tree using NJ ii) sampling the model parameters iii) accepting or rejecting the model parameters according to Metropolis-Hastings criteria [68] iv) modifying the current tree. The sampling scheme is stopped when the convergence and chain mixing criteria are met.

### 6.3.5 Ancestral sequence reconstruction

The reconstruction of the ancestral sequences follows similar principles applied by ML methods [36]. They seek to maximize the likelihood or conditional probability of the set of ancestral sequences given the current sequences and a phylogeny. There are two main types of reconstruction methods a) joint reconstruction, which determines the most likely set of amino acids per site for all internal nodes b) marginal reconstruction estimation, which involves maximizing the marginal probability per individual ancestor node (not sites) by summing over the likelihoods of each of the nodes except the one of interest [69, 36, 70].

# 7 Introduction to Bayesian Inference

## 7.1 Essential concepts in probability theory

Probability theory is based on a handfull of probability concepts that are easier to illustrate using Figure 3. The figure represents the joint probability distribution of 2 random variables, individual marginal probabilities and conditional probabilities. The complete definitions of the probability concepts can be found in the Glossary section 19.

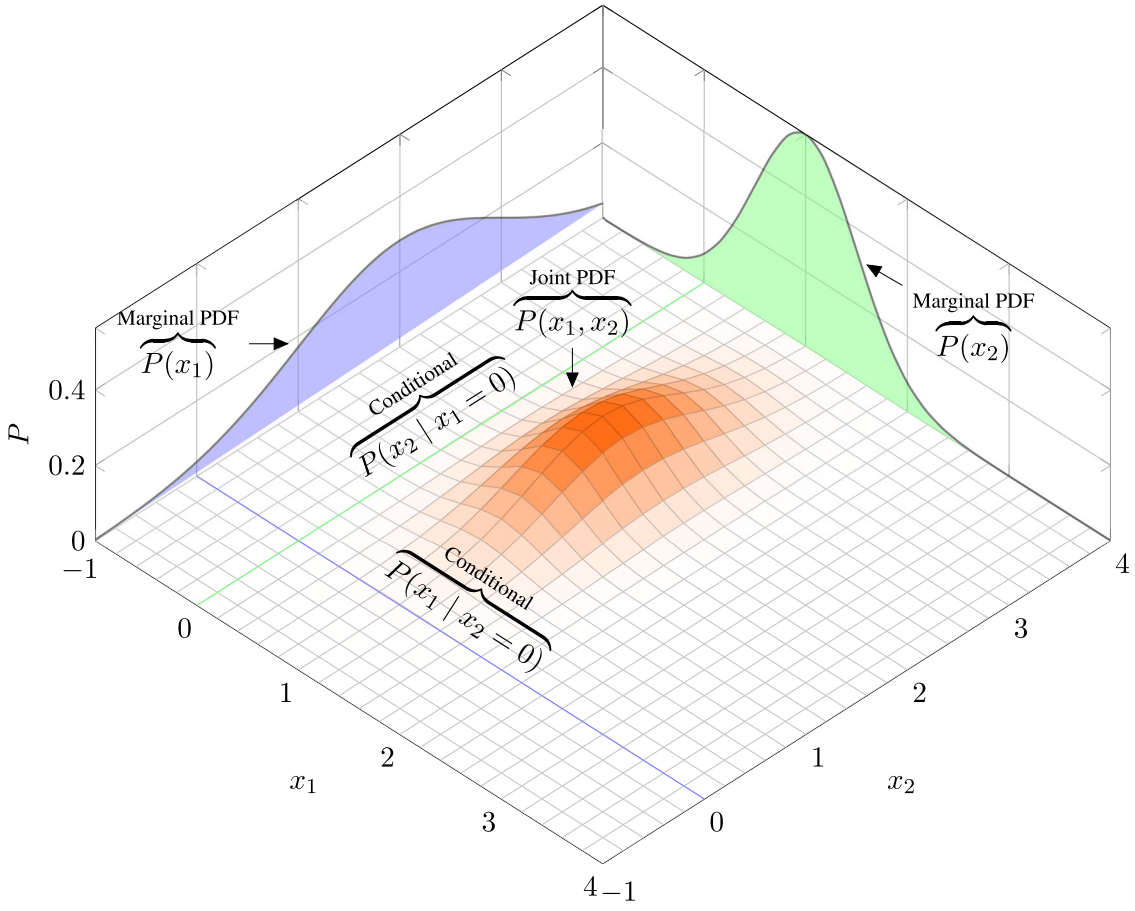



**Figure 3:** Illustration of the essentials concepts in probability theory applied to a bivariate Normal distribution of 2 random variables $x_1$ and $x_2$.

## 7.2 Bayes' Theorem

Bayes' Theorem has been part of the mathematical repertoire for over 200 years. The theorem was firstly drafted by a minister of a Presbyterian church, named Thomas Bayes, around the 1740s, who was advocated to deduce how to quantify the chance of something happening in the future given that another condition took place [71]. Unfortunately, his deduction was not discovered until the

ideologist's death in 1762. Despite him being the pioneer, the full theory on Bayesian probability was developed shortly after by Pierre-Simon Laplace, unaware of Bayes' proposition. Laplace was a french mathematician, an overall largely scientific achiever, who released a complete proof and postulate of the theorem around 1810-1814. Regardless of this fact, Bayes was still the designated name for the theorem due to a series of historical reasons [72, 73].
The Bayes theorem states that the conditional probability of a variable A given a variable B is the ratio between their joint probability and the marginal probability of B. The Bayes theorem can be used in statistical modelling to calculate the exact inference of the posterior probability of the parameters of the model $\Theta$ given the observed data $X$ as the following,

$$\overbrace{P(\Theta|X)}^{\text{Posterior}} = \frac{\overbrace{P(X,\Theta)}^{\text{Joint}}}{\underbrace{P(X)}_{\text{Evidence}}} = \frac{\overbrace{P(X|\Theta)}^{\text{Likelihood}}\overbrace{P(\Theta)}^{\text{Prior}}}{\int_{-\infty}^{\infty} P(X|\Theta)p(\Theta)d\Theta} = \frac{P(X|\Theta)p(\Theta)}{Z} \tag{5}$$

As can be seen in equation 5, the exact inference of the posterior probability is problematic due to the intractability of the integral of the marginal probability of the data. The intractability is due to the absence of close-forms solutions to solve the integral or infeasibility of the computations [74]. For simplification, the evidence can be assigned as the normalization constant $Z$. The marginal likelihood of the data $P(X)$ is usually omitted during inference, however, it is often used during model comparison or evaluation post-inference via the Bayes factor [75]. The marginal likelihood evaluates the probability of the observations at every possible value of $\Theta$. This can be calculated by marginalizing the parameters $\Theta$ over the joint distribution $p(X, \Theta)$. Equation 6 reflects how we can omit the estimation of the normalization constant.

$$\arg\max_{\Theta}(p(\Theta|X)) \propto \arg\max_{\Theta}(p(X,\Theta)) \propto \arg\max_{\Theta}(p(X|\Theta)p(\Theta)) \tag{6}$$

$\Theta$ : Parameters of the model
$X$ : Collection of $n$ observations , $X \in \{x_1, ..., x_n\}$
$P(\Theta|\text{X})$ : Posterior probability, conditional probability of the parameters given the observations.
$P(\Theta)$ : Prior probability, prior belief over the values of the model's parameters without taking into account the observations
$P(\text{X}|\Theta)$ : Likelihood, conditional probability of the observations being generated by the model's parameters.
$P(\text{X})$ : Evidence, marginal probability density of the data (observations) or marginal likelihood.

Within statistical modelling it is important to note that *generative* algorithms (Naive Bayes [76], ...) will be those that would learn the posterior probability distribution $P(\Theta \mid X = x)$. Meanwhile, *discriminative* algorithms (Support Vector Machines [77], Linear regression ...) simply learn the likelihood or conditional probability distribution $p(X \mid \Theta = \theta)$ to learn decision boundaries.

## 7.3 Graphical models

Graphical models (GM) are visual representations of the structure of the joint probability distribution that describe the interactions among the random variables in our model. Graphical models are designed to capture the most significant relationships among random variables in a simplistic and tractable manner. Therefore, it is important to note that graphical models capture reality with a certain degree of error due to those simplifications [78, 79, 80].
In a graphical model $G(V, E)$, the random variables are represented as nodes or vertices ($V$) whilst the interactions are described by the absence or presence of edges ($E$) among the nodes. The absence or presence of edges is directly correlated to the independence or dependence among the random variables. Graphical models allows statisticians to define general message-passing algorithms that implement probabilistic inference efficiently [78, 79, 80].

### 7.3.1 Directed Graphical Models: Bayesian Networks

Bayesian Networks are a type of Factor Graph [81] that contains directed edges (Directed Acyclic Graphs or DAG) which represent causality relationships between random variables in the form of conditional probability density functions. Bayesian Networks have received several names throughout history, such as probabilistic networks, causal networks, belief networks, or even knowledge maps. Bayesian Networks can capture the uncertainty over the probabilities of the random variables compensating for the simplifying nature of the graphical models [79, 82].

In order to understand the over-simplification that graphical models suffer when assuming full independence among the random variables recall that if,

a) The marginal probability of $B$ is $P(B)$ ,

b) The joint probability of $A$ and $B$ is $P(A \cap B) = P(A, B)$ ,

c) The conditional probability $P(A \mid B)$ is set as the ratio: $P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A,B)}{P(B)}$.

d) The *chain rule in probability theory* : $P(A, B) = P(A \mid B)P(B)$

We can conclude that the independency among random variables will factorize the joint probability into the product of their marginal probabilities and will eliminate the dependency among them as shown in the following equation,

$$P(A \mid B) = \frac{P(A)P(B)}{P(B)} = P(A)$$

This declaration will over-simplify the *chain rule in probability theory* by introducing independence among the variables in the model as $P(A, B) = P(A \mid B)P(B) = P(A)P(B)$. Although this assumption is very computationally efficient, complete independence among variables is not realistic, therefore conditional independence was introduced later to re-introduce some dependencies [82]. This states that if

a ) $A$ and $C$ are independent, $P(A \mid B, C) = P(A \mid B)$

b ) $A$ and $B$ are conditionally independent given another variable $C$,
$P(A, B \mid C) = P(A \mid C)P(B \mid C)$

This new perspective allows to express the joint probability function of a Bayesian Network as the

product among the *conditional probabilities* in the graph.

$$P(X_0, ..., X_n) = \overbrace{\prod_{n=1}^{N}}^{\text{Number of nodes}} p(X_n \mid \underbrace{X_{p_n}}_{\substack{\text{Set of parent} \\ \text{nodes of node n}}}) \tag{7}$$

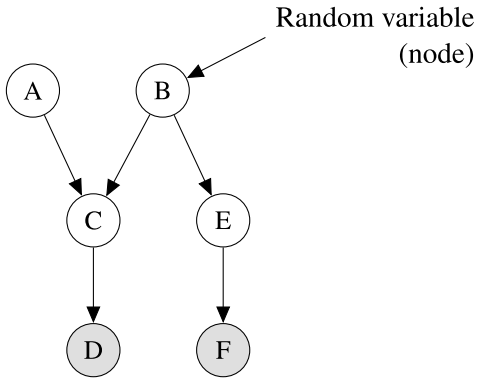



**Figure 4:** Bayesian Network example. Variables $A$ and $B$ are independent; $C$ is dependent on $A$ and $B$; $E$ is dependent on $B$; $D$ is dependent on $C$; $F$ is dependent on $E$; $D$ and $F$ are the observations and the rest of the variables are hidden. The joint probability is expressed as $P(A, B, C, D, E, F) = P(A)P(B)(C \mid A, B)P(D \mid C)P(E \mid B)P(F \mid E)$

### 7.3.2 Temporal Graphical Models: Dynamic Bayesian Networks

During the research presented here, I collaborated on the implementation of a Deep Markov Model (see 15), an advanced version of the classical Temporal Graphical models that I will now introduce. Temporal graphical models are a type of Bayesian Network that deal with uncertainty and change of the random variables through time. In this case, time is usually discretized, meaning that the model captures snapshots of the changes in the states of the model at specific times. The sequence of observations is considered as the collection of events that occurs at each time step [83].

**Markov Chain (MC):** Markov chains are a type of simple generative sequential models that describe a sequence of events (observations) throughout time. Contrary to the simplest sequential models that assume that the elements in the chain are independent and identically distributed, the MC introduces conditional dependencies to be able to model more realistic phenomena [51, 84].
First-order Markov Chains are sequences of random variables elements $(H_1, H_2, ...H_n)$ whose conditional probability distribution (transition probability distribution), for any element in the chain $(H_n)$, depends only on the previous element $(H_{n-1})$, this is known as the Markov property. The conditional dependency can be extended to $n$ previous states, for example, second-order MC or third-order MC. However, first-order MC has been shown to be enough to represent the desired relationships among the random variables and we can benefit from its computational simplicity [85].

In order to represent that the conditional dependency relies only on the previous element of the chain, we can see that the transition probability to the last element of the sequence reduces to $p(H_n \mid H_0, H_1, ..., H_{n-1}, H_n) = p(H_n \mid H_{n-1})$. Therefore, the joint probability of the Markov chain is equal to

$$p(H_0, ...H_n) = \prod_{n=0}^{N} p(H_n \mid H_0, H_1, ..., H_{n-1}, H_n) = p(H_0) \prod_{n=1}^{N} p(H_n \mid H_{n-1}) \tag{8}$$

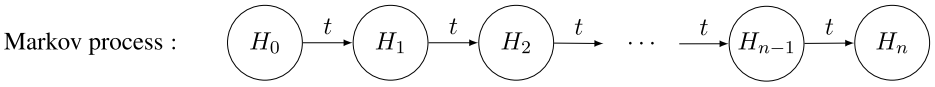



**Figure 5:** State machine representation of a Markov Chain. The transition probabilities between elements or states are represented as $t$.

**Hidden Markov Model:** Hidden Markov models are a type of sequential generative models in which we combine a sequential transition model for the hidden variables (Markov Chain) and independent sequential model for the observations. Both models intersect because each of the independent observations is conditionally dependent on a hidden state. This signifies that we can model the *transition probability* distribution between the hidden states and the conditional distribution of the observations given the hidden states (*the emission probabilities*) using the Markov property. HMM deals with discrete or continuous variables (hidden states and observations) and discretized time. Both HMM and MC assume a stationary process, implying that the transitions between states are static in time, meaning that they do not change or are deteriorated through time [86, 83, 87]. The joint probability of the chains expressed through the *chain rule of probability* is stated as,

$$p(\underbrace{H_0, ..., H_n}_{\text{Hidden states}}, \underbrace{X_0, ..., X_n}_{\text{Observations}}) = \underbrace{p(H_0)}_{\text{Init hidden state}} \prod_{n=0}^{N} \underbrace{p(H_n \mid H_{n-1})}_{\text{Transition probabilities}} \underbrace{p(X_n \mid H_n)}_{\text{Emission probabilities}} \tag{9}$$

where the initial hidden state of the chain is generally set to be equiprobable [83].

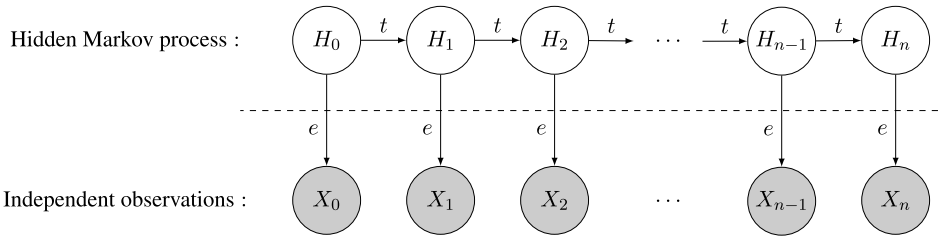



**Figure 6:** State machine of a Hidden Markov model. The hidden variables $(H_0, ...H_n)$ follow a Markov Chain and the observations $(X_0, ...X_n)$ are conditionally independent given the Markov Chain.

HMMs are a type of model on which we can perform Exact Bayesian Inference due to their typical small amount of hidden states. Inference over HMMs can be performed in several ways [86, 83]:

• Transition and emission probability matrices are known:

i) *Evaluation methods*: Estimate the probability of the sequence of observations given by the model parameters and sequence of hidden states. Typically they are dynamic programming-based methods, such as the *Forward algorithm* and the *Backward algorithm*. These procedures segment the calculation of the joint probability to take advantage of recursion. They reduce the complexity of the calculations from $\mathcal{O}(N^T T)$ to $\mathcal{O}(N^2 T)$, where $T$ is the number of observations [88].

ii) *Decoding methods*: Estimate the most likely sequence of hidden states given the sequence of observations and the model parameters ($\arg\max P(H_{0:n} \mid X_{0:n})$). This algorithm is referred as the Viterbi decoding method [89].

• Transition and emission probability matrices are not known:

i) *Training methods*: They estimate the model parameters $\theta$ given a model structure and sequences of observations.

⇒ Expectation-Maximization for HMM: Baum-Welch algorithm or Forward-Backward algorithm [90, 91, 92, 93, 94].

⇒ Viterbi extraction or training (unsupervised training), also named as the Baum-Viterbi algorithm [95, 96].

⇒ Maximum Likelihood Estimation (supervised training) [94].

⇒ Others : Markov Chain Monte Carlo methods (see Section 9.1.1), Stochastic Variational Inference (see Section 9.4).

# 8 Bayesian exact inference

Bayesian inference over graphical models seeks to estimate the posterior probability of the parameters $P(\Theta \mid X)$ to, for example, perform prediction. Bayesian prediction is used to compute several types of probabilities or *beliefs*:

i) *Marginal probabilities*: The marginal probability of an event calculates the probability of that event (random variable) occurring. We can calculate the marginal distribution of finite discrete events (random variables) by simply summing over every possible value of the parameter $\Theta$ and how it affects the observed data $X$. This is referred as the *marginal probability mass* and can be written as follows,

$$P(X) = \mathbb{E}[X, \Theta] \equiv \sum_{\Theta} P(X, \theta) \equiv \sum_{\Theta} P(X \mid \theta) P(\theta) \tag{10}$$

In the case that the random variables $X$ and $\Theta$ are continuous, meaning that they can take an infinite amount of values within some bounds, we marginalise by integration. This is referred as the *marginal probability density* and can be expressed as,

$$P(X) = \mathbb{E}[X, \Theta] \equiv \int_{\Theta} P(X, \theta)\, d\theta \equiv \int_{\Theta} P(X \mid \theta) P(\theta)\, d\theta \tag{11}$$

ii) *Posterior predictive distribution*: Estimates the probability distribution of a new data point $X^*$ marginalized over the inferred posterior $P(\Theta \mid X)$. These are the new predicted data points generated by the trained model.

$$P(X^* \mid X) = \int_{\Theta} P(X^* \mid \theta) P(\theta \mid X)\, d\theta \tag{12}$$

iii) *Prior predictive distribution*: Estimates the expected probability distribution of the data without conditioning on the observations. That is, the expected distribution of the data according solely to the model

$$P(X^*) = \int_{\Theta} P(X^* \mid \theta) P(\theta)\, d\theta \tag{13}$$

iv) *Expectations over parameters*: Inference of parameter values, for example,

$$\overline{\Theta} = \int_{\Theta} \theta P(\theta \mid X)\, d\theta \tag{14}$$

v) *Maximum a Posteriori* (MAP): Most likely value assignment of a variable.

$$\arg\max_{\theta^* \in \Theta} P(X, \theta^*) \tag{15}$$

Exact Bayesian inference can be attained in several manners, from the most rudimentary *brute force or enumeration algorithm*, via the slightly optimized *variable elimination algorithm* or the more advanced and efficient *belief propagation methods* [82]. The latter include well-known methods such us the *Forward-backward algorithm* (sum-product) [90, 91] or the *Viterbi decoding algorithm* (max-product) [89]. These methods allow for the exact computation of the joint probability for smaller models, with reduced amount of random variables and fewer dimensions. However, they quickly become infeasible for more complex and therefore realistic models [83, 97, 98].

# 9 Bayesian approximate inference

Performing Bayesian exact inference (analytical computations) quickly becomes intractable, especially due to the challenging computation of the normalization constant $Z$ [99]. This requires simplifying the calculations to Equation 6 and approximating the posterior distribution by estimating it from the joint probability described by the model. The most well-known methods are the stochastic simulations or Monte Carlo sampling methods and Variational Inference based methods. However, there are other types of algorithms, such as Expectation-Maximization [100], that approximate these probabilities for smaller models. Here, I describe the timeline of the predecessors of the No-U-Turns algorithm (NUTS) [4] and the Stochastic Variational Inference (SVI) [101] algorithms that have been used as the inference engines for the statistical models presented in this thesis. Concretely, NUTS is utilized in manuscript 14 and SVI in manuscripts 13, 15 and 16.

## 9.1 Monte Carlo sampling methods

Monte Carlo sampling methods are a series of techniques used to approximate certain characteristics of the entire population by taking a subset or sample. They were originated from ordinary card games and further developed for applications within the Physics field [102]. This approach allows to approximate an intractable sum or integral (i.e present in the joint, marginal or conditional probabilities), speed up the computation of tractable sums or integrals or approximate the probability density of the model and then impute missing data. There are several types of Monte Carlo sampling methods, for example, *Forward sampling*, *Rejection sampling*, *Importance sampling* or *Markov Chain Monte Carlo* methods (see Section 9.1.1) [103, 68].
*Forward or ancestral sampling* uses pre-defined probability tables for the statistical model and a probability density function $g(x)$ of choice to generate random values from (i.e Uniform distribution). The probability tables establish the joint and conditional probabilities between random variables in the model. Those tables also set the intervals that the random values from $g(x)$ will follow to be assigned to a category. The algorithm first samples a random number $u$ from $g(x)$ and then assigns it a value or category according to the probability intervals indicated in the tables [104].
In the case of *Rejection sampling* [105] we fix the values of one or more variables in the model (which are set as the evidence or observations). As we sample, if the sample is not consistent with the observations, we reject it. This method is more efficient than *Forward sampling* but might lead to high rejection of samples and computational waste. In addition to that, the sample probabilities are biased and do not add up to 1.
Lastly, *Importance or likelihood weighting sampling* [106]. This method ameliorates *Rejection sampling* by assigning probability weights to each of the samples aiming to correct their bias. The weights are given by the product among the probability values for the pre-fixed random variables of choice and the sampled probability values for the non-fixed variables [104].

**Monte Carlo estimate** Monte Carlo estimates allow the estimation of the expected values of the parameter $\Theta$ that describe the posterior probability distributions $P(\Theta \mid X)$ by simple summation instead of integration. The Monte Carlo estimate $I_n$ of a random variable $\Theta$, which is a collection of samples of $\theta_0, ..., \theta_n$, is equivalent to the true expectation of the random variable,

$$\overbrace{\mathbb{E}_{\theta_0,\ldots,\theta_n \sim \text{i.i.d} p(X,\Theta)}[I_n]}^{\text{Monte Carlo estimate}} \approx \overbrace{\mathbb{E}_{x \sim p(X,\theta)}[\theta]}^{\text{True expected value}} = \int \theta P(\theta \mid X)\, d\theta \tag{16}$$

The $N$ i.i.d samples from $P(\Theta \mid X)$ are generated to calculate an empirical estimate $\hat{\Theta}$ of the expected value of the parameter [107] as follows,

$$\mathbb{E}_{\theta_0,\ldots,\theta_n \sim \text{i.i.d} p(x,\Theta)}[I_n] = \frac{1}{N}\sum_{i=0}^{N} \theta_i = \hat{\Theta} \tag{17}$$

In probability theory, the distribution of the samples follows the Central Limit Theorem (CLT). It states that the distribution of a sample approximates a normal distribution as the sample size becomes larger, assuming that all samples are identical in size, regardless of the population's actual posterior distribution shape. This entails that the mean of the samples is normally distributed with a value equal to the expected value of the mean from the actual unknown posterior distribution of the random variable $\Theta$ [108]. The standard deviation calculated from the samples indicates the certainty that the mean of the samples (empirical estimate) is close to the actual expected value. The standard deviation of the samples is given by $\sigma = \sqrt{Var[\hat{\Theta}]} = \sqrt{\frac{\sigma^2}{N}}$, where N is the number of samples. Therefore as $N$ tends to infinite the uncertainty $\sigma$ decreases towards 0.

$$\text{Var}_{\theta_0,\ldots,\theta_n \sim \text{i.i.d} p(x,\Theta)}[I_n] = \text{Var}[\hat{\Theta}] = \frac{[(\Theta - \mathbb{E}[\Theta])^2]}{N} \tag{18}$$

### 9.1.1 Markov Chain Monte Carlo (MCMC)

MCMC is a type of Sequential Monte Carlo method (also known as Particle filters) where the target posterior distribution corresponds to the stationary distribution from the samples of a Markov Chain (see Section 7.3.2). The stationary distribution is unique if the Markov Chain is positive (the transition probabilities are positive), recurrent (the chain can always return to any state in a finite amount of time) and irreducible (we can continuously travel along the sequence of states, every state can be accessed from every other state). The posterior distribution over the optimizationoptimized parameters $\theta$, is calculated by using a prior distribution over the parameters $P(\theta)$ and a set of observations $X$ to calculate the likelihood given those parameters $P(X \mid \theta)$. The samples are taken by performing a *random walk* over the parameter space. This sampling procedure is "memoryless" since the next step in the sampling space is only performed based on the previous sample [103].
Metropolis Hastings [109, 110, 104], the Gibbs sampler [111] or the Bayesian Inference Using Gibbs Sampling (BUGS) [112] approaches are some of the most basic implementations of the MCMC sampler. Thereafter, to increase the computational efficiency of those methods, in terms of speed, automatization and accuracy, Hamiltonian Monte Carlo and shortly after No U-Turns Sampler (NUTS) were introduced [103].

### 9.1.2 Metropolis-Hastings (MH)

Metropolis-Hastings is a MCMC sampling method that counts with an acceptance/rejection criteria for the parameter's sample proposal [109, 110]. The algorithm requires the pre-selection of the proposal prior and likelihood's probability density function that will suit the observations. We select the

likelihood under which we assume our data ($X$) is distributed, meaning that we assume that each data point ($x_i$) is originated from a distribution function $f$ (PDF) with parameters $\theta$. For example, we can choose the univariate Normal distribution with parameters $\theta = \{\mu \in \mathbb{R}, \sigma \in \mathbb{R}^+\}$;

$$f(x_i|\theta) = f(x_i|\mu, \sigma) = \mathcal{N}(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \tag{19}$$

The prior will set the constraints over the values of the parameters that we will optimize. Considering this simple example, if we are optimizing the standard deviation $\sigma$ we have to reject the values lower or equal to 0. We can simply do so by assigning likelihood 0 for invalid values ($log(0) = -\infty$) (which stands for a highly negative contribution, therefore rejected) and likelihood 1 for valid values ($\log(1) = 0$).

We can compute the likelihood ($\mathcal{L}$) of the data under a selected probability density function by calculating the product of the likelihood of the proposed sampled parameters ($\theta$) under each data point ($x_i$) as follows,

$$\mathcal{L}(X|\theta) = \prod_i^n f(x_i|\theta) \tag{20}$$

However, for numerical stabilization, we will compute the log-likelihood instead and therefore rely on summation of the individual log-likelihoods instead,

$$\log \mathcal{L}(X|\theta) = \sum_i^n \log(f(x_i|\theta)) \tag{21}$$

In order to decide if we accept the parameter(s) sample proposals, we need to establish a acceptance/rejection criterion. This criterion says that, the sample will be accepted if the sum of the log-likelihood of the observations plus the log-likelihood of the prior(s) PDF(s) under the new parameter(s) proposal(s) ($\theta_{\text{new}}$) is higher than the one from the previously sampled parameter(s) ($\theta_{\text{current}}$). For example, we can define the acceptance criterion for a unique parameter $\theta$ as,

$$\sum_i^n \log(f(x_i|\theta_{\text{new}})) + \log(\pi(\theta_{\text{new}})) > \sum_i^n \log(f(x_i|\theta_{\text{current}})) + \log(\pi(\theta_{\text{current}})) \tag{22}$$

where $\pi(\theta)$ stands for the prior distribution function over the parameter $\theta$. Finally, to avoid massive rejection of samples and large computational time increase, we define a new acceptance criterion for the rejected samples from the first criterion. This second acceptance/rejection criterion says that we will also accept those samples whose difference in likelihood (not log-likelihood) between the $\theta_{\text{current}}$ and $\theta_{\text{new}}$, is higher than a random number sampled from a uniform distribution $\mathcal{U}(0, 1)$. The latter avoids massive rejection of samples [109, 110].

### 9.1.3 Hamiltonian Monte Carlo (HMC)

HMC [113] is an improved MCMC sampler that uses Newtonian principles to update the values of the sampled parameters by replacing the random Gaussian walk with a system of Hamiltonian equations. This approach avoids the issue of sampling highly correlated parameters and not exploring correctly

the probability space. The sampled parameters are encapsulated as a particle and assigned an energy value according to the Hamiltonian ($H(x, v)$). In the Hamiltonian system, $x$ is the position of the particle, used here as a synonym of the value of $x$ in the parameter space, and $v$ represents the velocity of the particle, which basically stands for the update of the values of $x$. The total energy of the particle can be represented as the sum of the kinetic ($\mathcal{K}(v)$) and the potential ($\mathcal{E}(v)$) energies, see Equation 23. The system seeks to minimize the potential energy of the particle and by doing so reach the regions with the highest probabilities. The kinetic energy allows us to explore different areas of the probability density without wandering around in local minima [113].

$$\mathcal{H}(x, v) = \mathcal{K}(v) + \mathcal{E}(x) \tag{23}$$

The potential energy of the particle can be interpreted as the negative log probability of the target proposal distribution. This inversion of the probability space sets the regions with high probabilities at the valleys in the new HMC probability space, see Figure 7.

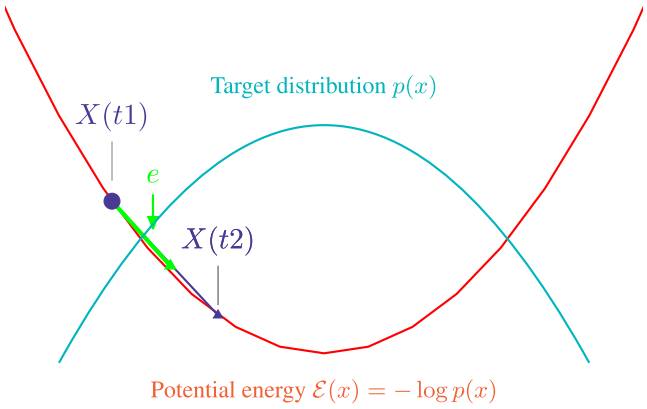



**Figure 7:** Visualization of the probability density landscape inversion applied to the target distribution by assuming Hamiltonian dynamics. A simple trajectory between time stamps $t1$ and $t2$ is marked. In each trajectory we propose a new sample every sub-step of size $\epsilon$. The trajectory shown in the plot contains only 2 step sizes, therefore we only integrate twice for this trajectory.

The kinetic energy of the particle represents the particle's change in velocity, also named *momentum*, this parameter establishes the range of change in the parameter values. The change in position with respect to the time is referred as the velocity of the system, it guarantees that the particle will also travel to low probability density regions, avoiding early convergence in the valleys of the HMC space. In the simplest case, to ease computations, we assume the mass of the particle to be equal to 1 (this value can be customized) and a friction-less system. The velocity of the system is initialized for every new particle trajectory to a value sampled from the normal distribution $v = \mathcal{N}(0, 1)$.

$$v = \frac{dx}{dt} = \frac{x(t2) - x(t1)}{\triangle t} \tag{24a}$$

$$a = \frac{dv}{dt} = \frac{v(t2) - v(t1)}{\triangle t} \tag{24b}$$

***The Leapfrog integrator***

For every trajectory of the particle in the HMC probability space, we require to calculate the new position and velocity of the particle. Unfortunately, we cannot calculate it exactly. We require an approximation method. Specifically, we require a method to integrate ordinary second-order differential equations, such as Newton's laws of motion that govern the energy of the particle. The motion equation of a particle is defined as follows,

$$m \overbrace{\frac{d^2x}{dt^2}}^{a} = \overbrace{-\nabla \mathcal{E}(x)}^{F} \tag{25}$$

where $m$ is the mass, $x$ is the position, $a$ is the acceleration, $v$ the is velocity and $F$ is the force applied to the particle, which in HMC is equal to the gradient of the potential energy.

The Leapfrog integrator is a type of symplectic integrator designed for this task. Due to its symplectomorphism, it preserves the energy of the system at any given point in time, in this case the energy of the particle defined by the Hamiltonian equations. The leapfrog integrator is capable of achieving a second-order accuracy since we approximate values of the motion properties of the particle (velocity, position and acceleration) using Taylor expansions that we truncate after the second-order summation term, assuming an error of $\mathcal{O}\Delta t^3$ at each step. This method is also attractive due to its time reversibility property that allows to calculate the integration backward in time for $k$ steps and allows to arrive back to the start of the trajectory [114].

For a given trajectory of the particle, between positions $x_0$ and $x_L$, with total length or number of steps/samples $L$, we have to update the position and velocity of the particle for $L/\epsilon$ times. We need to discretize the time steps along the trajectory length in intervals of $\Delta t$ or $\epsilon$ size. Subdividing the integration allows us to be able to calculate the total amount of particle position and velocity change. The Leapfrog integrator uses the *mid-point* method to reduce the error when approximating the integral, which simply states that the velocity is initialized half a step ahead of the position [114].

**Algorithm 1** The Leapfrog integrator

$x_0 \in R$ ▷ Initialize the position of the particle
$v_0 \sim N(0, 1)$ ▷ Add momentum to the particle
$v_n = v_0 \frac{1}{2}\Delta t F(x_n) = -v_0 \frac{1}{2}\Delta t \nabla E(x_n)$ ▷ Integrate the velocity of the particle half a step ahead
**for** step $\in L - 1$ **do** ▷ For each step
  $x_n = x_n + \Delta t v_{n+1/2}$ ▷ Update the position
  $v_n = v_n + \Delta t F(x_n) = v_n - \Delta t \nabla_{\mathcal{E}}(x_n)$ ▷ Update the velocity
$x_L = x_n + \Delta t v_n$ ▷ Position at the end of the trajectory
$v_L = v_n + \frac{1}{2}\Delta t F(x_n) = v_n - \frac{1}{2}\Delta t \nabla_{\mathcal{E}}(x_n)$ ▷ Velocity at the end of the trajectory

After each trajectory, the particle's positions (or sampled parameters values) are accepted or rejected following a similar approach to the Metropolis-Hastings criteria. In this case, we accept the new positions of the particles if their energy, computed using Equation 23, is lower than the one in the previous position. In order to avoid massive rejection of particle positions, we will also accept particles that have slightly higher energy if it's not above certain sampled random number.

### 9.1.4 No U-Turn Sampler (NUTS)

HMC is a great MCMC sampler, however, it is highly sensitive to the choice of the step size $\epsilon$ and the trajectory length $L$. NUTS is a type of HMC sampler where the choice of $\epsilon$ and $L$ is made automatically [4]. NUTS defines a stop criterion to cease the trajectory length to keep growing (adding step sizes). The algorithm detects when the position of the particle at the beginning of the trajectory (initial parameters), $x_0$, and the current estimated position of the particle (current new sampled parameters), $x_n$ are getting close, meaning that the trajectory is looping onto itself and sampling around the same probability region where it started.
The algorithm takes advantage of the aforementioned time reversibility of the Leapfrog integrator to be able to extend the trajectory of the sampling chain not only forward but also backward. As the chain of samples grows in both directions, it keeps track of the order of the samples via a balanced binary tree. A balanced binary tree is one where right and left sub-trees always have the same number of child nodes, therefore they have to be filled up in a doubling manner (the next step will have twice the amount of samples (child nodes)). The tree is filled either from the right (forward in time) or from the left (backward in time) according to a uniform distribution that selects the direction at each step [4].

The growth of the trajectory, aka the binary tree, stops when the trajectory is looping onto itself. This can be detected by performing the vector dot product between 2 vectors, the distance travelled by the particle (amount of change in the parameters), and the current momentum of the particle (how fast the parameter values are changing), see Figure 8.

$$\begin{matrix}\text{stop} \\ (\mathrm{x}_n - x_{n-k}).v_n^T\end{matrix} == \begin{cases} s \leqslant 0; \text{stop chain (angle is equal or less than } 90^\circ) \\ s > 1; \text{continue sampling (angle is more than } 90^\circ) \end{cases} \tag{26}$$

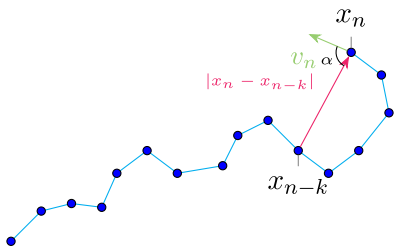



**Figure 8:** Example of stopping criteria in NUTS. Representation of a sampling sub-trajectory where the algorithm encounters a U-turn, entailing that the angle $\alpha$ between the dot product of the vector $|x_n - x_{n-k}|$ and $v_n$ is smaller than $90^\circ$. $k$ indicates the number steps (samples) behind the current one.

The algorithm is refined by applying the stop criterion along sub-trajectories ($x_n \rightarrow x_{n-k}$) of the main trajectory (where $k$ is the number of backward steps that we perform in the chain), therefore performing an early detection of the loops. The evaluation of the sub-trajectories is only performed on the children of the same sub-tree not across sub-trees [4].

## 9.2 Variational Calculus

Before we proceed to explain the next group of methods to approximate Bayesian inference we need an overview of Variational Calculus which will help us solve the upcoming *variational problems* through Variational Inference. The calculus of variations or variational calculus implies studying how infinitesimal variations of a path (or function) change an integral. This is used for solving optimization problems by finding the minimum or the maximum of such integral [115].
The *Euler-Lagrange equation* is frequently used to solve those variational problems and find the parameters that minimize the integral. The Lagrangian equation relates to classical mechanics as the equation that characterizes the state of motion of a system (i.e a particle) at any particular point in time, which is described by the potential and kinetic energies. In Lagrangian mechanics, unlike Hamiltonian's the total energy of the system is portrayed as the difference in energies, whilst in the Hamiltonian is the sum of energies (see Section 9.1.3). This means that within Lagrangian mechanics the energy changes with time (the potential energy is converted into kinetic energy and vice versa) instead of remaining constant [115].

$$\overbrace{\frac{\partial f}{\partial y}}^{\text{Kinetic energy}} - \overbrace{\frac{d}{dx}\frac{\partial f}{\partial y'}}^{\text{Potential energy}} = 0 \tag{27}$$

The rate of change of the potential energy into the kinetic energy (or vice versa) is given by the Lagrangian equation. Lagrangian mechanics also state *the principle of stationary action* where the Lagrangian equation is set to 0, see Equation 27. This is the case because the optimal path through space and time from point A to point B is that one where the variance in the path is close to 0. The derivative of the function of the path is close to 0 and therefore it is a straight path with no changes in the slope [115].

## 9.3 Variational Inference (VI)

MCMC sampling-based methods quickly become computationally expensive as the number of observations increased, it was hard to assess their convergence and struggled with complex distributions. Variational Inference (VI) arose as an alternative algorithm for approximating the posterior probability distributions.
The concept behind Variational Inference methods is to propose a family of densities $Q$. (i.e a Normal distribution that can take any values for $\mu_\phi$ and $\sigma_\phi$) and find a member $q_\phi(z \mid x)$ (i.e a Normal distribution with specific values for $\mu_\phi$ and $\sigma_\phi$) of that family which is close to the target posterior distribution $p(z|x)$, which minimizes the KL divergence [99, 116] (see Section 9.3.1).

$$q^*_\phi(z \mid x) = \arg\min_{q \in Q} KL(q(z \mid x)||p(z \mid x)) \tag{28}$$

A single member of $Q$ is referred as a variational distribution $q(z \mid x)$ from which we sample the latent random variable $z$ (i.e $\mu_z$). The variational distribution is parameterized by the variational factors $\phi$, represented as $q_\phi(z \mid x)$. The variational factors $\phi$ (referred also as variational or optimizable parameters) allow $q$ to acquire different densities shapes, see Figure 11.

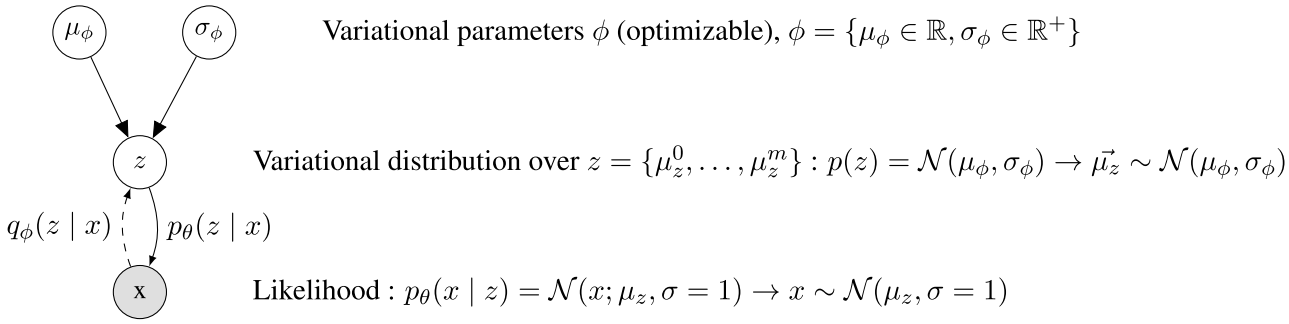



**Figure 9:** Example of variational inference over a simple graphical model. The observations $x$ are dependent on a single latent variable $z$ sampled from a variational distribution $p(z)$ parameterized by the variational factors $\phi$.

### 9.3.1 Kullback-Leibler divergence

Kullback-Leibler KL divergence [117, 118], also referred as *relative entropy*, is the measurement of the difference between two distributions at any point of the x-axis in the log-scale. The KL divergence has several fundamental properties, among them:

- KL is positive ($KL > 0$) and zero-prone, the distance tends to 0 as $q$ and $p$ are more similar.
- Strictly convex in $p$ for a fixed $q$.
- Non symmetrical. The non symmetrical property signifies that $KL(q(z)||p(z \mid x))! = KL(p(z \mid x)||q(z \mid x))$, which in this case translates to $KL(q(z \mid x)||p_\theta(z|x)) = -KL(p(z|x)||q(z))$

**Table 1:** Comparison of Forward and Reverse KL divergences. Reverse KL is the preferred option for optimization problems.

| Forward KL | Reverse KL |
|---|---|
| $KL(p(z \mid x)\|\|q(z \mid x))$ | $KL(q(z \mid x)\|\|p(z \mid x))$ |
| • Mean seeking or inclusive, finds a $q$ that captures all modes in $p$ by averaging the modes in $p$ | • Mode seeking or exclusive, finds a $q$ that best fits a mode (peak) of the $p$ distribution shape, which is not ideal for multi-modal $p$ |
| • Requires normalization w.r.t $p$ , which is often not computationally convenient. | • Does not require normalization w.r.t $p$ , which is computationally convenient. |

**Figure 10:** (*Left*) Forward KL: Mean seeking or inclusive (*Right*) Reverse KL: Mode seeking or exclusive.

The general formula for the Reverse KL can be stated as the expectation of the log-ratio between the variational distribution and the posterior distribution,

$$KL(q(z \mid x)||p(z|x)) = \mathbb{E}\left[\log \frac{q(z \mid x)}{p(z \mid x)}\right] \tag{29}$$

The expected value of KL divergence can be solved both for discrete probability distributions (via summation, see equation 9.3.1) and continuous probability distributions (via integration, see equation 9.3.1)

$$KL(q(z \mid x)||p(z|x)) = \sum_{x \epsilon X} q(z \mid x) log \frac{q(z|x)}{p(z|x)} \tag{30}$$

$$KL(q(z \mid x)||p(z \mid x)) = \int_{-\infty}^{\infty} q(z \mid x) log \frac{q(z|x)}{p(z|x)} \tag{31}$$

### 9.3.2 Evidence Lower Bound (ELBO)

The KL divergence, however, is not an appropriate optimization objective, since in order to calculate it we require $p(z|x)$ which is actually what we are trying to estimate. However, we can rearrange the KL divergence equation in order to figure out a lower bound on the log marginal probability $p(x)$ so that

it is maximized. The approach to find such lower bound, namely the Evidence Lower Bound (ELBO) [119, 120, 121], is as follows:

$$
\begin{aligned}
&KL(q(z \mid x)||p(z|x)) = \\
&\quad = \int_{-\infty}^{\infty} q(z \mid x) \log \frac{q(z \mid x)}{p(z \mid x)}\, d(z) \\
&\quad = \int_{-\infty}^{\infty} q(z \mid x) \log \frac{q(z \mid x)p(x)}{p(z \mid x)p(x)}\, d(z) && \text{Multiply and divide by } p(x), \\
&\quad = \int_{-\infty}^{\infty} q(z \mid x) \log \frac{q(z \mid x)p(x)}{p(x,z)}\, d(z) && \text{Using Bayes's theorem 5 substitute } p(z \mid x)p(x) \text{ by } p(x,z), \\
&\quad = \log p(x) + \int q(z \mid x) \log \frac{q(z \mid x)}{p(x,z)}\, d(z) && \text{Transform to log probabilities and extract the log probability of the data from the integral, since } \log p(x) \text{ is a constant with respect to } z. \\
&\quad = \log p(x) + \int q(z \mid x) \log q(z \mid x)\, dz - \int q(z \mid x) \log p(x,z)\, dz \\
&\quad = \log p(x) - \int q(z \mid x)[\log p(x,z) - \log q(z \mid x)]\, dz \\
&\quad = \log p(x) - \mathbb{E}_{z\sim q(z|x)}[\log p(x,z) - \log q(z \mid x)] && \text{Re-write it in the form of an expectation} \\
&\quad = \log p(x) - \underbrace{(\mathbb{E}_{z\sim q(z|x)}[\log p(x,z)] - \mathbb{E}_{z\sim q(z|x)}[\log q(z \mid x)])}_{\text{ELBO}(\mathcal{L}(Q))}
\end{aligned}
\tag{32}
$$

Maximizing the ELBO can be turned into our new optimization objective since as we can see in equation 32 when the ELBO is higher the KL divergence becomes smaller and we are converging towards a better approximation of the posterior. It is important to note that this is one popular way of calculating the KL divergence and the ELBO, however, there are several different options to do so and, that gives place to a wide range of optimization objectives.

$$
q^*_\phi(z \mid x) = \arg\max_{q^* \in Q} \mathcal{L}(q_\phi(z \mid x)) \tag{33}
$$

### 9.3.3 Mean-Field Approximation (MFA) assumption to solve VI

In order to solve the variational distribution $q(z \mid x)$ we have to make use of the mean-field assumption. The mean-field assumption states that the family of distributions $Q$ can be stated as the factorization or product of the $q$ family members, where each of the members is characterized with a different functional form due to being associated with a different set of $\phi$ parameter values (variational factors). Given this assumption, we establish independence among the possible functional forms of $q$ and therefore we can apply the *chain rule of probability* shown in Equation 34 and factorize $Q$ into the product of their marginal distributions [122, 123].

$$Q = q_\phi(z \mid x) = \prod_i q_\phi(z_i \mid x) \tag{34}$$

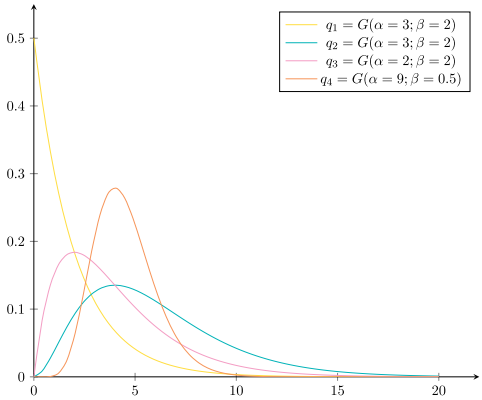



**Figure 11:** Subset of $q_n$ distribution functions from the Gamma family ($Q$) of distributions

### 9.3.4 Solving Variational Inference via Coordinate Ascent (CAVI)

Optimizing the variational approximation requires maximizing the ELBO, by doing so we require to find the optimal variational factors for the chosen variational distribution. We can use the simile of the Lagrangian mechanics, see section 9.2, and treat the ELBO equation as the expression of the energy of our system.
i) The ELBO formula from equation 32 can be expressed as the following (for $n$ data points and $m$ variational factors) [123],

$$\overbrace{\mathcal{L}(Q)}^{\textbf{a}} = \overbrace{\mathbb{E}_q[\log p(x_{1:n}, z_{1:m})]}^{\textbf{b}} - \overbrace{\mathbb{E}_q[\log q(z_{1:m} \mid x_{1:n})]}^{\textbf{c}} = \mathbb{E}_q\left[\log p(x, z) - \sum_{j=1}^{m} \log q_j(z_j \mid x)\right] \tag{35}$$

where a, b and c stand for:

**a** : Evidence Lower Bound or Negative Variational free energy

**b** : Expected complete log joint probability of the model for $m$ variable parameters. Sets higher probability to the $z$ that best explains the observed data, converges towards a MAP estimate of $z$.

**c** : Log probability of the variational distribution. "Negative Entropy" or regularization factor that diffuses the MAP estimate.

ii) The maximization of the ELBO occurs at each training step over a single $q_i$ and not over the rest of distributions $q_{j\neq i}$, therefore we can split the expectation.

$$\mathcal{L}(Q) = \mathbb{E}_{q_i}\left[\mathbb{E}_{q_{j\neq i}}\left[\log p(x, z) - \sum_{j=1}^{m} \log q_j(z_j \mid x)\right]\right] \tag{36}$$

iii) We can now extract the log probability of the fixed $q_i$ from the summatory. Re-arrange the expectations to isolate the terms that do not depend on $q_i$.

$$\begin{aligned}\mathcal{L}(Q) &= \mathbb{E}_{q_i}\left[\mathbb{E}_{q_{j\neq i}}\left[\log p(x,z) - \log q_i(z_i \mid x) - \sum_{j\neq i}^{m} \log q_j(z_j \mid x)\right]\right] \\ &= \mathbb{E}_{q_i}\left[\mathbb{E}_{q_{j\neq i}}\left[\log p(x,z) - \log q_i(z_i \mid x)\right] - \sum_{j\neq i}^{m} \mathbb{E}_{q_{j\neq i}}[\log q_j(z_j \mid x)]\right] \\ &= \mathbb{E}_{q_i}[\mathbb{E}_{q_{j\neq i}}\left[\log p(x,z) - \log q_i(z_i \mid x)\right] - \sum_{j\neq i}^{m} \mathbb{E}_{q_{j\neq i}}[\log q_j(z_j \mid x)]\end{aligned} \tag{37}$$

iv) The left term can be re-arranged as a negative KL divergence. The right term is a constant since we only update $q_i$ while keeping the rest fixed.

$$\begin{aligned}\mathcal{L}(Q) &= -\mathbb{E}_{q_i}[\log q_i(z_i \mid x)] - \mathbb{E}_{q\neq i}[\log p(x,z)] - cte \\ &-\mathbb{E}_{q_i}[\log q_i(z_i \mid x)] - \log e^{\mathbb{E}_{q_{j\neq i}}[\log p(x,z)]} - cte \\ &-\mathbb{E}_{q_i}\left[\log \frac{q_i(z_i \mid x)}{e^{\mathbb{E}_{q_{j\neq i}}[\log p(x,z)]}}\right] - cte \\ &-\mathcal{KL}\left(q_i(z_i \mid x) || e^{\mathbb{E}_{q_{j\neq i}}[\log p(x,z)]}\right) - cte\end{aligned} \tag{38}$$

v) Therefore in order to maximize the ELBO with respect to $q_i$, we need to minimize the KL term. We can solve the optimization of the ELBO by making use of *Euler-Lagrange equation*, see Equation 27 and set the partial derivative to 0.

$$\frac{d\mathcal{L}(Q)}{dq(z_i \mid x)} = -\mathbb{E}_{q_i}[\log q_i(z_i \mid x)] - \log e^{\mathbb{E}_{q\neq i}[\log p(x,z)]} - cte = 0 \tag{39}$$

vi) The above equation resolves and the KL divergence is minimized and converges when $q_i$ is equal to,

$$\begin{aligned}q_i^*(z_i \mid x) &\propto e^{\mathbb{E}_{q_{j\neq i}}[\log p(z_i | z_{j\neq i}, x)]} \\ &\propto e^{\mathbb{E}_{q_{j\neq i}}[\log p(z_i, z_{j\neq j}, x)]} \\ &\propto e^{\mathbb{E}_{q_{i\neq j}}[\log p(z, x)]}\end{aligned} \tag{40}$$

CAVI displays several disadvantages, i) it assumes complete independence across the random variables in the model and their components ii) the parameter update (variational factors) requires a manual derivation of the computation of the expected value and variation iii) data mini-batching is not achievable, requires observing the entire dataset to perform a parameter update [99].

## 9.4 Stochastic Variational Inference

The work presented in this manuscript has primarily employed Stochastic Variational Inference (SVI) [101] as the inference engine, see papers 13, 15 and 16. To introduce the concept of SVI, first, we need to define Stochastic Gradient descent [124] as the method that turns Variational Inference (VI) into a practical and generalized approach.

### 9.4.1 Gradient Descent

Gradient descent is an optimization algorithm that seeks to determine the set of parameter values (i.e weights and biases values in a Neural Network) that minimize the target function (i.e error loss function, such as Mean Squared Error, ELBO, cross-entropy ...). This method is a combination of the forward and backward propagation algorithms. The 2 steps are illustrated in Sections 9.4.2 and 9.4.3 with a small example [125, 126].

### 9.4.2 Forward-propagation

Forward-propagation consists of a series of functional transformations (linear or non-linear) that the input data (i.e matrix or vector or scalar of features) goes through when given to a Feed-forward Network. As a result, the input data is then transformed into a prediction (i.e class type, continuous representation of an angle ...) whose validity can be evaluated through an error loss function. Recall that linear functions in Machine Learning are simply the classical linear regression architecture $y = mx + b$ but instead, we have replaced the coefficient $m$ with a weight matrix $W$ and we also use a bias vector $B$ [127].

The following formulations are the equivalent representation of the functions shown in Figure 12. We have a 4 dimensional input vector and scalar output. The cost function is computed within the predicted output $\hat{y}$ and the target $y$ (i.e labels). Vectors and matrices are symbolized in capital letters, $y$ refers to the true value of the prediction.

$$
\begin{aligned}
X &= \mathbb{R}^4 \\
H^1 &= W^1 X + B^1 \\
H^2 &= W^2 H^1 + B^2 \\
h^3 &= W^3 H^2 + b^3 \\
\text{Sigmoid activation: } \rightarrow \hat{y} &= h^* = \sigma(h^3) = \frac{1}{1 + e^{-h^3}} \\
\text{Mean-squared error } \rightarrow \mathcal{L} &= \frac{1}{2}(y - \hat{y})^2
\end{aligned}
\tag{41}
$$

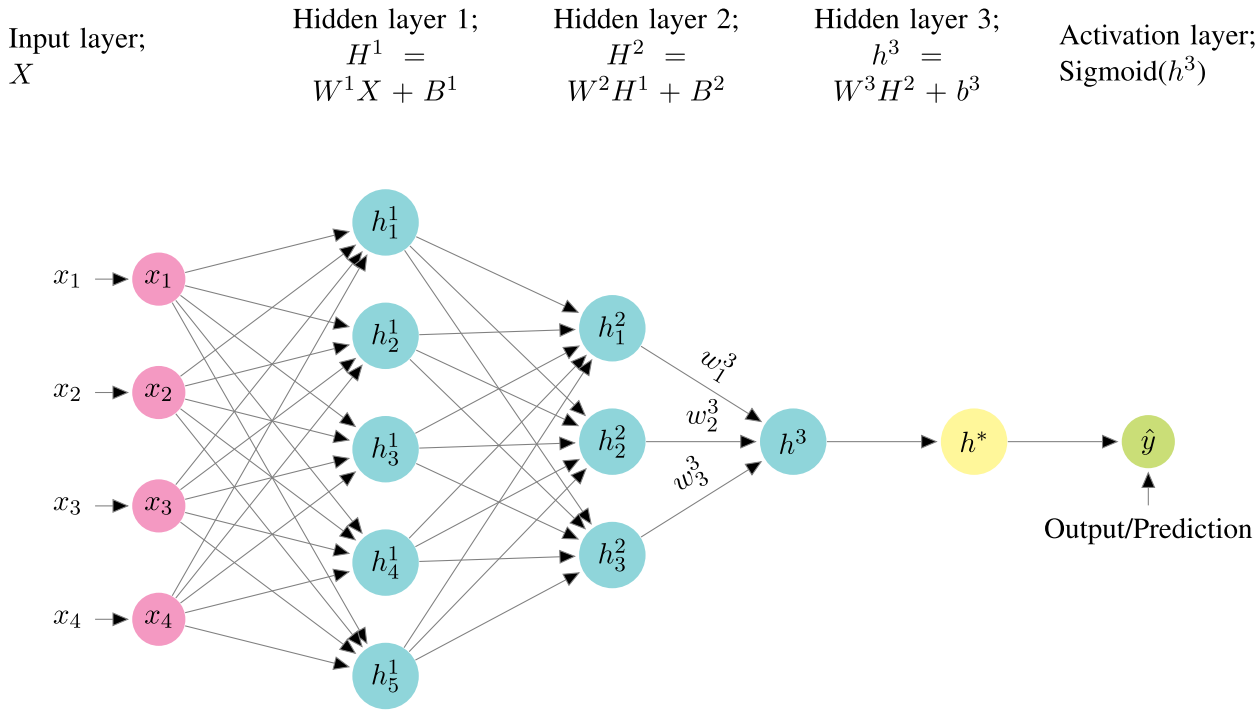



**Figure 12:** 4-Layer *Feed-forward* network with a series of 3 linear transformations and 1 activation or non-linear function. There are 4 input data points or feature vectors that are transformed into 1 output scalar that will be evaluated with an error loss function.

### 9.4.3 Backpropagation

Backpropagation [128, 129] computes the gradient of the network and then adjusts the parameters of the model to decrease the loss of the network between the resulting predictions and real values, which we denote as the *backpropagate* and *parameter update* steps, respectively. This process continues iteratively until the loss is less than given bound [128, 129].

i) *Backpropagate*: The gradient of the network is computed by computing partial derivatives for each layer of the network and composing them via the chain rule; we illustrate this process for $w_1^3$ parameter in the the 4-layer network illustrated in Figure 12.

a) Calculate the partial derivative of the loss function, in this case, the mean-squared error, with respect to the predicted outputs $\hat{y}$ and the real values $y$:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y \tag{42}$$

b) Calculate the partial derivative of the prediction $\hat{y}$ with respect to activation $h^*$:

$$\frac{\partial \hat{y}}{\partial h^*} = \sigma(h^3) * (1 - \sigma(h^3)) = \hat{y} * (1 - \hat{y}) \tag{43}$$

c) Calculate the partial derivative of the hidden layer $y^3$ with respect to the weights of interest $w_1^3$

$$\frac{\partial y^3}{w_1^3} = \frac{\partial(h_1^2 w_1^3 + h_2^2 w_2^3 + h_3^2 w_3^3)}{\partial w_1^3} = h_1^2 \tag{44}$$

d) Combine the above partial derivatives to estimate the gradient of the loss function with respect to the optimized parameter $w_1^3$.

$$\frac{\partial \mathcal{L}}{\partial w_1^3} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h^*} \frac{\partial y^3}{w_1^3} \tag{45}$$

ii) *Parameter update*: Update each of the parameters $\theta$ in the network $\theta \in \{W_1, B_1, W_2, B_2, W_3, B_3\}$. The general equation for parameter update, where $\nabla\theta$ is the gradient and $\eta$ is the learning rate is stated as,

$$\theta = \theta - \eta\nabla\theta \begin{cases} \nabla\theta = 0; \text{Stationary/Saddle point} \\ \nabla\theta < 0; \text{Move forward in the gradient (negative/downhill slope)} \\ \nabla\theta > 0; \text{Move backwards in the gradient (positive/uphill slope)} \end{cases}$$

Example of parameter update:

$$W^3 = W^3 - \eta \frac{\partial \mathcal{L}}{\partial W^3}$$

### 9.4.4 Variations of Gradient Descent

The are several methods of Gradient Descent that differ on how the data is divided and when the backpropagation computations occur [124, 130].

**1) Batch Gradient Descent (BGD):** The training set is divided into batches. First, the forward algorithm is run through every single batch in the data set, without calculating the gradients. After all the batches have been through the forward-network, then we can compute the backpropagation algorithm over an average prediction across all batches in the data set. This approach means that if we have very larger datasets we will not be able to update the parameters until all the gradients across all batches have been updated. Consequently this design is not very efficient [124, 130].

**2) Stochastic Gradient Descent (SGD):** SGD scales up to larger datasets by taking as input one training data point at the time. We supply the training data point to the feed forward network, calculate its gradient and then update it. This approach produces a fluctuating gradient descent and tends not to completely converge due to its unstable character. However, SGD can be used for larger datasets since it can converge faster due to the more frequent updates of the parameters. Unfortunately, the sequential nature of this approach makes it impossible to parallelize the computations, slowing down the method [124].

**3) Mini batch Gradient Descent (SGD):** The training data set is also split into what is called "mini-batches". The change in name from batch to mini-batch relies solely on that now we will input the mini-batch to the network, calculate its gradient, and then update the parameter. On this occasion, we do not hold the gradient computation until all batches have run through the feed-forward network. This approach also generates a fluctuating gradient descent, nonetheless, we can benefit from vectorization and scalability to larger datasets [124, 130].

**4) Optimizers: Upgrading Vanilla Gradient Descent** Optimizers are different versions of the classical or Vanilla Gradient Descent described above. They seek to boost the exploration of the gradient landscape by making smarter decisions on how to update the parameters and consequently avoiding local minima. Among the most well-known methods we find the first-order gradient-based optimizer Adam [131] which benefits from combining the AdaGrad [132] and RMSProp [133] optimizers. During the presented research I have used Adam its different deviations as my optimizers of choice.
AdaGrad uses a momentum vector $m_t$, the *exponentially weighted average of the gradients*, to correct the learning rate and the parameter update (i.e $W_1$) as follows,

$$\begin{gathered}\overbrace{m_t}^{\text{Momentum vector 1}} = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\partial \mathcal{L}}{\partial W_t}\right] \\ \overbrace{w_{t+1} = w_t - \alpha m_t}^{\text{Weight parameter update in AdaGrad}}\end{gathered} \tag{46}$$

where $\beta_1$ is a constant representing the exponetial decay or moving average parameters. RMSProp uses a second momentum vector, the *exponential moving average*, to correct the paramater update as,

$$\begin{gathered}\overbrace{v_t}^{\text{Momentum vector 2}} = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\partial \mathcal{L}}{\partial W_t}\right]^2 \\ \overbrace{w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\epsilon)^{\frac{1}{2}}}\left[\frac{\partial \mathcal{L}}{\partial w_t}\right]}^{\text{Weight parameter update in RMSProp}}\end{gathered} \tag{47}$$

where $\epsilon$ is a small positive constant and $\beta_2$ is another constant representing the moving average parameter. Adam combines both momentum vectors using a biased corrected version,

$$\begin{aligned}\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}\end{aligned} \tag{48}$$

Finally, the weight parameter in Adam is updated using a combination of both unbiased momentum vectors,

$$w_{t+1} = w_t - \hat{m}_t\left(\frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon}\right) \tag{49}$$

### 9.4.5 Black Box Variational Inference (BBVI): Optimizing the ELBO by Gradient Ascent

We can make use of Gradient Descent to maximize the ELBO, by setting the optimization objective to - ELBO. On this occasion, the variational parameter update, instead of being given by the Coordinate Ascent function, will be performed via gradient estimation. We have discarded CAVI, see Section 9.3.4, since determining the closed-form expressions of the density functions can become a complex error-prone problem to solve depending on the chosen variational distribution for the parameters. Furthermore, we seek to automatize inference to deal with any type of model, to handle large datasets, and reduce the possibilities of errors during inference [123, 134]. To do so, we then request to fulfill the following points [101],

1. Re-write the gradient of the ELBO as an expectation with respect to the (log) probability of the variational distribution $q$ . See Equations 51 or 52.
2. Sample $z$ from the variational distribution $q$ to calculate a Monte Carlo unbiased estimator of the ELBO's gradient (weighted sum among the samples). This estimate is regarded as the expected value of the gradient.
3. Use the Monte Carlo unbiased ELBO's gradient estimator of the sample $z$ to update the variational parameters $\phi$. This update of the variational parameter is stochastic due to the noisiness of ELBO's gradient. The general rule for variational parameter update is the following,

$$\phi_{\text{new}} = \phi_{\text{old}} + \eta \nabla_{\phi} \mathcal{L}(\phi_{\text{old}}) \tag{50}$$

**Solving BBVI via the "score gradient"** The "score gradient" or likelihood ratio or reinforce gradient [135, 136, 134] fulfils our first requirement by rewriting the gradient of the ELBO as the following,

$$\overbrace{\nabla_{\phi} \mathcal{L}}^{\text{Score gradient}} = \mathbb{E}_{q_{\phi}(z|x)}[\overbrace{\nabla_{\phi} \log q_{\phi}(z \mid x)}^{\text{Score function}} \overbrace{(\log p(x, z) - \log q_{\phi}(z \mid x))}^{\text{Instantaneous ELBO}}] \tag{51}$$

The score function can be calculated as the partial derivative of the log-likelihood of the variational distribution with respect to each of its variational parameters $\phi$, as $\frac{\partial \log q_{\phi}(z)}{\partial \phi}$

**Solving BBVI via the "reparametrization of the gradient" or "reparameterization trick"** The "Score gradient" has a problem of high variance on the estimation of the gradient, which might update the variational parameters to a non-optimal region. To control this issue, several approaches have been formulated, for example, the Rao-Blackwellization (computes the conditional expectation with respect to one of the variables) [137] or the use of control-variates (penalization technique over the expectation of the gradient via a function and scalar of choice)[138]. However, the formulation of the "reparametrization trick" [139, 140, 141, 142] approached the issue differently and it has become the most generalized method.

The trick is to express the variational distribution $q_{\phi_j}(z)$ using a transformed variational distribution $T(\epsilon_i, \phi_j)$.

$$\nabla_{\phi} \mathcal{L} = \mathbb{E}_{S(\epsilon)}[\overbrace{\nabla_{\theta}[\log p_{\theta}(x, z) - \log q_{\phi}(z \mid x)]}^{\text{Gradient of Instantaneous ELBO}} \overbrace{\nabla_{\phi} T_{\epsilon}(\phi)}^{\text{Gradient of the transformed parameter}}] \tag{52}$$

**Algorithm 2** Optimization via the "score gradient" [135, 136, 134] of a single variational parameter $\phi$.

$\phi_0 = -10$ ▷ Initialize randomly the variational parameter to optimize $\phi$
$\eta = 1e-3$ ▷ Set a learning rate
**while** ELBO not converged **do**
  $\hat{g}_j = 0$ ▷ For this current epoch initialize the gradient to 0
  **for** i ∈ S **do** ▷ Take S samples from the variational distribution $q_{\phi_j}(z)$
    $z_i \sim q_{\phi_j}(z \mid x)$
    $\hat{g}_j \mathrel{+}= \nabla_\phi \log q_{\phi_j}(z_i \mid x)(\log p(x, z_i) - \log q_{\phi_j}(z_i \mid x))$ ▷ Accumulate the gradients for every sample of $z$
  $\hat{g}_j = \frac{\hat{g}_j}{S}$ ▷ Normalize the gradient by the number of samples taken of $z$ from $q_{\phi_j}(z \mid x)$ for the current $\phi$
  $\phi_{j+1} = \phi_j + \eta \hat{g}_j$ ▷ Update the variational parameter
  $j \mathrel{+}= 1$ ▷ Add one epoch

We cannot optimize over a sampled random variable $z$. Instead, now we sample an auxiliary variable $\epsilon$ from the distribution $d(\epsilon)$ that helps "sampling" indirectly the parameter $z$ given $\epsilon$ and $\phi$. For the normal distribution the transformation is linear, $z = T(\epsilon, \phi) = \epsilon\sigma + \mu$; where $z$ is now equal to a mean $\mu$ plus a weighted variance $\sigma$. Recall that $\mu$ and $\sigma$ are the variational parameters $\phi$ that become updated every epoch. Using this trick, we now control the variance over the parameters of the model and the gradient with an auxiliary variable $\epsilon$ [139, 140, 141, 142].

**Algorithm 3** Optimization via the "reparameterization trick" [139, 140, 141, 142] of a single variational parameter $\phi$.

$\phi_0 = -10$ ▷ Initialize randomly the variational parameter to optimize $\phi$
$\eta = 1e-3$ ▷ Set a learning rate
**while** ELBO not converged **do**
  $\hat{g}_j = 0$ ▷ For this current epoch initialize the gradient to 0
  **for** i ∈ S **do**
    $\epsilon_i \sim d(\epsilon)$ ▷ Take S samples from the auxiliary variable $\epsilon$ from its distribution $d$
    $z_i = T(\epsilon_i, \phi_j)$ ▷ $z$ is NOT sampled, is the product of a transformation of the auxiliary variable $\epsilon$ and the current $\phi$
    $\hat{g}_j \mathrel{+}= \nabla_\theta[\log p(x, z_i) - \log q_{\phi_j}(z_i \mid x)]\nabla_{\phi_j} z_i$ ▷ Accumulate the gradients for every value of $z$
  $\hat{g}_j = \frac{\hat{g}_j}{S}$ ▷ Normalize the gradient by the number of samples taken of $\epsilon$ for $T(\epsilon, \phi)$ for the current $\phi$
  $\phi_{j+1} = \phi_j + \eta \hat{g}_j$ ▷ Update the variational parameter
  $j \mathrel{+}= 1$ ▷ Add one epoch

**Table 2** Comparison of the BBVI methods

| "Score gradient" | "Reparameterization trick" |
|---|---|
| Valid for discrete and continuous models. | Requires differentiable (non-discrete) variables. |
| Works with a large class of variational approximations. | Requires the variational approximation to be transformable. |
| The variance of the noisy gradient can be large. | The variance of the gradient is well behaved. |
| | Requires differentiable models with respect to the parameters $z$. |

### 9.4.6 Stochastic Variational Inference

Stochastic Variational Inference or Stochastic Gradient Variational Bayes [101] is a generalizable inference framework to perform approximated Bayesian inference. It benefits from the combination of Automated Variational Inference [134], Black Box Variational inference [123], mini-batching and Stochastic Gradient Descent [124].

# 10 Neural Networks architectures

## 10.1 Autoencoders (AEs)

Autoencoders [143] are a type of unsupervised representation learning methodology that forces an information bottleneck in the neural network architecture, see Figure 13. The bottleneck effect reduces the dimensionality of the network parameters to achieve a compression of the input data features to capture its most representative form without information loss (lossless encoding).

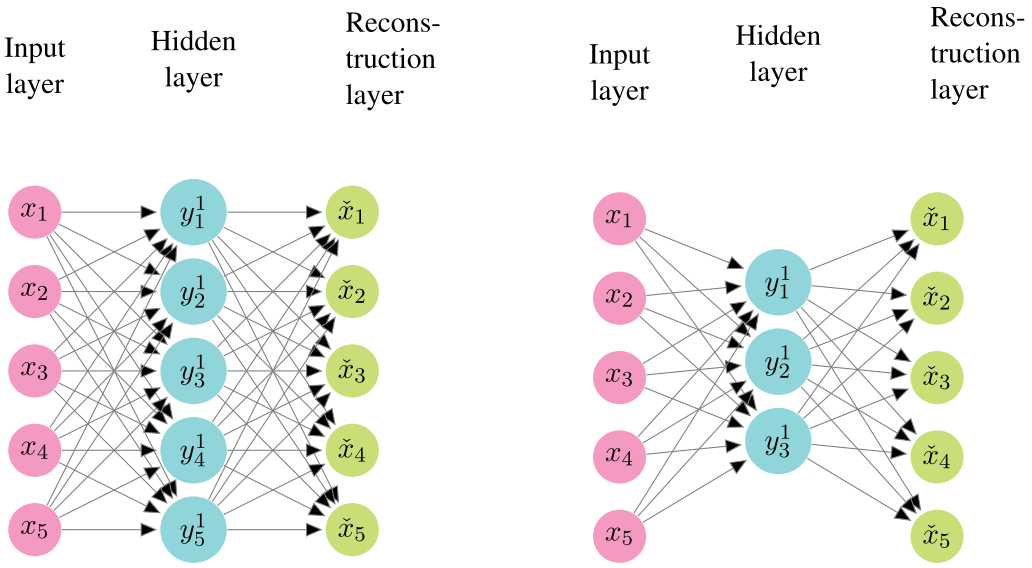



**Figure 13:** (*Left*): Ordinary feedforward linear network , (*Right*) Autoencoder linear network with a bottleneck (dimensionality reduction) effect

Autoencoders are a combination of an encoder and a decoder. The encoder $g_\phi$ comprises a series of hidden layers (more layers, deeper networks) between the input and the bottleneck. Similarly, the decoder $f_\theta$ is defined as the series of hidden layers between the bottleneck and the output. The structure is shown in Figure 14. The encoder network can learn the most important features that describe the data $z$ and the decoder uses them to reconstruct the original data. This means that the autoencoder learns the function that summarizes the data [143].

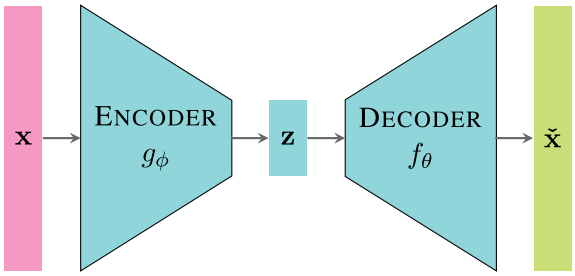



**Figure 14:** Autoencoder structure for data reconstruction. First, the encoder network compresses the input data $x$ into the bottleneck latent representation $z$. The decoder network decompresses $z$ into a representation of the data $\hat{x}$. The encoder network parameters (weights and biases) are labelled as $\phi$, whilst the decoder network parameters (weights and biases) are labelled as $\theta$.

Training a purely linear autoencoder is almost equivalent to the Principal Component Analysis (PCA) dimensionality reduction [144]. PCA finds the eigenvalue(s) that capture the highest variance in the dataset, which is equivalent to the representation of the data that reflects the highest amount of information. It is important to note that the linear projections given by the PCA are orthonormal to each other, independent, and maintain the same coordinate basis, however, this is not the case for the autoencoder. See Figure 15 for a visual explanation. Nonetheless, unlike PCA, autoencoders can

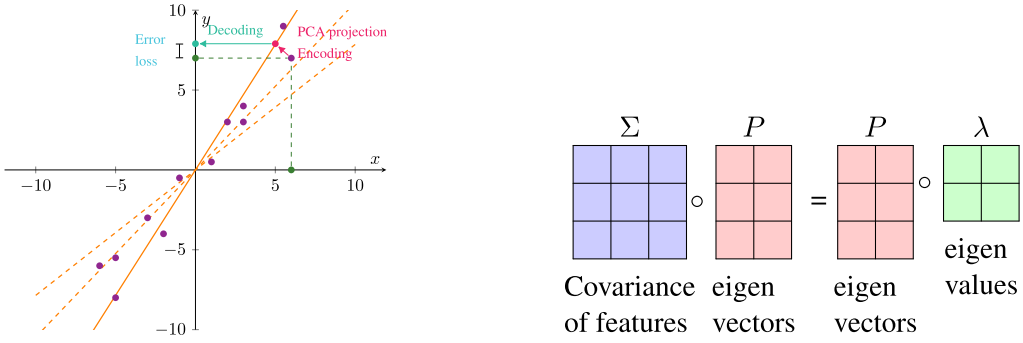



**Figure 15:** PCA dimensionality reduction. *Left* The first 3 PCA are shown in orange, solid line indicates the best fit. *Right* The encoder-decoder error loss is minimal when $\Sigma = P\lambda P^T$

benefit from non-linearities to go beyond simply stating a linear description (hyper-plane) of the data. Autoencoders can learn manifolds, defined as continuous non-intersecting surfaces. See Figure 16 for a visual representation that defines the data as the output of a linear function composing a hyper-plane versus the flexibility of capturing the data as the result of a non-linear function given by a manifold.

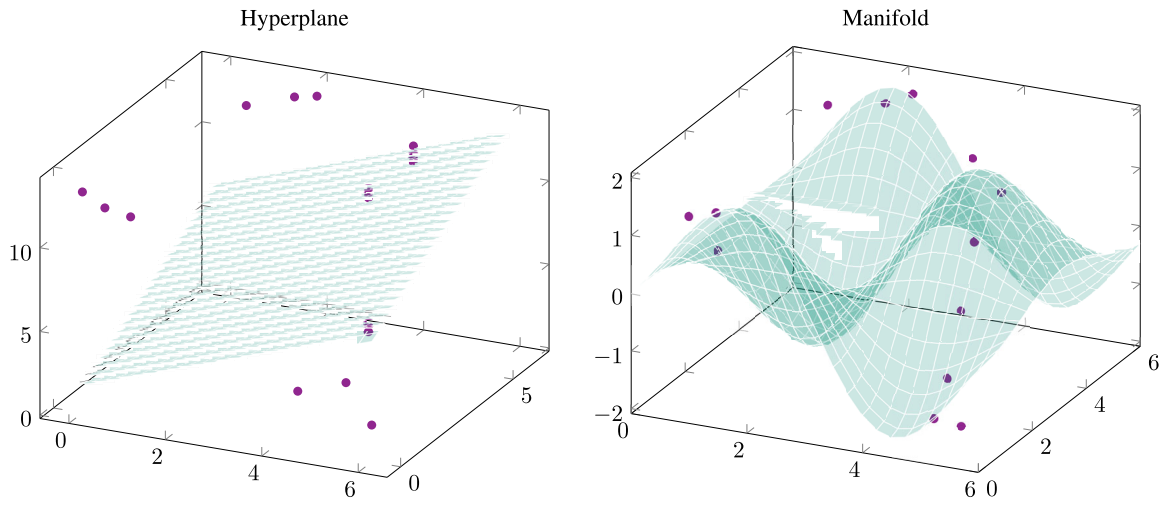



**Figure 16:** Differences between the flexibility of learning a linear function (hyperplane) vs a non-linear function (manifold). The manifold improves the capture of the data points structure.

The quality of the encoder and decoder functions is evaluated by computing an error loss function $\mathcal{L}$ between the input data $x$ and the reconstruction $\check{x}$,

$$\mathcal{L}(\theta, \phi) = \underbrace{\frac{1}{n}\sum_{i=1}^{n}(x_i - \check{x}_i)}_{\text{Reconstruction error loss (MSE)}} = \frac{1}{n}\sum_{i=1}^{n}(x^i - f_\theta(g_\phi(x_i))) \tag{53}$$

Autoencoders without a proper regularization term suffer from overfitting since they are persistently minimizing the reconstruction loss. This might lead to a latent space where some of its points are “meaningless" once decoded. For this reason, there are several types of latent space regularizers for Autoencoders, such as the L1 or L2 regularization terms.

## 10.2 Variational Autoencoders (VAEs)

Variational Autoencoders are a type of Autoencoder that maps the input data to a distribution instead of a simple matrix or vector. This is achieved by compressing the data into the parameters of the bottleneck “latent" distribution, which for many reasons (i.e easiness of being reparameterized) is usually set to the Normal distribution (or it’s higher dimensional counterpart, the Multivariate Normal). Therefore, the input data will be described (“encoded") in the form of a mean vector and a standard deviation. This adds a fundamental property and advantage to VAEs over AEs, which is that the latent representation or bottleneck must be continuous and regular and therefore we can use VAEs as generative models by sampling from the latent distribution [119, 145].

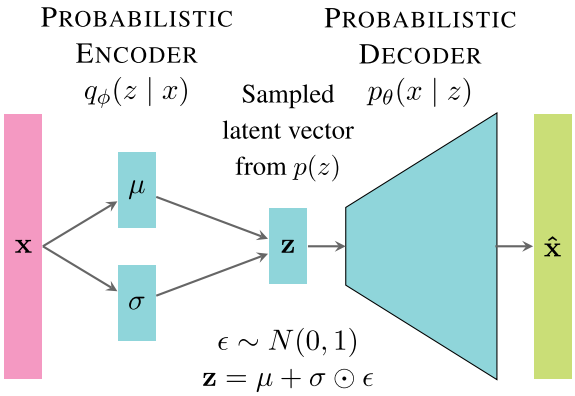



**Figure 17:** Variational Autoencoder structure. First, the probabilistic encoder network compresses the input data $x$ into the mean $\mu$ and variance $\sigma$ vectors that parameterize the distribution from which the latent representation of the data is sampled $p(z)$. The decoder network samples and decompresses $z$ into a reconstruction of the data $\hat{x}$. The reconstruction error is computed and backpropagated through the network.

The error loss function from the VAE needs to be upgraded to improve the generative process such that i) it's *continuous*, this grants that points close to each other in the latent space are decoded into similar outputs ii) it's *complete*, all the sampled points from the latent space when decoded, should be meaningful [146].

### 10.2.1 The error loss function for VAEs

The objective is to find the probability distribution of the latent space given the input data $q_\phi(z \mid x)$ that can encode the input data into its best feature representation. Simultaneously, we seek to find the best probability distribution $p_\theta(x \mid z)$ that decodes the latent space into it's native form.

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)} \xrightarrow[\text{by}]{\text{approximated}} q_\phi(z \mid x) \tag{54}$$

We proceed then to set the error loss function as the KL divergence between $q_\phi(z \mid x)$ and $p_\theta(z \mid x)$ which should be minimized to optimize the parameters. This minimization will find the best $g^*$ and $h^*$ functions, which are usually a combination of linear and non linear functions, by optimizing the best set of parameters $\phi$ and $\theta$ (i.e Neural Network's weights and biases) that parameterize them. The optimal $g^*$ and $h^*$ functions generate the parameters of the $p(z)$ distribution. For example, in the case of $z$ being set to a univariate Normal distribution, $g$ and $h$ will find the best values for $\mu_z$ and $\sigma_z$.

$$\begin{aligned}
(g^*, h^*) &= \underset{(g,h)\in G,H}{\arg\min}\ KL(q_\phi(z \mid x) \| p(z \mid x)) = \\
&= \underset{(g,h)\in G,H}{\arg\min} \left( \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z \mid x)] - \mathbb{E}_{z\sim q_\phi(z|x)} \left[ \log \frac{p_\theta(x \mid z)p(z)}{p(x)} \right] \right) \\
&= \underset{(g,h)\in G,H}{\arg\min} \left( \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z \mid x)] - \mathbb{E}_{z\sim q_\phi(z|x)}[\log p(x,z)]}_{\text{-ELBO}} + \mathbb{E}_{z\sim q_\phi(z|x)}[\log p(x)] \right)
\end{aligned} \tag{55}$$

As we can recognize, we cannot compute the above equation due to the intractability of $p(x)$, however, if we re-call from the ELBO derivation in Equation 32, we can see that we confront exactly the same issue. Therefore, if we want to simultaneously minimize the the KL divergence and maximize the probability of the observations, we can just simply maximize the ELBO. The transformation of the ELBO into an error loss function is as follows,

$$
\begin{aligned}
&\log p(x) \geq \text{ELBO} \\
&\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x,z) - \log q_\phi(z \mid x)] \\
&\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z \mid x)}\right] \\
&\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}\left[\frac{p_\theta(x \mid z)p(z)}{q_\phi(z \mid x)}\right] \\
&\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[p_\theta(x \mid z)] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\frac{p(z)}{q_\phi(z \mid x)}\right] \\
&\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[p_\theta(x \mid z)] + \int_z q_\phi(z \mid x)\log\frac{p(z)}{q_\phi(z \mid x)}\,dz \\
&\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[p_\theta(x \mid z)] + KL\left[p(z)\|q_\phi(z \mid x)\right] \\
&\log p(x) \geq \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}[p_\theta(x \mid z)]}_{\text{Reconstruction error}} - \underbrace{KL\left[q_\phi(z \mid x)\|p(z)\right]}_{\text{Regularization term}}
\end{aligned}
\tag{56}
$$

We have arrived at the “typical" form of the error loss function. In this case, the ELBO is a special kind of loss function, since we seek to maximize it while ordinary losses are usually minimized. In practice we compute the negative ELBO, which is optimized as it increases [119, 146].

$$
\text{-ELBO} = -\mathcal{L}(\theta,\phi) = KL(q_\phi(z \mid x)\|p(z)) - \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x \mid z)] \tag{57}
$$

To clarify the above defined syntax, we will take a simple example with univariate Normal distributions for our distributions such as,

$q_\phi(z \mid x) = \mathcal{N}(g_1(x), g_2(x))$,
where $g_1$ and $g_2$ are encoding functions (defined with parameters $\phi$) of the input data $x$ into the parameters $\mu_z$ and $\sigma_z$. They belong to the family of distribution functions $g_1 \in G_1$ and $g_2 \in G_2$
$p(z) = \mathcal{N}(0,1)$,
is the prior distribution over the latent space $z$ set in the decoder (model)
$p(x \mid z) = \mathcal{N}(h(z), c.I)$
where $h$ is the decoding function of the latent space $z$. It belongs to the family of distribution functions $h \in H$. $I$ is the identity matrix and $c$ is a positive constant.

(58)

In this example case, the KL divergence between 2 univariate Normal distributions, where one of them is a standard Normal, is as follows,

$$
\begin{aligned}
KL(q_\phi(z \mid x) \| p(z)) &= -\int q_\phi(z \mid x) \log p(z)\, dz + \int p(z) \log q_\phi(z \mid x)\, dz \\
&= \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}(1 + \log 2\pi\sigma_1^2) \\
&= \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \\
&\text{The components indexed with 2 correspond to } p(z) = N(0,1) \\
&= -\log\sigma_1 + \frac{\sigma_1^2 + \mu_2^2}{2} - \frac{1}{2}
\end{aligned} \tag{59}
$$

And the computation of the log-likelihood of the univariate Normal function is

$$
\mathbb{E}_{z\sim q_\phi(z|x)}[p_\theta(x \mid z)] = (2\pi\sigma^2)^{-\frac{n}{2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)\right) \tag{60}
$$

The univariate Normal distribution is very simple use-case and the Multivariate Normal case is usually a better approach. This situation is described in the VAE paper [119], where the encoder approximates the Multivariate latent space by making some independence assumptions. This allows to set the covariance of the Multivariate Normal as a diagonal matrix, which can be easily vectorized, and simplifies the calculation KL divergence between 2 Multivariate distributions. We use the same procedure in our variational evolutionary model presented in the manuscript 16.

### 10.2.2 The reparameterization trick for VAEs

Lastly, it is not possible to propagate the error loss through the random variable $p_\theta(z)$, therefore we again need to use the "reparameterization trick" (seen in Section 9.4.5) [139, 140, 141, 142] in order to transform $p_\theta(z)$ into a differentiable term. For example, if the prior over the latent space is a univariate Normal distribution, we simply will take the mean $\mu_z$, which is obtained from $g(x)$, plus the variance $\sigma_z$, which is obtained from $h(x)$, multiplied by a variance regulator $\epsilon \sim N(0,1)$. This means that the "sample" $z$ from the latent space is now a transformed version of the mean, $z = \mu_z + \epsilon\sigma_z$. Note that we do not backpropagate over $\epsilon$, it remains as a random variable without dependencies on $\phi$.

### 10.2.3 Amortized inference in VAEs

It is important to note that VAEs can benefit from amortized inference, meaning that we introduce correlations among the model's parameters [147]. This is the case if the statistical model counts with the appropriate feed-forward network that maps the observations to the latent space $q_\phi(z \mid x)$ and an additional feed-forward network that maps the latent space to the observations $p_\theta(x \mid z)$. This opposes the Mean Field Assumption utilised by ordinary CAVI (see Section 9.3.4), where each of the variational and latent parameters (and the dimensions within the latent parameters) is independent (uncorrelated) from each other. VAEs can also become factorized models by building independent networks for each of the parameters. The amortization character of VAEs has been a key contribution of the Draupnir evolutionary model for performing Ancestral Sequence Reconstruction [8].

# 11 Kabsch-Umeyama algorithm applied to protein superposition

The Theseus-PP and Theseus-HMC papers present a new method to confront the classical approaches in the field of protein superposition. The most commonly used method is the Kabsch-Umeyama algorithm [148, 149, 150]. This method is the 3D implementation of the Ordinary Partial Procrustes [151, 152] algorithm for performing superposition over a set of 3D coordinates. The method estimates the optimal rotation and translation via Singular Value Decomposition (SVD) that minimizes the Root Mean Squared Error (RMSD) [153]. The Kabsch-Umeyama algorithm for superimposing a protein B onto a protein A can be described as follows [154],

i) Perform translation by centering the protein coordinates A and B, of dimensions $nx3$, by subtracting the average of each coordinate from its respective column.

ii) Build the covariance matrix C by multiplying the sets of coordinates of the proteins to superimpose.

$$C_{nxn} = A_{nx3}B^T_{3xn} \text{ or } C_{nxn} = A^T_{3xn}B_{nx3}$$

iii) Compute the SVD of the covariance matrix C. This procedure decomposes C into the 3 transformations over the vectors of the coordinate basis that resulted in C : an initial rotation V, a scaling matrix D along the coordinate axes, and a final rotation U.

$$\text{SVD}(C) = UDV^T$$

iv) Define S, a correction matrix to scale V that detects and prevents reflection of the structure. Calculate the sign of the multiplication of the determinants of U and V:

$$\text{sign} = \det U \det V^T$$

$$\text{S} = \begin{cases} \text{Identity matrix (I) of size } nxn & \text{if sign} > 0 \\ \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{-1} \end{bmatrix} & \text{otherwise} \end{cases}$$

v) Compute the rotation matrix (R):

$$R_{3x3} = USV^T$$

vi) Rotate B:

$$\hat{B} = BR$$

vii) Compute RMSD between the x,y,z coordinates of the $C_\alpha$ atoms from fixed protein A and the newly rotated $\hat{B}$:

$$\text{RMSD}(A, \hat{B}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}((a_{ix} - b_{ix})^2) + (a_{iy} - b_{iy})^2) + (a_{iz} - b_{iz})^2)}$$

# 12 Stochastic Processes: An intuition

The last paper presented in this Ph.D. thesis makes use of a variant of the Gaussian process, the Ornstein–Uhlenbeck process, to encode the evolutionary process. This is a brief and illustrated guide to the concept of stochastic processes.

## 12.1 Brownian Motion

To introduce Gaussian processes, first, we need to define Brownian motion, also referred as Wiener processes. Brownian motion describes the random motion of particles caused by molecular interactions. It was first observed by the botanist Robert Brown in 1827, nonetheless, it was not mathematically described until 1905-1906 by Albert Einstein [155] and Marian Smoluchowski [156]. It was finally proved experimentally by Jean Perrin in 1908 [157]. Brownian motions can be either determined mechanically or enclosed in a probability model defined as a stochastic process. The stochastic process represents the random trajectory of a particle during an interval of time, see Figure 18 for a visual example. Brownian motions are type of Gaussian Process [158].

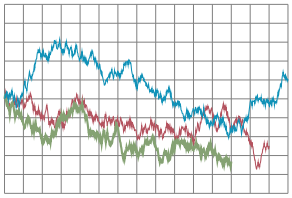


**Figure 18:** Brownian Motion: Visualization of the stochastic trajectory of 3 particles.

## 12.2 Gaussian Processes (GP)

### 12.2.1 Multivariate Normal distribution

The Multivariate Normal distribution is a generalization of the univariate normal distribution to higher dimensionalities. The univariate Normal samples 1 dimensional random variables (see Equation 61), whereas the multivariate counterpart samples n-dimensional random variables (see Equation 62). The Multivariate normal expresses the joint probability of correlated Gaussian distributions.

$$x \sim \mathcal{N}(\mu, \sigma^2) \tag{61}$$

$$\vec{x} \sim \mathcal{N}(\vec{\mu}, \Sigma) \tag{62}$$

Since we cannot visualize a Multivariate Normal, we appeal to the simpler Bivariate Normal distribution. The latter contemplates the correlations among a bivariate random variable, which is a vector with form $\vec{x} = [x_1, x_2]$. Those correlations are defined by the covariance matrix $\Sigma$. When the dimensions are independent, the covariance matrix is equal to the identity matrix, see Figure 19. The joint probability distribution of variable $\vec{x}$ given the covariance matrix is given by,

$$\begin{aligned} P(\vec{X}) = P(X_1, X_2) &= \mathcal{N}\left(\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}, \begin{bmatrix} Var(X_1) & Covar(X_1, X_2) \\ Covar(X_1, X_2)^T & Var(X_2)\end{bmatrix}\right) \\ &= \mid 2\pi\Sigma \mid^{-\frac{n}{2}} \; e^{-\frac{1}{2}(\vec{X}-\vec{\mu})^T\Sigma^{-1}(\vec{X}-\vec{\mu})} \end{aligned} \tag{63}$$

where $n$ is the number of dimensions.

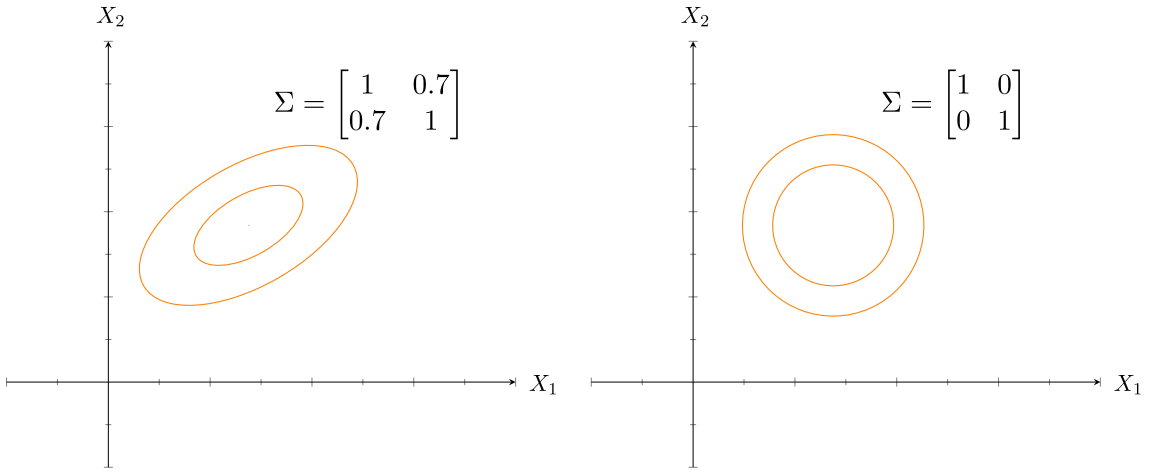



**Figure 19:** Illustration of the effect of the covariance matrix between the components of the random variable $\vec{x}$ that define a bivariate Gaussian. *(Left)* There is a correlation among the dimensions of 0.7. *(Right)* The dimensions are independent, the covariance matrix is the Identity matrix, the correlation is 0.

In Figures 20 and 21 we sample several data points from the joint distribution $P(X_1, X_2)$. Then we represent their correlations using the vectors between the coordinates of the sample and the base axis [159]. The new representations co-vary equivalently as the samples in the process do, see Figures 21 and 22. We can extend this methodology to illustrate the covariance between high dimensional variables, without needing to directly visualize a Multivariate Normal, see figure 23.

If we wish to calculate the conditional probability of the variable $X_2$ given a fixed value of $Y_1$ and $\Sigma$, we can observe that it also follows a Gaussian distribution from which we can sample several values of $X_2$ (see Figure 24). The ability to calculate the conditional probability has been crucial property to perform ancestral sequence reconstruction in paper 16.

$$P(X_2 \mid X_1 = x_1) = \mathcal{N}(\overbrace{\mu_1 + Covar(X_1, X_2)Var(X_2)^{-1}(x_1 - \mu_2)}^{\text{Conditional mean}}, \underbrace{Var(X_1) - Covar(X_1, X_2)Var(X_2)^{-1}Covar(X_1, X_2)^T}_{\text{Conditional covariance}}) \tag{64}$$

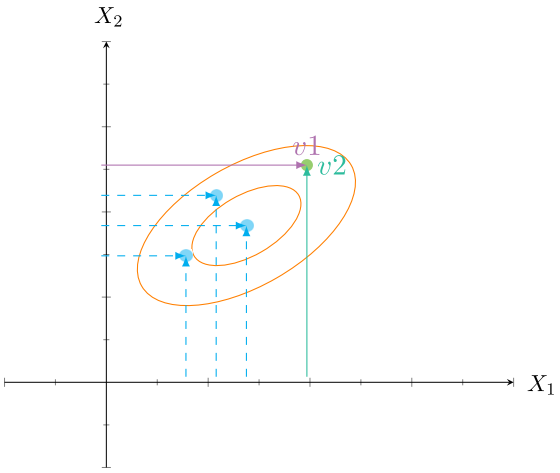



**Figure 20:** Sampling from the joint bivariate distribution $P(X_1, X_2)$

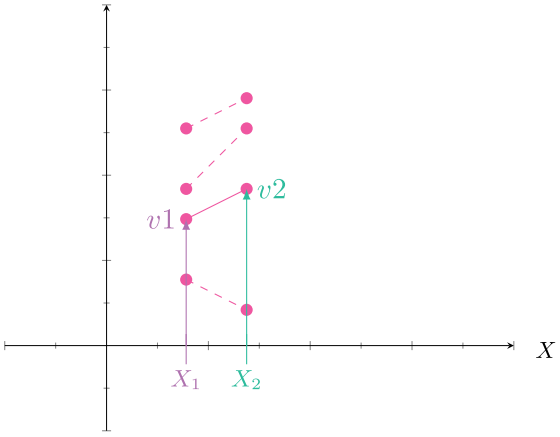



**Figure 21:** Re-intrepretation of the sampling from the joint bivariate probability $P(X_1, X_2)$ in Figure 20 to visualize the correlations among the samples.

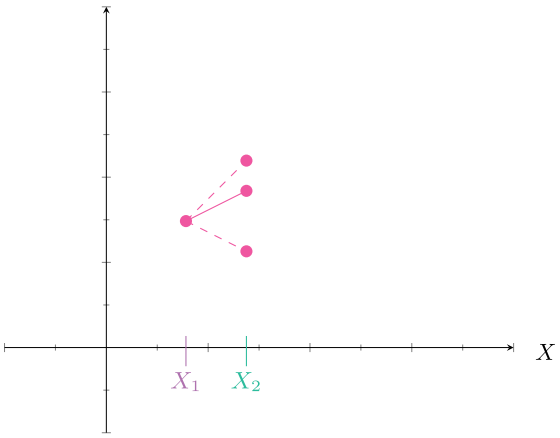



**Figure 22:** Conditionally sampling $X_2$ given a fixed value of $X_1$.

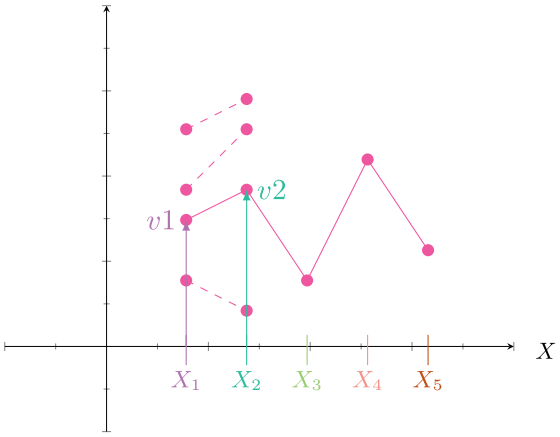



**Figure 23:** Visualization of the correlations among the samples from a Multivariate distribution $P(X_1, X_2, X_3, X_4, X_5)$ with a 5 dimensional random variable.

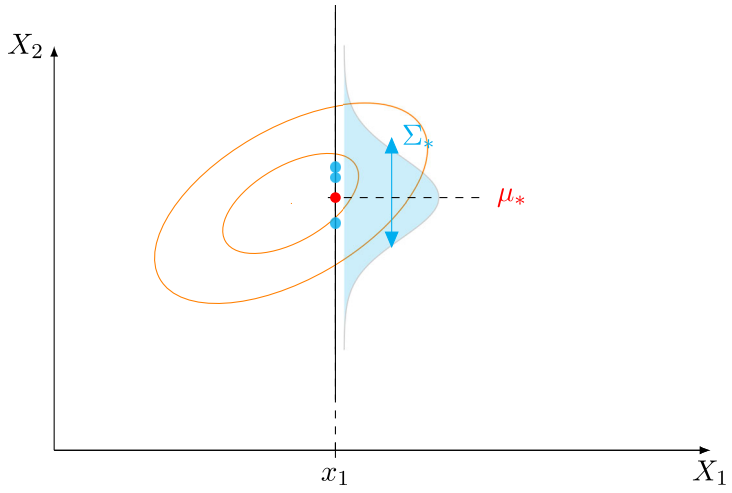



**Figure 24:** Representation of $P(X_2 \mid X_1, \Sigma_*, \mu_*)$, the conditional probability of $X_2$ given a fixed value of $X_1$, a mean $\mu_*$ and a covariance matrix $\Sigma_*$.This conditional probability gets closer to the standard normal as $X_1$ and $X_2$ become uncorrelated.

### 12.2.2 Gaussian Process prior

Gaussian Processes determine a joint Gaussian probability distribution over smooth (continuous) functions $f(x)$. Smooth functions can be understood as infinite vectors, which in this case are Gaussian density functions that are defined in all $\mathbb{R}$. Gaussian Processes are known as infinite (non-parametric) models which are characterized by a mean function $\mu(x)$ and a kernel function $\mathcal{K}(x, x')$ [158]. They can be observed as a tool to solve non-linear regression with uncertainty over the predictions. One of the desired properties of a Gaussian Process is that the uncertainty over the predictions increases as the number of observations decreases [160]

$$\overbrace{f(x)}^{\text{Smooth function}} \sim \mathcal{GP}(\mu(x), \Sigma(x, x')) \tag{65}$$

$\mu(x)$ : Average function, indicates the average at any point in the space.
$\mathcal{K}(x, x') : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ : Function that takes as an input 2 points from the domain $\mathbb{R}$ (x and x') and models their covariance.

### 12.2.3 Marginalizing the GP

In reality we cannot deal with infinite dimensions, rather we have a finite set of train data points $x \in \{x_1, ..., x_n\}$ and test data points $x^* \in \{x_1^*, ..., x_n^*\}$ . Instead, we need to find the joint Gaussian distribution that is defined only at those points. We require to marginalize the Gaussian Process, thus, we arrive at a finite-dimensional Multivariate Normal distribution. Marginalization implies discarding the remaining data points in $\mathbb{R}$ that are not $x$ or $x^*$, which are indicated as $x^0$ in Equation 66.

$$\begin{bmatrix} f(x) \\ f(x^*) \\ \cancel{f(x^0)} \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} \mu(x) \\ \mu(x^*) \\ \cancel{m(x^o)} \end{bmatrix}, \begin{bmatrix} K(x,x) & K(x,x^*) & K(x,x^0) \\ K(x,x^*)^T & K(x^*,x^*) & K(x^*,x^0) \\ \cancel{K(x,x^0)^T} & \cancel{K(x^*,x^0)} & \cancel{K(x^0,x^0)} \end{bmatrix}\right) \tag{66}$$

$$P\left(\begin{bmatrix} f(x) \\ f(x^*) \end{bmatrix}\right) = \mathcal{N}(\vec{\mu}, \Sigma(x, x')) \tag{67}$$

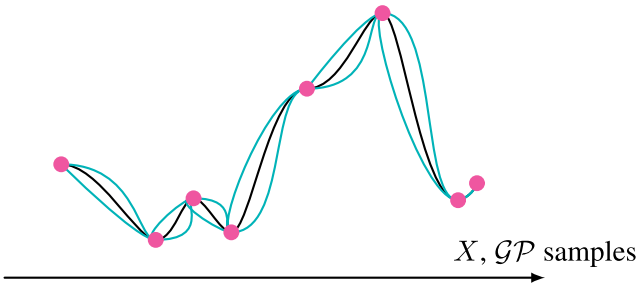



**Figure 25:** Gaussian Process (see Equation 67). The training points (pink dots) have reduced uncertainty, whereas the rest of the points have larger variances (blue regions).

The covariance of a Gaussian Process is defined by a kernel of choice. A kernel is a function that can generate a positive definite symmetric matrix. The shapes (smooth, periodic, noisy...) of the functions that capture how the data was originated are characterized by the choice of the kernel. For example, the squared exponential kernel for a Gaussian process is determined as follows,

$$\begin{aligned}\Sigma(x, x') &= \mathcal{K}(x, x') + \overbrace{I\sigma_n^2}^{\text{Noise}} \\ &= \sigma_v^2 e^{-\frac{(x-x')^2}{2l^2}} + I\sigma_n^2\end{aligned} \tag{68}$$

where $x$ and $x'$ are the points at which the function is evaluated, $l$ is the characteristic length scale or horizontal noise, which defines the smoothness in the curve (higher values form a more wiggly function) and $\sigma_v$ is the vertical scale which determines the vertical noise.

Gaussian processes optimize the parameters of the kernel by observing the entire data set simultaneously. This means that their computation is expensive and escalates cubically with the number of observations $\mathcal{O}^3$, due to the need of calculating the inverse of the covariance matrix for sampling from the Multivariate Normal distribution (see Equation 12.2.1). However, recent advances in the field of escalating Gaussian Processes with the number of observations have been developed. For example, sparse Gaussian Processes [161] with their use of *pseudo-points*, a small subset of observations that shifts with every update of the kernel, can now account to perform inference on large Gaussian Processes.

# 13 Paper 1: A probabilistic programming approach to protein superposition

**Authors:** Lys Sanz Moreta, Ahmad Salim Al-Sibahi, Douglas Theobald, William Bullock, Basile Nicolas Rommes, Andreas Manoukian, and Thomas Hamelryck.

**Motivation:** Within the probabilistic programming framework Pyro [9], we implement a new probabilistic version of Theseus [2] with the additional use of quality priors over the superposition parameters. This new version of the original model was envisioned to serve as an improved scoring function during training methods for protein structure prediction.

# A Probabilistic Programming Approach to Protein Structure Superposition

Lys Sanz Moreta[1*], Ahmad Salim Al-Sibahi[1,2*], Douglas Theobald[3], William Bullock[4], Basile Nicolas Rommes[4], Andreas Manoukian[4], and Thomas Hamelryck[1,4*]

[1] Department of Computer Science. University of Copenhagen, Denmark
[2] Skanned.com, Denmark
[3] Department of Biochemistry. Brandeis University. Waltham, MA 02452, USA
[4] The Bioinformatics Centre. Section for Computational and RNA Biology. University of Copenhagen, Denmark

[*] Corresponding authors.
moreta@di.ku.dk (Lys Sanz Moreta),
ahmad@di.ku.dk (Ahmad Salim Al-Sibahi),
thamelry@bio.ku.dk (Thomas Hamelryck)

***Abstract*—Optimal superposition of protein structures is crucial for understanding their structure, function, dynamics and evolution. We investigate the use of probabilistic programming to superimpose protein structures guided by a Bayesian model. Our model THESEUS-PP is based on the THESEUS model, a probabilistic model of protein superposition based on rotation, translation and perturbation of an underlying, latent mean structure. The model was implemented in the deep probabilistic programming language Pyro. Unlike conventional methods that minimize the sum of the squared distances, THESEUS takes into account correlated atom positions and heteroscedasticity (i.e., atom positions can feature different variances). THESEUS performs maximum likelihood estimation using iterative expectation-maximization. In contrast, THESEUS-PP allows automated maximum a-posteriori (MAP) estimation using suitable priors over rotation, translation, variances and latent mean structure. The results indicate that probabilistic programming is a powerful new paradigm for the formulation of Bayesian probabilistic models concerning biomolecular structure. Specifically, we envision the use of the THESEUS-PP model as a suitable error model or likelihood in Bayesian protein structure prediction using deep probabilistic programming.**

***Index Terms*—protein superposition, Bayesian modelling, deep probabilistic programming, protein structure prediction**

## I. Introduction

In order to compare biomolecular structures, it is necessary to superimpose them onto each other in an optimal way. The standard method minimizes the sum of the squared distances (root mean square deviation, RMSD) between the matching atom pairs. This can be easily accomplished by shifting the centre of mass of the two proteins to the origin and obtaining the optimal rotation using singular value decomposition [1] or quaternion algebra [2], [3]. These methods however typically assume that all atoms have equal variance (homoscedasticity) and are uncorrelated. This is problematic in the case of proteins with flexible loops or flexible terminal regions, where the atoms can posit high variance. Here we present a Bayesian model that is based on the previously reported THESEUS model [4]–[6]. THESEUS is a probabilistic model of protein superposition that allows for regions with low and high variance (heteroscedasticity), corresponding respectively to conserved and variable regions [4], [5]. THESEUS assumes that the structures which are to be superimposed are translated, rotated and perturbed observations of an underlying latent, mean structure $\mathbf{M}$.

In contrast to the THESEUS model which features maximum likelihood parameter estimation using iterative expectation maximization, we formulate a Bayesian model (THESEUS-PP) and perform maximum a-posteriori (MAP) parameter estimation. We provide suitable prior distributions over the rotation, the translations, the variances and the latent, mean model. We implemented the entire model in the deep probabilistic programming language Pyro [7], using its automatic inference features. The results indicate that deep probabilistic programming readily allows the implementation, estimation and deployment of advanced non-Euclidean models relevant to structural bioinformatics. Specifically, we envision that THESEUS-PP can be used as a likelihood function in Bayesian protein structure prediction using deep probabilistic programming.

## II. Methods

### A. Overall model

According to the THESEUS model [4], each observed protein structure $\mathbf{X}_n$ is a noisy observation of a rotated and

978-1-7281-1462-0/19/$31.00 2019 IEEE

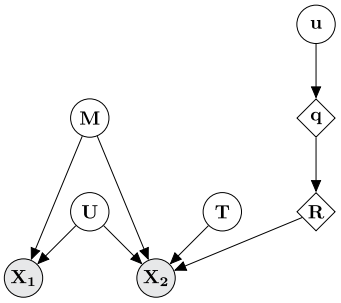



Fig. 1: The THESEUS-PP model as a Bayesian graphical model. $\mathbf{M}$ is the latent, mean structure, which is an $N$-by-3 coordinate matrix, where $N$ is the number of atoms. $\mathbf{T}$ is the translation. $\mathbf{q}$ is a unit quaternion calculated from three random variables $\mathbf{u}$ sampled from the unit interval. $\mathbf{R}$ is the corresponding rotation matrix. $\mathbf{U}$ is the among-row variance matrix of a matrix-normal distribution. $\mathbf{X}_1$ and $\mathbf{X}_2$ are $N$-by-3 coordinate matrices representing the proteins to be superimposed. Circles denote random variables. A lozenge denotes a deterministic transformation of a random variable. Shaded circles denote observed variables. Bold capital and bold small letters represent matrices and vectors, respectively.

translated latent, mean structure $\mathbf{M}$ with noise $\mathbf{E}_n$,

$$\mathbf{X}_n = (\mathbf{M} + \mathbf{E}_n)\mathbf{R}_n - \mathbf{1}_N\mathbf{T}_n \tag{1}$$

where $n$ is an index that identifies the protein, $\mathbf{R}$ is a rotation matrix, $\mathbf{T}$ is a three-dimensional translation, $\mathbf{E}$ is the error and $\mathbf{M}$ and $\mathbf{X}$ are matrices with the atomic coordinate vectors along the rows,

$$\mathbf{M} = \begin{bmatrix} \mathbf{m}_0 \\ \cdots \\ \mathbf{m}_{N-1} \end{bmatrix}, \qquad \mathbf{X}_n = \begin{bmatrix} \mathbf{x}_{n,0} \\ \cdots \\ \mathbf{x}_{n,N-1} \end{bmatrix}. \tag{2}$$

Another way of representing the model is seeing $\mathbf{X}_n$ as distributed according to a matrix-normal distribution with mean $\mathbf{M}$ and covariance matrices $\mathbf{U}$ and $\mathbf{V}$ - one concerning the rows and the other the columns.

The matrix-normal distribution can be considered as an extension of the standard multivariate normal distribution from vector-valued to matrix-valued random variables. Consider a random variable $\mathbf{X}$ distributed according to a matrix-normal distribution with mean $\mathbf{M}$, which in our case is an $N \times 3$ matrix where N is the number of atoms. In this case, the matrix-normal distribution is further characterized by an $N \times N$ row covariance matrix $\mathbf{U}$ and a $3 \times 3$ column covariance $\mathbf{V}$. Then, $\mathbf{X} \sim \mathcal{MN}(\mathbf{M}, \mathbf{U}, \mathbf{V})$ will be equal to

$$\mathbf{X} = \mathbf{M} + \sqrt{\mathbf{U}}\mathbf{Q}\sqrt{\mathbf{V}}, \tag{3}$$

where $\mathbf{Q}$ is an $N \times 3$ matrix with elements distributed according to the standard normal distribution.

To ensure identifiability, one (arbitrary) protein $\mathbf{X}_1$ is assumed to be a fixed noisy observation of the structure $\mathbf{M}$:

$$\mathbf{X}_1 \sim \mathcal{MN}(\mathbf{M}, \mathbf{U}, \mathbf{V}). \tag{4}$$

The other protein $\mathbf{X}_2$ is assumed to be a noisy observation of the rotated as well as translated mean structure $\mathbf{M}$:

$$\mathbf{X}_2 \sim \mathcal{MN}(\mathbf{MR} - \mathbf{1}_N\mathbf{T}, \mathbf{U}, \mathbf{V}). \tag{5}$$

Thus, the model uses the same covariance matrices $\mathbf{U}$ and $\mathbf{V}$ for the matrix-normal distributions of both $\mathbf{X}_1$ and $\mathbf{X}_2$.

### *B. Bayesian posterior*

The graphical model of THESEUS-PP is shown in Figure 1. The corresponding Bayesian posterior distribution is

$$\begin{aligned} p(\mathbf{R}, \mathbf{T}, \mathbf{M}, \mathbf{U} | \mathbf{X}_1, \mathbf{X}_2) \propto \\ p(\mathbf{X}_1, \mathbf{X}_2 | \mathbf{M}, \mathbf{R}, \mathbf{T}, \mathbf{U}) p(\mathbf{M}) p(\mathbf{T}) p(\mathbf{R}) p(\mathbf{U}) = \\ p(\mathbf{X}_1 | \mathbf{M}, \mathbf{U}) p(\mathbf{X}_2 | \mathbf{MR} - \mathbf{1}_N\mathbf{T}, \mathbf{U}) \\ p(\mathbf{M}) p(\mathbf{T}) p(\mathbf{R}) p(\mathbf{U}). \end{aligned} \tag{6}$$

Below, we specify how each of the priors and the likelihood function is formulated and implemented.

### *C. Prior for the mean structure*

Recall that according to the THESEUS-PP model, the atoms of the structures to be superimposed are noisy observations of a mean structure $\mathbf{M}$. Typically, only $C_\alpha$ atoms are considered and in that case, $N$ corresponds to the number of amino acids. Hence, we need to formulate a prior distribution over the latent, mean structure $\mathbf{M}$.

We use an uninformative prior for $\mathbf{M}$. Each element of $\mathbf{M}$ is sampled from a Student's t-distribution with degrees of freedom ($\nu = 1$), mean ($\boldsymbol{\mu} = 0$) and a uniform diagonal variance ($\boldsymbol{\sigma}^2 = 3$). The Student's t-distribution is chosen over the normal distribution for reasons of numerical stability: the fatter tails of the Student's t-distribution avoid numerical problems associated with highly variable regions.

### *D. Prior over the rotation*

In the general case, we have no *a priori* information on the optimal rotation. Hence, we use a uniform prior over the space of rotations. There are several ways to construct such a uniform prior. We have chosen a method that makes use of quaternions [8]. Quaternions are the 4-dimensional extensions of the better known 2-dimensional complex numbers. Unit quaternions form a convenient way to represent rotation matrices. For our goal, the overall idea is to sample uniformly from the space of unit quaternions. Subsequently, the sampled unit quaternions are transformed into the corresponding rotation matrices, which establishes a uniform prior over rotations.

A unit quaternion $\mathbf{q} = (w, x, y, z)$ is sampled in the following way. First, three independent random variables are sampled from the unit interval,

$$u_0, u_1, u_2 \sim U(0, 1). \tag{7}$$

Then, four auxiliary deterministic variables $(\theta_1, \theta_2, r_1, r_2)$ are calculated from $u_1, u_2, u_3$,

$$\theta_1 = 2\pi u_1, \tag{8a}$$
$$\theta_2 = 2\pi u_2 \tag{8b}$$
$$r_1 = \sqrt{1-u_0}, \tag{8c}$$
$$r_2 = \sqrt{u_0}. \tag{8d}$$

The unit quaternion $\mathbf{q}$ is then obtained in the following way,

$$\begin{aligned}\mathbf{q} &= (w, x, y, z) \\ &= (r_2 \cos\theta_2, r_1 \sin\theta_1, r_1 \cos\theta_1, r_2 \sin\theta_2).\end{aligned} \tag{9}$$

Finally, the unit quaternion $\mathbf{q}$ is transformed into its corresponding rotation matrix $\mathbf{R}$ as follows,

$$\mathbf{R} = \begin{bmatrix} w^2+x^2-y^2-z^2 & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & w^2-x^2+y^2-z^2 & 2(yz-wx) \\ 2(xz-wy) & 2(yz+wx) & w^2-x^2-y^2+z^2 \end{bmatrix} \tag{10}$$

### E. Prior over the translation

For the translation, we use a standard trivariate normal distribution,

$$\mathbf{T} \sim \mathcal{N}(0, \mathbf{I}_3) \tag{11}$$

where $\mathbf{I}_3$ is the three-dimensional identity matrix.

### F. Prior over U

The Student's t-distribution variance over the rows is sampled from the half-normal distribution with standard deviation set to 1.

$$\sigma_i \sim \mathcal{N}_+(1). \tag{12}$$

### G. Likelihood

In our case, the matrix-normal likelihood of THESEUS reduces to a product of univariate Student's t-distributions. Again, we use the Student's t-distribution rather than the normal distribution (as in THESEUS) for reasons of numerical stability. Below, we have used trivariate Student's t-distributions with diagonal covariance matrices for ease of notation. The likelihood can thus be written as

$$\begin{aligned} &p(\mathbf{X}_1, \mathbf{X}_2 \mid \mathbf{M}, \mathbf{T}, \mathbf{R}, \mathbf{U}) \\ &\quad = p(\mathbf{X}_1 \mid \mathbf{M}, \mathbf{U}) p(\mathbf{X}_2 \mid \mathbf{M}, \mathbf{T}, \mathbf{R}, \mathbf{U}) \\ &\quad = \prod_{i=1}^{N} t_1(\mathbf{x}_{1,i} \mid \mathbf{m}_i, \sigma_i \mathbf{I}_3) \\ &\qquad \times t_1(\mathbf{x}_{2,i} \mid [\mathbf{MR} - \mathbf{1}_N \mathbf{T}]_i, \sigma_i \mathbf{I}_3), \end{aligned} \tag{13}$$

where the product runs over the matrix rows that contain the $x, y, z$ coordinates of $\mathbf{X}_1, \mathbf{X}_2$ and the rotated and translated latent, mean structure $\mathbf{M}$.

### H. Algorithm

**Algorithm 1** The Theseus-PP model.

▷ *Prior over the elements of* $\mathbf{M}$
$\mathbf{m}_i \sim t_1(\mathbf{0}, \sigma_M \mathbf{I}_3)$, where $i$ indicates the atom position
▷ *Prior over the translation*
$\mathbf{T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_3)$
▷ *Prior over rotation*
$\mathbf{u}_j \sim U[0, 1]$, for $j$ from 0 to 2
$\mathbf{q} \leftarrow$ Quaternion$(\mathbf{u})$
$\mathbf{R} \leftarrow$ RotationMatrix$(\mathbf{q})$
▷ *Prior over diagonal covariance matrix* $\mathbf{U}$
$\sigma_i \sim \mathcal{N}_+(1)$
▷ *Likelihood over the* $N$ *atom coordinates*
$\mathbf{x}_{1,i} \sim t_1(\mathbf{m}_i, \sigma_i \mathbf{I}_3))$
$\mathbf{x}_{2,i} \sim t_1([\mathbf{RM} - \mathbf{1}_N \mathbf{T}]_i, \sigma_i \mathbf{I}_3)$

### I. Initialization

Convergence of the MAP estimation can be greatly improved by selecting suitable starting values for certain variables and by transforming the two structures $\mathbf{X}_1$ and $\mathbf{X}_2$ in a suitable way. First, we pre-superimpose the two structures using conventional least-squares superposition. Therefore, the starting rotation can be initialized close to the identity matrix (ie., no rotation). This is done by setting the vector $\mathbf{u}$ to $(0.9, 0.1, 0.9)$.

We further improve performance by initializing the mean structure $\mathbf{M}$ to the average of the two pre-superimposed structures $\mathbf{X}_1$ and $\mathbf{X}_2$.

### J. Maximum a-posteriori optimization

We performed MAP estimation using Pyro's AutoDelta guide. For optimization, we used AdagradRMSProp [9], [10] with the default parameters for the learning rate (1.0), momentum (0.1) and step size modulator ($1.0 \times 10^{-16}$). A fragment of the model implementation in Pyro can be seen in Figure 3 in the Appendix.

Convergence was detected using Earlystop from Pytorch's Ignite library (version 0.2.0) [11]. This method evaluates the stabilization of the error loss and stops the optimization according to the value of the *patience* parameter. The *patience* value was set to 25.

## III. Materials

### Proteins

The algorithm was tested on several proteins from the RCSB protein database [12] that were obtained from Nuclear Magnetic Resonance (NMR) experiments. Such structures typically contain several models of the same protein. These models represent the structural dynamics of the protein in an aqueous medium and thus typically contain both conserved and variable regions. This makes them challenging targets for conventional RMSD superposition. We used the following structures: 1ADZ, 1AHL, 1AK7, 2CPD, 2KHI, 2LKL and 2YS9.

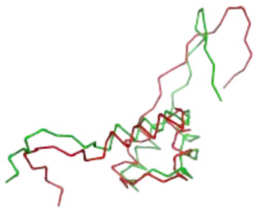
## IV. Results

The algorithm was executed 15 times on each protein (see TABLE I) with different seeds. The computations where carried on a Intel Core i7-8750H CPU 2.20GHz processor.

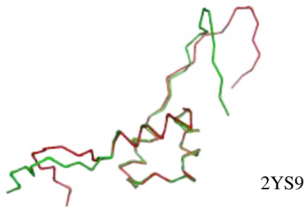
TABLE I: Results of applying THESEUS-PP to the test structures. First column: PDB identifier. Second column: the number of $C_\alpha$ atoms used in the superposition. Third column: the model identifiers. Fourth column: mean convergence time and standard deviation. Last column: Number of epochs.

| PBD ID | Length (Amino Acids) | Protein Models | Average Computational Time (seconds) | Epochs |
|---|---|---|---|---|
| 1ADZ | 71 | 0 and 1 | 0.64± 0.15 | 210±45 |
| 1AHL | 49 | 0 and 2 | 0.53± 0.119 | 166±36 |
| 1AK7 | 174 | 0 and 1 | 0.74± 0.23 | 222±67 |
| 2CPD | 99 | 0 and 2 | 0.51± 0.0.093 | 161±30 |
| 2KHI | 95 | 0 and 1 | 0.47±0.11 | 149±39 |
| 2LKL | 81 | 0 and 8 | 0.59±0.092 | 187±30 |
| 2YS9 | 70 | 0 and 3 | 0.47±0.11 | 144±33 |

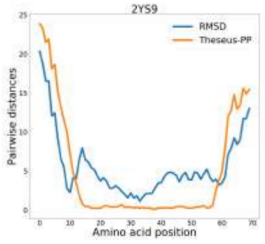
An example of a pair of superimposed structures is shown in Figure 2. For comparison, the superposition resulting from the conventional RMSD method, as calculated using Biopython [13], is shown on the left (Figure 2a). The THESEUS-PP superposition is shown on the right (Figure 2b). Note how the former fails to adequately distinguish regions with high from regions with low variance, resulting in poor matching of conserved regions. Additional, similar figures of superimposed structures can be found in the Appendix.



(a) Kabsch-RMSD




(b) Theseus-PP




(c)

Fig. 2: Protein superposition for two conformations of protein 2YS9 obtained from (a) conventional RMSD superimposition and (b) THESEUS-PP. The protein in green is rotated ($X_2$). The images are generated with PyMOL [14]. Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.

## V. Conclusion

Probabilistic programming is a powerful, emerging paradigm for probabilistic protein structure analysis, prediction and design. Here, we present a Bayesian model for protein structure superposition implemented in the deep probabilistic programming language Pyro and building on the previously reported THESEUS maximum likelihood model. MAP estimates of its parameters are readily obtained using Pyro's automated inference engine.

The original THESEUS algorithm, which makes use of maximum likelihood estimation using iterative expectation maximization, is considerably faster with an average execution time under 0.1 s. Although some of the longer execution time in THESEUS-PP is due to the use of variational inference and priors, it is clear that the flexibility and productivity of a probabilistic programming language can come with a speed penalty.

Recently, end-to-end protein protein structure prediction using deep learning methods has become possible [15]. We envision that Bayesian protein structure prediction will soon be possible using a deep probabilistic programming approach, which will lead to protein structure predictions with associated statistical uncertainties. In order to achieve this goal, suitable error models and likelihood functions need to be developed and incorporated in these models. The THESEUS-PP model can potentially serve as such an error model, by interpreting $\mathbf{M}$ as the predicted structure and a single rotated and translated $\mathbf{X}$ as the observed protein structure. During training of the probabilistic model, regions in $\mathbf{M}$ that are wrongly predicted can be assigned high variance, while correctly predicted regions can be assigned low variance. Thus, it can be expected that an error model based on THESEUS-PP will make estimation of these models easier, as the error function can more readily distinguish between partly correct and entirely wrong predictions, which is notoriously difficult for RMSD-based methods [16].

## Contributions and Acknowledgements

Implemented algorithm in Pyro: *LSM*. Contributed code: *ASA*, *AM*. Wrote article: *LSM*, *TH*, *ASA*. Prototyped algorithm in the probabilistic programming language PyMC3 [17]: *TH*, *WB*, *BNR*. Performed experiments: *LSM*. Designed experiments: *TH*, *DT*. *LSM* and *ASA* acknowledge support from the Independent Research Fund Denmark (grant: "Resurrecting ancestral proteins in silico to understand how cancer drugs work") and Innovationsfonden/Skanned.com (grant: "Intelligent accounting document management using probabilistic programming"), respectively. We thank Jotun Hein, Michael Golden, Kanti Mardia and Wouter Boomsma for suggestions and discussions.

## Data and Software Availability

Pyro [7] and PyTorch [11] based code is a available at https://github.com/LysSanzMoreta/Theseus-PP

## References


[1] W. Kabsch, "A discussion of the solution for the best rotation to relate two sets of vectors," *Acta Cryst. A*, vol. 34, pp. 827–828, 1978.

[2] B. Horn, "Closed-form solution of absolute orientation using unit quaternions," *J. Opt. Soc. Am. A*, vol. 4, pp. 629–642, 1987.

[3] E. Coutsias, C. Seok, and K. Dill, "Using quaternions to calculate rmsd," *J. Comp. Chem.*, vol. 25, pp. 1849–1857, 2004.

[4] D. L. Theobald and D. S. Wuttke, "Empirical Bayes hierarchical models for regularizing maximum likelihood estimation in the matrix Gaussian Procrustes problem," *PNAS*, vol. 103, pp. 18 521–18 527, 2006.

[5] D. L. Theobald and P. A. Steindel, "Optimal simultaneous superpositioning of multiple structures with missing data," *Bioinformatics*, vol. 28, pp. 1972–1979, 2012.

[6] K. Mardia and I. Dryden, "The statistical analysis of shape data," *Biometrika*, vol. 76, no. 2, pp. 271–281, 1989.

[7] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep Universal Probabilistic Programming," *Journal of Machine Learning Research*, 2018.

[8] X. Perez-Sala, L. Igual, S. Escalera, and C. Angulo, "Uniform sampling of rotations for discrete and continuous learning of 2D shape models," in *Robotic Vision: Technologies for Machine Learning and Vision Applications*. IGI Global, 2013, pp. 23–42.

[9] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[10] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[12] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic acids research*, vol. 28, no. 1, pp. 235–242, 2000.

[13] P. J. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski *et al.*, "Biopython: freely available python tools for computational molecular biology and bioinformatics," *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, 2009.

[14] Schrödinger, LLC, "The PyMOL molecular graphics system, version 1.8," November 2015.

[15] M. AlQuraishi, "End-to-end differentiable learning of protein structure," *Cell Systems*, 2019.

[16] I. Kufareva and R. Abagyan, "Methods of protein structure comparison," in *Homology Modeling*. Springer, 2011, pp. 231–257.

[17] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in python using PyMC3," *PeerJ Computer Science*, vol. 2, p. e55, 2016.

## VI. Appendix

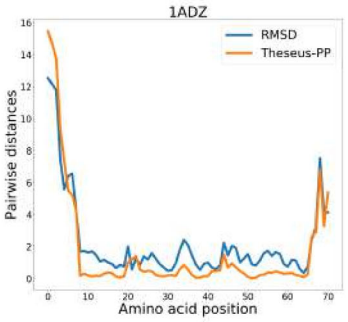
```
    # Prior over mean M, with N=number of atoms
    M = pyro.sample("M", dist.StudentT(1,0,3).expand_by([N,3]).to_event(2))
    # Prior over variances U
    U = pyro.sample("U", dist.HalfNormal(1).expand_by([N]).to_event(1))
    U =  U.reshape(N,1).repeat(1,3).view(-1)
    # Prior over translation T
    T = pyro.sample("T", dist.Normal(0,1).expand_by([3]).to_event(1))
    # Prior over rotation R
    u = pyro.sample("u",dist.Uniform(0,1).expand_by([3]).to_event(1))
    # Transformation: turn u via a unit quaternion into a rotation R
    R = u_to_quat_to_R(u)
    # Transformation: rotate and translate M for X2
    M_RT = M @ R + T
    # Likelihood
    with pyro.plate("plate_students", N*3,dim=-1):
        pyro.sample("X1", dist.StudentT(1, M.view(-1), U),obs=X1)
        pyro.sample("X2", dist.StudentT(1, M_RT.view(-1), U), obs=X2)
```

Fig. 3: Code fragment from the THESEUS-PP implementation in Pyro. *pyro.sample* calls a primitive stochastic function from which a named sample is drawn. *expand_by* specifies the shape of the batch that is to be drawn from the distribution. *pyro.plate* declares the variables within a tensor dimension as conditionally independent, while *to_event* declares them as dependent.

1ADZ

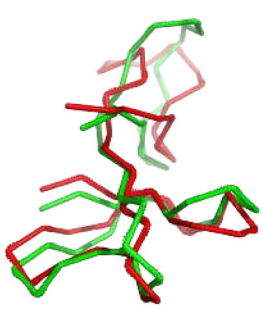
(a) Kabsch-RMSD

(b) Theseus-PP

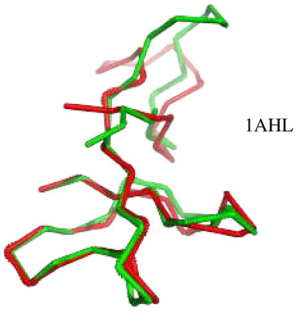



(c)

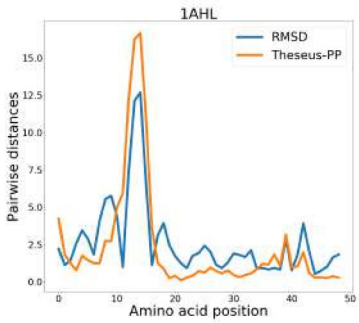
Fig. 4: Protein superposition for two conformations of protein 1ADZ obtained from (a) conventional RMSD superimposition and (b) THESEUS-PP. The protein in green is rotated ($X_2$). The images are generated with PyMOL [14]. Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.



(a) Kabsch-RMSD

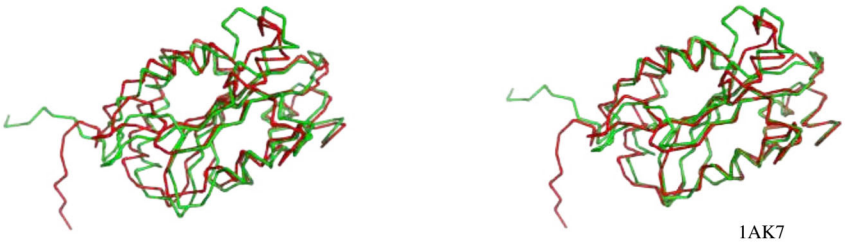



(b) Theseus-PP



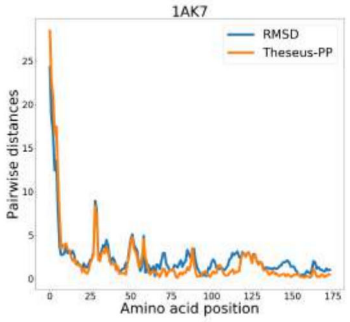

(c)

Fig. 5: Protein superposition for two conformations of protein 1AHL obtained from (a) conventional RMSD superimposition and (b) THESEUS-PP. The protein in green is rotated ($X_2$). The images are generated with PyMOL [14]. Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.




(a) Kabsch-RMSD

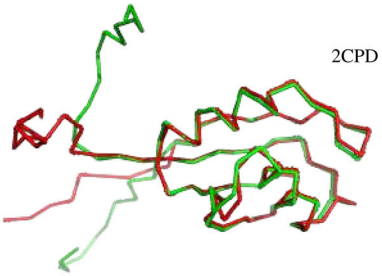
(b) Theseus-PP



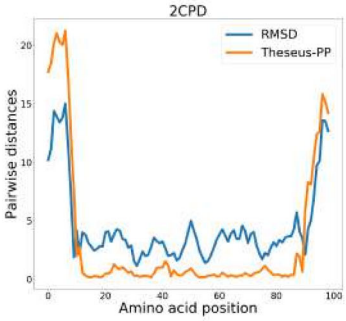

(c)

Fig. 6: Protein superposition for two conformations of protein 1AK7 obtained from (a) conventional RMSD superimposition and (b) THESEUS-PP. The protein in green is rotated ($X_2$). The images are generated with PyMOL [14]. Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.

(a) Kabsch-RMSD

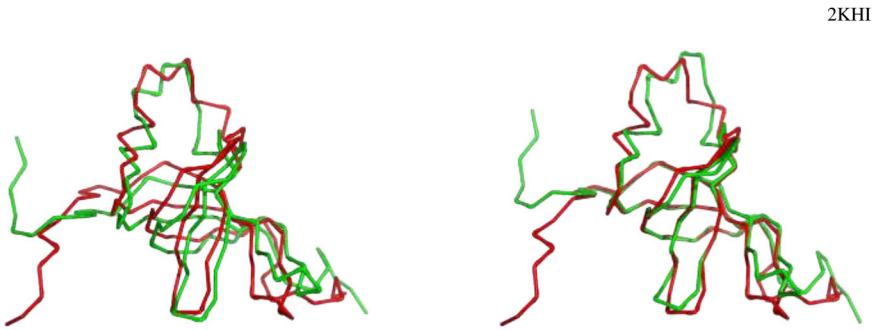



(b) Theseus-PP



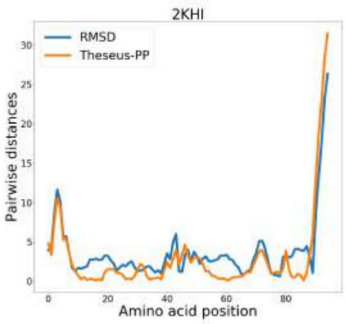

(c)

Fig. 7: Protein superposition for two conformations of protein 2CPD obtained from (a) conventional RMSD superimposition and (b) THESEUS-PP. The protein in green is rotated ($X_2$). The images are generated with PyMOL [14]. Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.




(a) Kabsch-RMSD

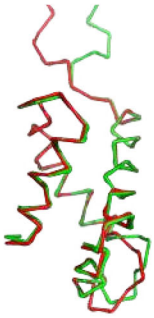
(b) Theseus-PP




(c) Theseus-PP

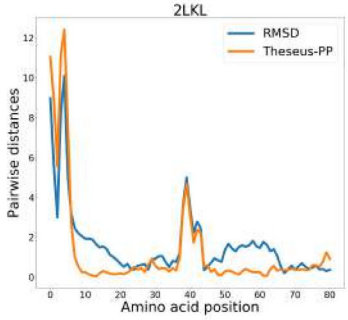
Fig. 8: Protein superposition for two conformations of protein 2KHI obtained from (a) conventional RMSD superimposition and (b) THESEUS-PP. The protein in green is rotated ($X_2$). The images are generated with PyMOL [14]. Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.

(a) Kabsch-RMSD



(b) Theseus-PP

2LKL




(c)

Fig. 9: Protein superposition for two conformations of protein 2LKL obtained from (a) conventional RMSD superimposition and (b) THESEUS-PP. The protein in green is rotated ($X_2$). The images are generated with PyMOL [14]. Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.

# 14 Paper 2: Bayesian protein superposition using Hamiltonian Monte Carlo

**Authors:** Lys Sanz Moreta, Ahmad Salim Al-Sibahi and Thomas Hamelryck

**Motivation:** Computing the uncertainty over the superposition parameters and testing the versatility of the rotation matrix to capture ambiguous protein superpositions had not been tested. Therefore we proposed a new version of Theseus-PP named Theseus-HMC where we were able to see the complete potential of the model under Hamiltonian Monte Carlo inference methodology.

# Bayesian protein superposition using Hamiltonian Monte Carlo

Lys Sanz Moreta[1*], Ahmad Salim Al-Sibahi[1*], and Thomas Hamelryck[1,4*]

[1] Department of Computer Science. University of Copenhagen, Denmark
[4] The Bioinformatics Centre. Section for Computational and RNA Biology. University of Copenhagen, Denmark

[*] Corresponding authors.
moreta@di.ku.dk (Lys Sanz Moreta),
ahmad@di.ku.dk (Ahmad Salim Al-Sibahi),
thamelry@bio.ku.dk (Thomas Hamelryck)

***Abstract*—Optimally superimposing protein structures is essential to study their structure, function, dynamics and evolution. We present THESEUS NUTS (No U-Turn Sampler), a Bayesian version of the THESEUS model [1]–[3] which relies on maximum likelihood estimation. The probabilistic model interprets each protein as a rotated and translated noisy observation of a latent mean structure. Unlike conventional methods [4], THESEUS takes into account the differences in correlations between the atoms in the structure. This paper extends the previous THESEUS MAP (Maximum A Posteriori) model, [5] to full Bayesian inference by making use of the iterative NUTS [6], a Hamiltonian Monte Carlo method. The model delivers consistent results and is computationally efficient thanks to its implementation in the probabilistic programming language NumpPyro [7], [8] which in turn relies upon JAX [9], a system for high-performance machine learning.**

***Index Terms*—protein superposition, Bayesian modelling, probabilistic programming, NUTS, Hamiltonian Monte Carlo, protein structure superposition**

## I. Introduction

Superimposing proteins optimally is crucial to compare their structures. The standard method minimizes the root mean squared deviation (RMSD) of the distances between paired atoms. This approach centers the proteins to the origin and calculates the rotation matrix by singular value decomposition [4] or quaternion algebra [10], [11]. These methods presume that all atoms positions share equal variance (homoscedasticity). This becomes problematic in the case of proteins with flexible loops or flexible terminal regions, where some of the atoms can exhibit high variance. The THESEUS model [1], [2] solved this issue by allowing identification of atoms with higher or lower variance (heteroscedasticity), corresponding to variable and conserved regions.

Previously we implemented the THESEUS model in the probabilistic programming language Pyro [8] (THESEUS MAP). This made it possible to calculate a maximum a posteriori (MAP) estimate using automated stochastic variational inference (SVI) and suitable priors [5].

Here, we present a fully Bayesian version of THESEUS based on iterative NUTS sampling. The model is implemented in the deep probabilistic programming language NumPyro which facilitates high-performance machine learning research thanks to its JAX backend [9]. JAX is an extensible system for composable function transformations which allows the implementation of sophisticated algorithms with high performance in Python.

## II. Methods

### A. Algorithm summary

THESEUS considers the observed structures, $X_1$ and $X_2$, as distributed according to a multivariate matrix normal distribution with latent mean structure $\mathbf{M}$, with $N$x3 size, and covariance matrices $\mathbf{U}$ and $\mathbf{V}$. $\mathbf{X}_2$ is rotated ($\mathbf{R}$) and translated ($\mathbf{T}$) to achieve the optimal superposition 2. The rotation matrix is represented by quaternions in order to formulate a uniform prior over the space of rotations.

$$\mathbf{X}_1 \sim \mathcal{MN}(\mathbf{M}, \mathbf{U}, \mathbf{V}). \tag{1}$$

$$\mathbf{X}_2 \sim \mathcal{MN}(\mathbf{MR} - \mathbf{1}_N\mathbf{T}, \mathbf{U}, \mathbf{V}). \tag{2}$$

In practice, the matrix-normal distributions of $\mathbf{X}_1$ and $\mathbf{X}_2$ reduce to a product of multivariate normal distributions. For details on the model and its prior distribution we refer to [5].

### B. Hamiltonian Monte Carlo methods

Hamiltonian Monte Carlo (HMC) is a Markov chain Monte Carlo (MCMC) algorithm that replaces random walk dynamics with Hamiltonian dynamics which rely on gradient information to perform the sampling. The No-U-Turn Sampler (NUTS) [12] is an HMC extension that allows automatically tuning the required hyperparameters (the step size and the number of steps).

For our model, we rely on the iterative NUTS implementation in Numpyro to perform the sampling. Iterative NUTS [6] improves on conventional NUTS [12] by replacing recursion with iteration in the construction of the binary tree used in the proposal. The iterative approach can be implemented efficiently in JAX [9]. JAX can in turn construct a graph representing the computation, and compile it for efficient execution on modern hardware using optimizations based on linear algebra. These optimizations were critical in making NUTS sampling scale to high-dimensional protein systems with hundreds of atoms, and allow us to retrieve a large number of samples for the posterior distribution in seconds.

### C. Inference details

The posterior distribution was approximated by 1000 samples after a burn-in period of 500 samples. The parameters of the inference method were chosen in order to optimize speed: the tree depth was restricted to 10 nodes, the acceptance probability of the samples was set to 0.8 or higher and the chain was initialized by a prior based on the median of 15 samples. We ran a single Markov chain.

## III. Materials

The algorithm was tested on several proteins from the RCSB protein database [13] whose structures were determined by Nuclear Magnetic Resonance (NMR). Such files typically contain several models of the same protein. These models represent the structural dynamics of the protein in an aqueous medium, and thus provide structural snapshots that reveal both conserved and variable regions. This makes them challenging targets for conventional RMSD superposition. We used the following structures: 1AB2, 1AHL, 1JE3, 1RLP, 1ZWG, 2HF5, 2KHI and 2LMP, which vary from sizes of 35 to 576 residues.

## IV. Results

The computations where carried out on an Intel© Xeon © Gold 6136 CPU @ 3.00GHz and an Nvidia graphic card (Quadro RTX 6000). The model was implemented both in Pyro and Numpyro, re-running them for comparison. The latter implementation was on average approximately 1000 times faster due to the implementation of iterative NUTS within the JAX environment. As the computational speed of the Numpyro implementation was much higher on the CPU than on the GPU, we focus here on the former . The computational times register in I are an average compilation of 10 runs for the Numpyro model. The Pyro version was only run once due to its low computational speed.

An example of a pair of superimposed structures is shown in 1. For comparison, the superposition resulting from the conventional RMSD method [4], calculated using Biopython [14], is shown on 1a. The THESEUS MAP superposition is shown in 1b. Finally, the THESEUS NUTS superposition is displayed in 1c. These images illustrate the differences between these different methods and their abilities to handle regions with high and low variability. Supplementary figures of superimposed structures exhibiting different degrees of variability can be found in the Appendix.

TABLE I
RESULTS OF APPLYING THESEUS NUTS TO THE TEST STRUCTURES. FIRST COLUMN: PDB IDENTIFIER. SECOND COLUMN: THE NUMBER OF $C_\alpha$ ATOMS USED IN THE SUPERPOSITION. THIRD COLUMN: THE MODEL IDENTIFIERS. FOURTH COLUMN: CPU RUNNING TIME FOR THE NUMPYRO MODEL. FIFTH COLUMN: GPU RUNNING TIME FOR PYRO MODEL.

| PBD ID | Length $C_\alpha$ | Protein Models | CPU Time (Numpyro) | GPU Time (Pyro) |
|---|---|---|---|---|
| 1AB2 | 109 | 0 and 3 | 24.3s | 1h07m35s |
| 1AHL | 49 | 0 and 2 | 19.1s | 1h05m44s |
| 1JE3 | 97 | 0 and 1 | 22s | 1h05m35s |
| 1RLP | 65 | 0 and 5 | 26s | 58m48s |
| 1ZWG | 35 | 0 and 3 | 20s | 52m30s |
| 2HF5 | 68 | 0 and 3 | 30.9s | 1h04m37s |
| 2KHI | 95 | 0 and 1 | 25.1s | 1h05m24s |
| 2LMP | 576 | 0 and 3 | 154.5s | 57m25s |

## V. Conclusion

Here, we present a Bayesian model for protein structure superposition implemented in the deep probabilistic programming language Numpyro [7], [8] with JAX [9] as its back-end . This is the first time that the full Bayesian posterior over the parameters of protein superposition is inferred.

The results achieved using Bayesian inference suggest that THESEUS could be potentially used as a suitable error model for probabilistic protein structure prediction. The THESEUS model has the potential to distinguish among partially correct and utterly incorrect predictions when used as a potential likelihood model. Our results show that it is capable of delivering a rich posterior with multiple or unique superposition solutions, as seen for example in Fig. 1 or Fig. 4 in the Supplementary section, respectively.

## Contributions and Acknowledgements

Implemented algorithm in NumPyro and Numpy JAX: *LSM* and *ASA*. Wrote article: *LSM*, *TH*, *ASA*. Performed experiments: *LSM*. Designed experiments: *TH*. *LSM* and *ASA* acknowledge support from the Independent Research Fund Denmark (grant: "Resurrecting ancestral proteins in silico to understand how cancer drugs work") and "Deep Probabilistic Programming for Protein Structure Prediction" grant from Independent Research Fund Denmark(DFF), respectively. We thank the Numpyro team for bench-marking the model and further suggestions on how to increase the speed performance.

## Data and Software Availability

NumPyro [7], [8] and Numpy JAX [9] based code is available at https://github.com/LysSanzMoreta/BayesTheseus-PP

## References

[1] D. L. Theobald and D. S. Wuttke, "Empirical Bayes hierarchical models for regularizing maximum likelihood estimation in the matrix Gaussian Procrustes problem," *PNAS*, vol. 103, pp. 18 521–18 527, 2006.
[2] D. L. Theobald and P. A. Steindel, "Optimal simultaneous superpositioning of multiple structures with missing data," *Bioinformatics*, vol. 28, pp. 1972–1979, 2012.
[3] K. Mardia and I. Dryden, "The statistical analysis of shape data," *Biometrika*, vol. 76, no. 2, pp. 271–281, 1989.
[4] W. Kabsch, "A discussion of the solution for the best rotation to relate two sets of vectors," *Acta Cryst. A*, vol. 34, pp. 827–828, 1978.

[5] L. S. Moreta, A. S. Al-Sibahi, D. Theobald, W. Bullock, B. N. Rommes, A. Manoukian, and T. Hamelryck, "A probabilistic programming approach to protein structure superposition," in *2019 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, July 2019, pp. 1–5.

[6] N. P. Du Phan, "Iterative NUTS," https://github.com/pyro-ppl/numpyro/wiki/Iterative-NUTS, 2019.

[7] D. Phan, N. Pradhan, and M. Jankowiak, "Composable effects for flexible and accelerated probabilistic programming in numpyro," *arXiv preprint arXiv:1912.11554*, 2019.

[8] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. A. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep universal probabilistic programming," *J. Mach. Learn. Res.*, vol. 20, pp. 28:1–28:6, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-403.html

[9] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

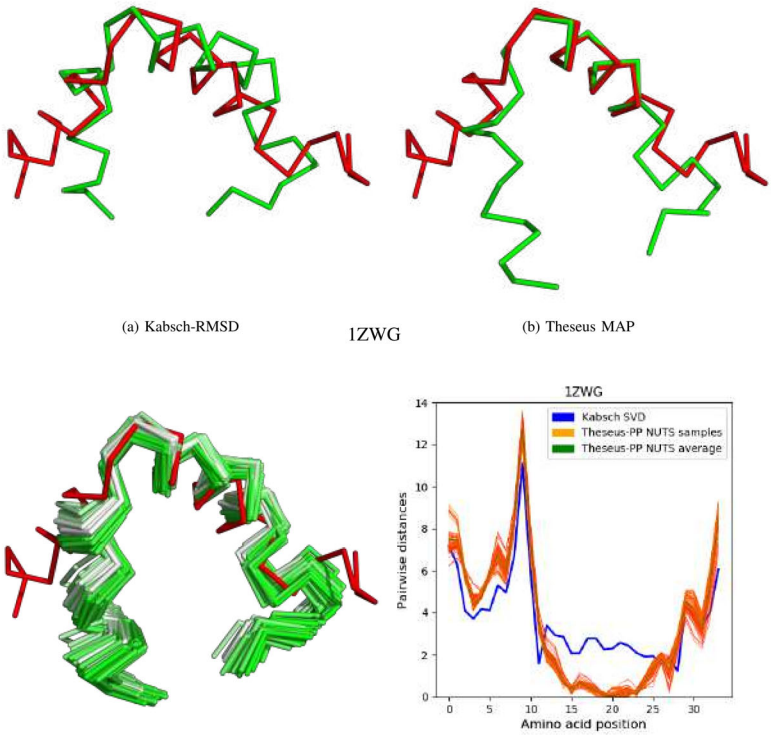
[10] B. Horn, "Closed-form solution of absolute orientation using unit quaternions," *J. Opt. Soc. Am. A*, vol. 4, pp. 629–642, 1987.

[11] E. Coutsias, C. Seok, and K. Dill, "Using quaternions to calculate rmsd," *J. Comp. Chem.*, vol. 25, pp. 1849–1857, 2004.

[12] M. D. Hoffman and A. Gelman, "The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014.

[13] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic acids research*, vol. 28, no. 1, pp. 235–242, 2000.

[14] P. J. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski *et al.*, "Biopython: freely available python tools for computational molecular biology and bioinformatics," *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, 2009.

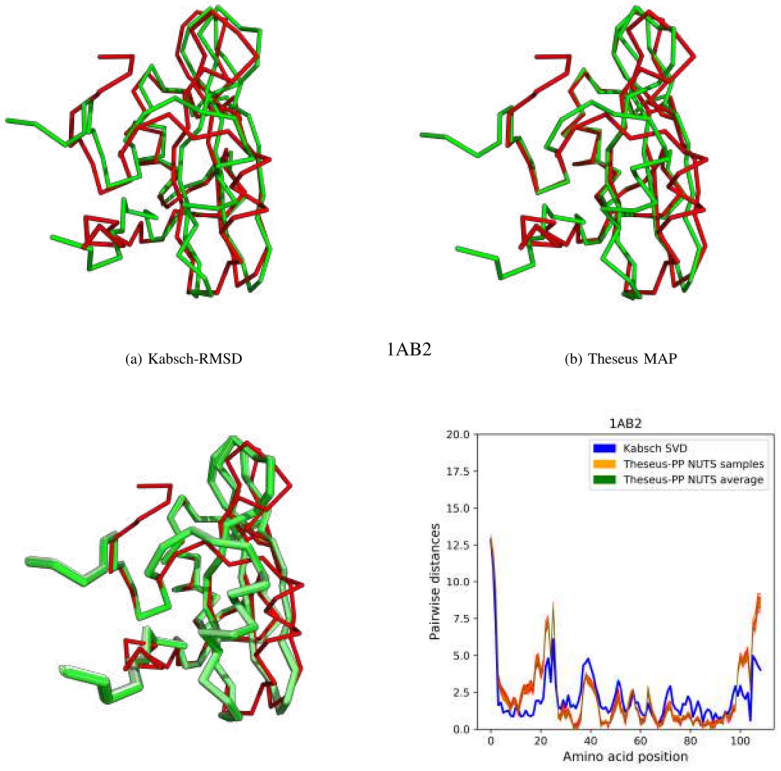



(a) Kabsch-RMSD (b) Theseus MAP

(c) Theseus NUTS (d) Pairwise Distances NUTS

Fig. 1. Protein superposition for two conformations of protein 1ZWG obtained from (a) conventional RMSD superimposition and (b) THESEUS MAP and (c) THESEUS NUTS. The protein in green is rotated ($X_2$). Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.

## VI. APPENDIX



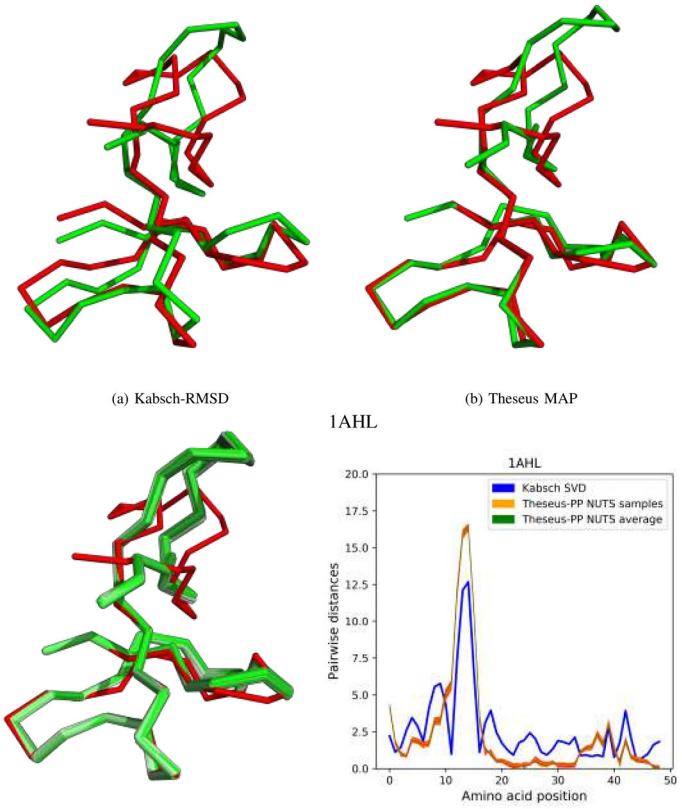

(a) Kabsch-RMSD

(b) Theseus MAP

(c) Theseus NUTS

(d) Pairwise Distances NUTS

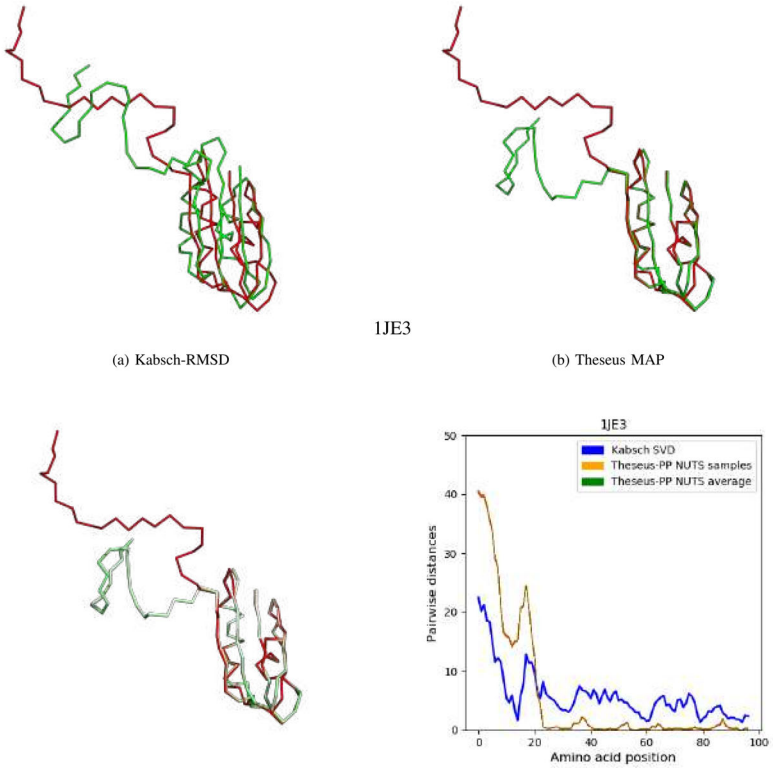
Fig. 2. Superposition of two conformations of protein 1AB2 obtained from (a) conventional RMSD superimposition and (b) THESEUS MAP and (c) THESEUS NUTS. The protein in green is rotated ($X_2$). Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.




(a) Kabsch-RMSD (b) Theseus MAP

(c) Theseus NUTS (d) Pairwise Distances NUTS

Fig. 3. Protein superposition of two conformations of protein 1AHL obtained from (a) conventional RMSD superimposition and (b) THESEUS MAP and (c) THESEUS NUTS. The protein in green is rotated ($X_2$). Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.



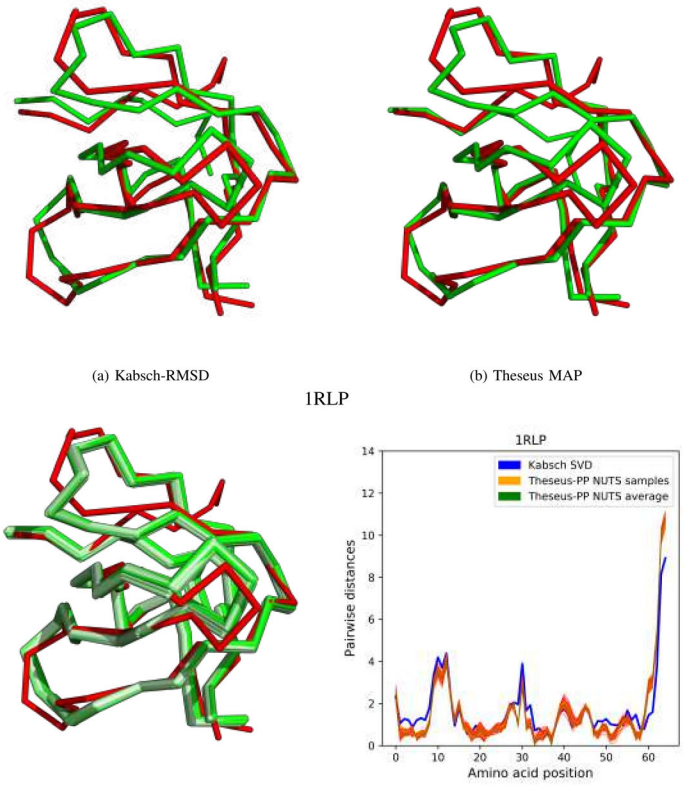

(a) Kabsch-RMSD

(b) Theseus MAP

(c) Theseus NUTS

(d) Pairwise Distances NUTS

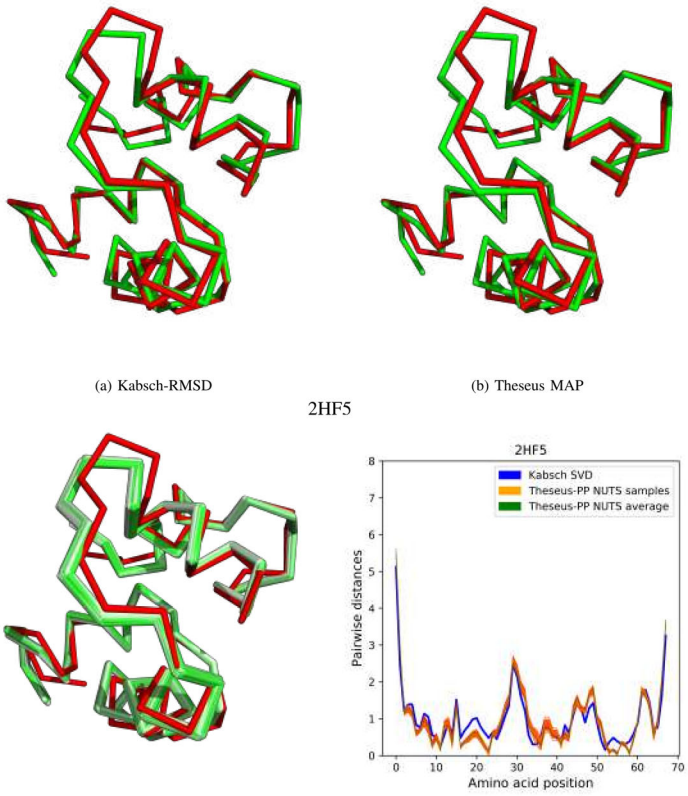
Fig. 4. Superposition of two conformations of protein 1JE3 obtained from (a) conventional RMSD superimposition and (b) THESEUS MAP and (c) THESEUS NUTS. The protein in green is rotated ($X_2$). Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.




(a) Kabsch-RMSD (b) Theseus MAP

(c) Theseus NUTS (d) Distances NUTS

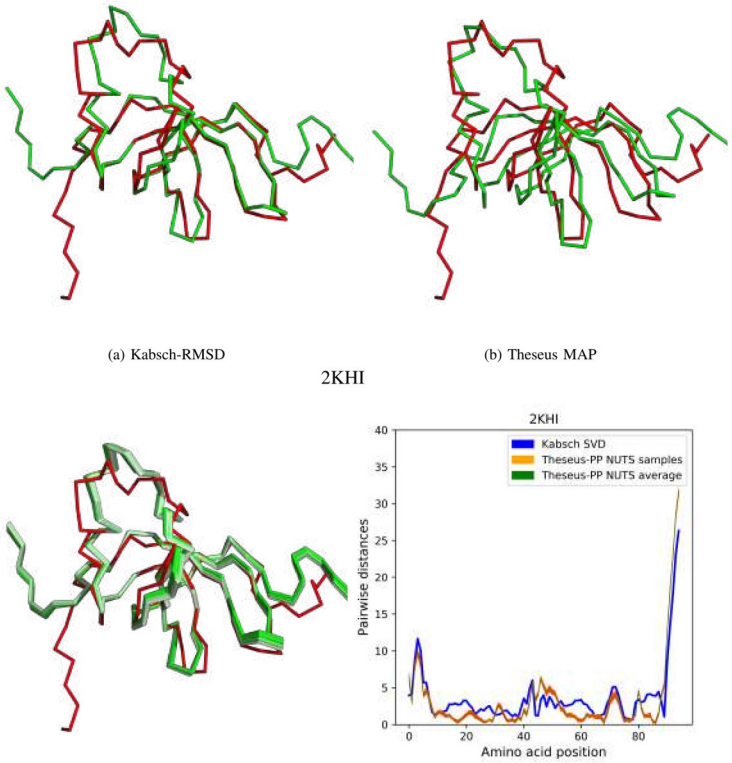
Fig. 5. Superposition of two conformations of protein 1RLP obtained from (a) conventional RMSD superimposition and (b) THESEUS MAP and (c) THESEUS NUTS. The protein in green is rotated ($X_2$). Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.




(a) Kabsch-RMSD (b) Theseus MAP

(c) Theseus NUTS (d) Distances NUTS

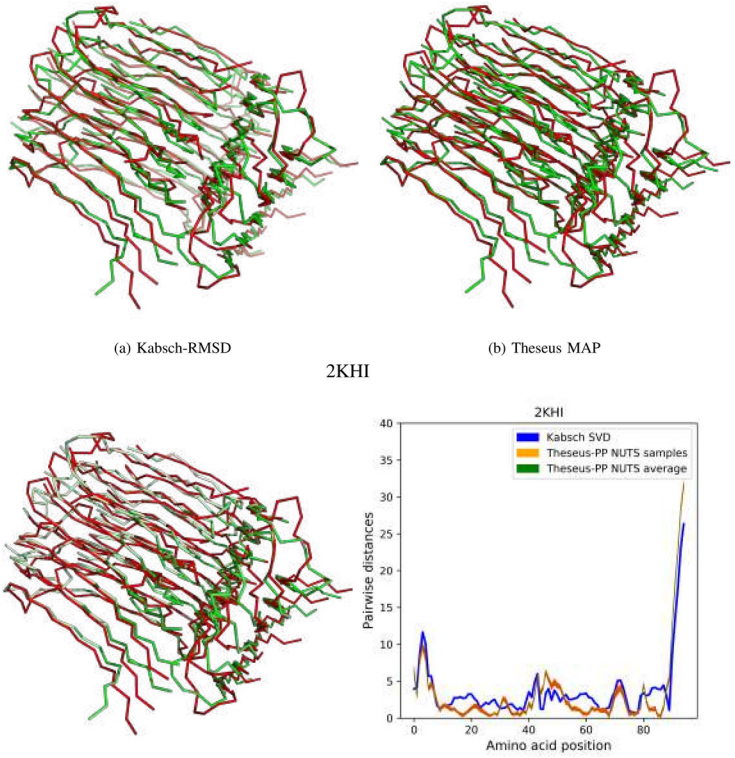
Fig. 6. Superposition of two conformations of protein 2HF5 obtained from (a) conventional RMSD superimposition and (b) THESEUS MAP and (c) THESEUS NUTS. The protein in green is rotated ($X_2$). Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.




(a) Kabsch-RMSD (b) Theseus MAP

(c) Theseus NUTS (d) Distances NUTS

Fig. 7. Superposition of two conformations of protein 2KHI obtained from (a) conventional RMSD superimposition and (b) THESEUS MAP and (c) THESEUS NUTS. The protein in green is rotated ($X_2$). Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.




(a) Kabsch-RMSD (b) Theseus MAP

(c) Theseus NUTS (d) Distances NUTS

Fig. 8. Superposition of two conformations of protein 2KHI obtained from (a) conventional RMSD superimposition and (b) THESEUS MAP and (c) THESEUS NUTS. The protein in green is rotated ($X_2$). Graph (c) shows the pairwise distances (in Å) between the $C_\alpha$ coordinates of the structure pairs. The blue and orange lines represent RMSD and THESEUS-PP superposition, respectively.

# 15 Paper 3: Efficient Generative Modelling of Protein Structure Fragments using a Deep Markov Model

**Authors:** Christian B.Thygesen, Ahmad Salim Al-Sibahi, Christian S. Steenmanns, Lys Sanz Moreta, Anders B.Sørensen, and Thomas Hamelryck.

**Motivation:** Generation of *ab-initio* protein fragment libraries is a challenging task due to the insufficient ability of the state-of-the-art methods to capture uncertainty over the generated predictions. Achieving a stage of substantial variability over the predictions is needed to account for the noisiness within the experimental data used for training and the intrinsic dynamism of the protein structure. We developed a new probabilistic model, BIFROST, based on the continuous Deep Markov Model [5, 162] that solely relies on learning the distribution of the dihedral angles from the protein backbone. BIFROST offers on par accuracy results with ROSETTA [24, 24] with better computational performance. This model is designed to be extended with the Theseus-PP error likelihood function in means of improving its global structure prediction capabilities.

# Efficient Generative Modelling of Protein Structure Fragments using a Deep Markov Model

**Christian B. Thygesen** [1 2] **Ahmad Salim Al-Sibahi** [1] **Christian S. Steenmanns** [2] **Lys S. Moreta** [1]
**Anders B. Sørensen** [* 2] **Thomas Hamelryck** [* 1 3]

## Abstract

Fragment libraries are often used in protein structure prediction, simulation and design as a means to significantly reduce the vast conformational search space. Current state-of-the-art methods for fragment library generation do not properly account for aleatory and epistemic uncertainty, respectively due to the dynamic nature of proteins and experimental errors in protein structures. Additionally, they typically rely on information that is not generally or readily available, such as homologous sequences, related protein structures and other complementary information. To address these issues, we developed BIFROST, a novel take on the fragment library problem based on a Deep Markov Model architecture combined with directional statistics for angular degrees of freedom, implemented in the deep probabilistic programming language Pyro. BIFROST is a probabilistic, generative model of the protein backbone dihedral angles conditioned solely on the amino acid sequence. BIFROST generates fragment libraries with a quality on par with current state-of-the-art methods at a fraction of the run-time, while requiring considerably less information and allowing efficient evaluation of probabilities.

## 1. Introduction

Fragment libraries (Jones & Thirup, 1986) find wide application in protein structure prediction, simulation, design and experimental determination (Trevizani et al., 2017; Chikenji et al., 2006; Boomsma et al., 2012). Predicting the fold of a protein requires evaluating a conformational space that is too vast for brute-force sampling to be feasible (Levinthal, 1969). Fragment libraries are used in a divide-and-conquer approach, whereby a full length protein is divided into a manageable sub-set of shorter stretches of amino acids for which backbone conformations are sampled. Typically, sampling is done using a finite set of fragments derived from experimentally determined protein structures. Fragment libraries are used in state-of-the-art protein structure prediction frameworks such as Rosetta (Rohl et al., 2004), I-TASSER (Roy et al., 2010), and AlphaFold (Senior et al., 2019).

Generally, knowledge-based methods for protein structure prediction follow two main strategies: homology (or template-based) modelling (Eswar et al., 2006; Šali & Blundell, 1993; Song et al., 2013) and *de novo* modelling (Rohl et al., 2004). Both approaches assume that the native fold of a protein corresponds to the minimum of a physical energy function and make use of statistics derived from a database of known proteins structures (Alford et al., 2017; Leaver-Fay et al., 2013). Whereas homology modelling relies on the availability of similar structures to limit the search space, knowledge-based *de novo* protocols require extensive sampling of the conformational space of backbone angles (figure 1).

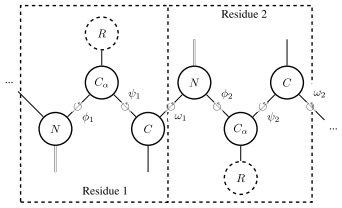



*Figure 1.* Schematic of the three dihedral angles ($\phi$, $\psi$, and $\omega$) that parameterise the protein backbone. $R$ represents the side chain.

To overcome the shortcomings of either strategy, modelling tools like Rosetta (Rohl et al., 2004) use a combined approach of extensive sampling and prior information. Rosetta employs simulated annealing of backbone conformations

[*]Equal contribution [1]Department of Computer Science, University of Copenhagen, Copenhagen, Denmark [2]Evaxion Biotech, Copenhagen, Denmark [3]Department of Biology, University of Copenhagen, Copenhagen, Denmark. Correspondence to: Christian B. Thygesen <christiank.thygesen@di.ku.dk>, Thomas Hamelryck <thamelry@bio.ku.dk>.

*Proceedings of the $38^{th}$ International Conference on Machine Learning*, PMLR 139, 2021. Copyright 2021 by the author(s).

according to an energy function (Alford et al., 2017), while reducing the conformational space by sampling fragments of typically 3 or 9 amino acids at a time (Simons et al., 1997).

Fragments are typically extracted from experimentally determined protein structures in the Protein Data Bank (Berman et al., 2000) and used in prediction based on similarities in sequence and sequence-derived features (Gront et al., 2011; Kalev & Habeck, 2011; Santos et al., 2015; De Oliveira et al., 2015; Trevizani et al., 2017; Wang et al., 2019). Generative probabilistic models of protein backbone angles (Hamelryck et al., 2006; Boomsma et al., 2008; Bhattacharya et al., 2016; Edgoose et al., 1998; Li et al., 2008; Lennox et al., 2010) offer an alternative way to construct fragment libraries and aim to represent the associated epistemic and aleatory uncertainty. In this case, epistemic uncertainty is due to experimental errors from the determination of protein structures, while aleatory or inherent uncertainty is due to the dynamic nature, or flexibility, of proteins (Best, 2017).

Here, we present BIFROST - Bayesian Inference for FRagments Of protein STructures - a deep, generative, probabilistic model of protein backbone angles that solely uses the amino acid sequence as input. BIFROST is based on an adaptation of the Deep Markov Model (DMM) architecture (Krishnan et al., 2017) and represents the angular variables ($\phi$ and $\psi$) in a principled way using directional statistics (Mardia & Jupp, 2008). Finally, BIFROST makes it possible to evaluate the probability of a backbone conformation given an amino acid sequence, which is important for applications such as sampling the conformational space of proteins with correct statistical weights in equilibrium simulations (Boomsma et al., 2014).

## 2. Background and related work

**Probabilistic, generative models of local protein structure** Most generative, probabilistic models of local protein structure are Hidden Markov Models (HMMs) that represent structure and sequence based on the assumption of a Markovian structure (Hamelryck et al., 2012). The first such models did not include the amino acid sequence (Edgoose et al., 1998), discretised the angular variables (Bystroff et al., 2000), or used continuous, but lossy representations (Camproux et al., 1999; Hamelryck et al., 2006), making sampling of conformations with atomic detail problematic. These early models are thus probabilistic but only approximately "generative" at best. TorusDBN (Boomsma et al., 2008) was the first joint model of backbone angles and sequence that properly accounted for the continuous and angular nature of the data. Others introduced richer probabilistic models of local protein structure including Dirichlet Process mixtures of HMMs (DPM-HMMs) Lennox et al. (2010) and Conditional Random Fields (CRFs) (Zhao et al., 2010; 2008). As far as we know, BIFROST is the first deep generative model of local protein structure that aims to quantify the associated aleatory and epistemic uncertainty using an (approximate) Bayesian posterior.

**Deep Markov Models** The DMM, introduced in (Krishnan et al., 2017), is a generalisation of the variational autoencoder (VAE) (Kingma & Welling, 2014) for sequence or time series data. Related stochastic sequential neural models were reported by Fraccaro et al. (2016) and Chung et al. (2015). Published applications of DMMs include natural language processing tasks (Khurana et al., 2020), inference of time series data (Zhi-Xuan et al., 2020), and human pose forecasting (Toyer et al., 2017). Our application of the DMM and the modifications made to the standard model will be described in section 3.3.

## 3. Methods

### 3.1. Data set

BIFROST was trained on a data set of fragments derived from a set of 3733 proteins from the *cullpdb* data set (Wang & Dunbrack, 2005). Quality thresholds were (i) resolution $< 1.6$Å, (ii) R-factor $< 0.25$, and (iii) a sequence identity cutoff of $20\%$. For the purpose of reliable evaluation, sequences with $> 20\%$ identity to CASP13 targets were removed from the dataset.

Fragments containing angle-pairs in disallowed regions of the Ramachandran plot (Ramachandran et al., 1963) were removed using the Ramalyze function of the crystallography software PHENIX (Liebschner et al., 2019). The resulting data set consisted of $\sim 186000$ 9-mer fragments. Prior to training, the data was randomly split into train, test, and validation sets with a $60/20/20\%$ ratio.

### 3.2. Framework

The presented model was implemented in the deep probabilistic programming language Pyro, version 1.3.0 (Bingham et al., 2019) and Pytorch version 1.4.0 (Paszke et al., 2019). Training and testing were carried out on a machine equipped with an Intel Xeon CPU E5-2630 and Tesla M10 GPU. The model trains on a single GPU and converges after 150 epochs for a total training time of approximately 34 hours.

### 3.3. Model

BIFROST consists of a DMM (Krishnan et al., 2017) with an architecture similar to an Input-Output HMM (IO-HMM) (Bengio & Frasconi, 1995). The model employs the Markovian structure of an HMM, but with continuous, as opposed to discrete, latent states ($\mathbf{z}$) and with *transition and emission neural networks* instead of transition and emission matrices.

Consequently, the latent states are iteratively transformed using the transition neural network, such that the value of the current latent state depends on the previous state and the (processed) amino acid information at that position (figure 2).

Observed angles ($\phi$ and $\psi$) are generated from the latent state sequence by applying an *emitter neural network* at each position (figure 2). Since the backbone angle $\omega$ is most often narrowly distributed around $180°$, this degree of freedom is not included in the current version of BIFROST.

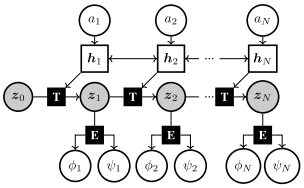



*Figure 2.* The BIFROST model. Grey nodes are latent random variables, white circular nodes are observed variables, white rectangular nodes represent hidden states from a bidirectional Recurrent Neural Network (RNN) $H$, and black squares represent neural networks. $E$ and $T$ denote the emitter and the transition network, respectively.

The structure of the model is shown in figure 2. For notational simplicity, the sequence of $\phi$ and $\psi$ pairs will be denoted by $\mathbf{x}$. The joint distribution of the latent variable $\mathbf{z}$ and the angles $\mathbf{x}$ conditioned on the amino acid sequence $\mathbf{a}$ with length $N$ of the graphical model in figure 2 factorises as

$$p(\mathbf{z}, \mathbf{x}|\mathbf{a}) = \prod_{n=1}^{N} p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{h}_n(\mathbf{a}))p(\mathbf{x}_n|\mathbf{z}_n) \tag{1}$$

where $\boldsymbol{h}_n(\mathbf{a})$ is the deterministic hidden state generated at position $n$ by a bidirectional RNN $H$ with parameters $\boldsymbol{\theta}_H$ running across the amino acid sequence. The bidirectional RNN incorporates information from amino acids upstream and downstream of position $n$. The initial latent state $\mathbf{z}_0$ is treated as a trainable parameter and is thus shared for all sequences.

The transition densities are given by a multivariate Gaussian distribution,

$$p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{h}_n(\mathbf{a})) = \mathcal{N}(\boldsymbol{\mu}_T(\mathbf{z}_{n-1}, \mathbf{h}_n(\mathbf{a})), \boldsymbol{\Sigma}_T(\mathbf{z}_{n-1}, \mathbf{h}_n(\mathbf{a}))) \tag{2}$$

where the mean vector ($\boldsymbol{\mu}_T$) and the (diagonal) covariance matrix ($\boldsymbol{\Sigma}_T$) are given by a neural network $T$ parameterised by $\boldsymbol{\theta}_T$.

The emission densities are given by a *bivariate periodic student-T distribution* (Pewsey et al., 2007) (section 3.5) such that

$$p(\mathbf{x}_n|\mathbf{z}_n) = \mathcal{T}(\mathbf{x}_n|\nu_E(\mathbf{z}_n), \boldsymbol{\mu}_E(\mathbf{z}_n), \boldsymbol{\Sigma}_E(\mathbf{z}_n)) \tag{3}$$

where the single, shared degree of freedom ($\nu_E$), the vector of two means ($\boldsymbol{\mu}_E$), and the $2\times2$ diagonal covariance matrix ($\boldsymbol{\Sigma}_E$) of the distribution are given by a neural network $E$ parameterised by $\boldsymbol{\theta}_E$.

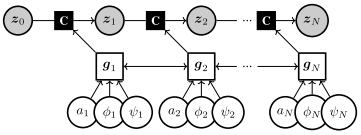



*Figure 3.* Variational distribution for approximating the posterior. Grey nodes are latent random variables, white circular nodes are observed variables, white rectangular nodes represent hidden states from a bidirectional RNN $G$, while black squares represent neural networks. $C$ denotes the combiner network.

### 3.4. Estimation

In order to perform inference of the intractable posterior, we introduce a variational distribution or *guide* $q$ (Kingma & Welling, 2019) (figure 3), which makes use of a *combiner neural network* $C$ parameterised by $\boldsymbol{\zeta}_C$,

$$q(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{a}, \mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_C(\mathbf{z}_{n-1}, \mathbf{g}_n(\mathbf{a}, \mathbf{x})), \boldsymbol{\Sigma}_C(\mathbf{z}_{n-1}, \mathbf{g}_n(\mathbf{a}, \mathbf{x}))) \tag{4}$$

where $\boldsymbol{g}_n(\boldsymbol{a}, \boldsymbol{x})$ is the deterministic hidden state generated at position $n$ by a bidirectional RNN $G$ with parameters $\boldsymbol{\zeta}_G$ running across the amino acid sequence $\boldsymbol{a}$ and the angles $\boldsymbol{x}$.

For the parameters of the neural networks $(\boldsymbol{\zeta}_C, \boldsymbol{\zeta}_G, \boldsymbol{\theta}_T, \boldsymbol{\theta}_E, \boldsymbol{\theta}_H)$, point estimates are obtained using Stochastic Variational Inference (SVI), which optimises the Evidence Lower Bound (ELBO) using stochastic gradient descent (SGD) (Kingma & Welling, 2014; 2019). The ELBO variational objective is given by

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\zeta}}(\boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\zeta}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{a})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x}|\boldsymbol{a}) - \log q_{\boldsymbol{\zeta}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{a})\right] \tag{5}$$

where $\boldsymbol{\zeta} = (\boldsymbol{\zeta}_C, \boldsymbol{\zeta}_G)$ and $\boldsymbol{\theta} = (\boldsymbol{z}_0, \boldsymbol{\theta}_T, \boldsymbol{\theta}_E, \boldsymbol{\theta}_H)$ are the parameters of the guide and the model, respectively.

### 3.5. Periodic student T distribution

As angle-pairs are periodic values, i.e. distributed on a torus (Boomsma et al., 2008), they need to be modelled by an appropriate periodic distribution. Traditionally, angles are assumed distributed according to the von Mises distribution, which is defined by a mean that can be any real number and a concentration parameter, which can be any positive number. SVI showed poor performance when the von Mises distribution was used. Here, we circumvent this by representing the likelihood of the angles by a student T distribution that is wrapped around a circle (Pewsey et al., 2007). This allows for appropriate modelling of the periodicity of the angles, while being more robust with regards to outlier issues than the von Mises distribution due to the wider tails of the T distribution. It should be noted as well that Pewsey et al. (2007) showed that the wrapped student T distribution can approximate the von Mises distribution closely.

### 3.6. Neural network architecture overview

The overall architecture is based on the originally proposed DMM (Krishnan et al., 2017) with modifications. The main difference is the addition of an RNN $H$ in the model that processes the amino acid sequence $\mathbf{a}$, thus providing explicit conditioning on the amino acid sequence. A similar architecture was used by Fraccaro et al. (2016) for time series. In the guide, a second RNN $G$ is used that processes the angles and the amino acid sequence during training. The initial values for both RNNs are treated as trainable parameters. In addition to the RNNs, the model contains an emitter network $E$ and a transition network $T$, while the guide relies on a combiner network $C$.

**Emitter architecture** The emitter network $E$ parameterises the emission probabilities as stated in equation 3. $E$ is a feed-forward neural network containing two initial layers that branch into three. One output branch is a single layer that outputs the degree of freedom of the Student T distribution, which is shared between the two angles. The other two branches output a mean $\mu$ and a standard deviation $\sigma$ for $\phi$ and $\psi$, respectively. Each hidden layer of the neural network contained 200 neurons with rectified linear unit (ReLU) activation. Output layers for $\mu$ values had no activation, as the periodic distribution automatically transforms values to a range between $-\pi$ and $\pi$. Output layers for $\sigma$ and degrees of freedom $\nu$ used softplus activation to ensure positive, real numbered values. The architecture of $E$ is depicted in figure 4.

**Transition and combiner architecture** The transition network $T$ and the combiner network $C$ specify the transition densities from the previous to the current latent state of the model (equation 2) and the guide (equation 4), respectively. In the original DMM (Krishnan et al., 2017), $C$ was inspired by the Gated Recurrent Unit (GRU) architecture (Cho et al., 2014), while $T$ was a simple feed forward network. Here, both $C$ and $T$ were based on GRU cells to allow for better horizontal information flow (figure 5).

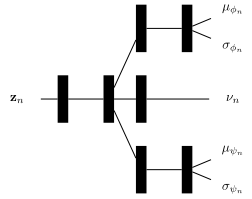



*Figure 4.* Architecture of the emitter neural network, $E$. Black rectangles represent ReLU-activated fully connected layers.

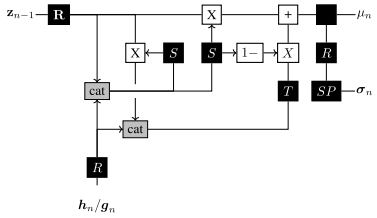



*Figure 5.* Architecture of the transition $T$ and combiner $C$ neural networks. Black squares represent single neural network layers activated by a ReLU (R), sigmoid (S), tanh (T), softplus (SP) or no activation. White squares represent element-wise mathematical operations. Gray squares represent tensor concatenation. Note that the network takes as input either $\boldsymbol{h}_n$ or $\boldsymbol{g}_n$ obtained from the RNN in the model or the guide, respectively.

The total number of parameters in BIFROST are shown in table 1.

### 3.7. Hyperparameter optimization

A simple hyperparameter search was performed with the test ELBO as the selection criterion (data not shown). The final model was trained with a learning rate of $0.0003$ with a scheduler reducing the learning rate by $90\%$ when no improvement was seen for $10$ epochs. Minibatch size was $200$. The Adam optimiser was used with a $\beta_1$ and $\beta_2$ of $0.96$ and $0.999$ respectively. The latent space dimensionality was $40$. All hidden activations (if not specified above) were ReLU activations. We employed norm scaling of the gradient to a norm of $10.0$. Finally, early stopping was employed with a patience of $50$ epochs.

| Neural networks | | | | | $Z_0$ | |
|---|---|---|---|---|---|---|
| E | T | C | H | G | p | q |
| 24 805 | 142 280 | 142 280 | 89 200 | 90 000 | 40 | 40 |
| **Total parameters:** | | | **488 645** | | | |

*Table 1.* Number of parameters in BIFROST. E: Emitter, T: Transition, C: Combiner, H: model RNN, G: guide RNN, p: model, q: guide

### 3.8. Sampling from the model

The BIFROST model (figure 2) is designed with explicit conditioning on amino acid sequences allowing a simple and efficient ancestral sampling approach that eliminates the need for using the guide for predictions. Thus, the guide is used solely for the purpose of model estimation and is discarded upon sampling.

### 3.9. Fragment library generation and benchmarking

Fragment libraries are a collection of fragments, consisting of typically 3 or 9 amino acids with known backbone angles. Here, we focus on fragments of nine amino acids. For each fragment in a protein, 200 possible backbone conformations are sampled from BIFROST resulting in a set of $(L-8)\times 200$ fragment candidates, where $L$ is the number of amino acids in the protein. These candidates are compared to the observed fragment by calculating the angular root mean square deviation (RMSD) between the corresponding angles as proposed in Boomsma et al. (2008). The choice of 9-mer fragments and the 200 samples per fragment were made to emulate the default behavior of the Rosetta fragment picker (see below), for fair comparison.

The aggregated quality of fragment libraries are generally represented by two metrics; *precision* and *coverage*. Precision is defined as the fraction of candidates with an RMSD to the observed below a certain threshold, whereas coverage is the fraction of positions covered by at least one candidate with an RMSD below a certain threshold. Evaluating the precision and coverage at increasing thresholds yields two curves, and the quality of the fragment library is quantified by the area under these two curves.

BIFROST was benchmarked against Rosetta's fragment picker (Gront et al., 2011) using the precision and coverage metrics. The fragment picker was run using default parameters, picking 200 fragments per position. Secondary structure predictions were performed using SAM-T08 (Karplus, 2009), PSIPRED (Jones, 1999) and Jufo (Leman et al., 2013). Sequences that were homologous to the targets were excluded (*–nohoms* flag).

Fragment libraries were generated for all available regular (denoted "T") targets from the latest installment of the bi-annual protein structure prediction competition Critical Assessment of Techniques for Protein Structure Prediction (CASP13).

### 3.10. Runtime comparison

In order to compare the runtime of BIFROST to that of the fragment picker, nine proteins of varying lengths were selected. Both tools generated 200 samples per fragment. The experiment was run on the same 32-core machine for both the fragment picker and BIFROST.

## 4. Results

To show that the model is able to capture general protein backbone behavior, angles were generated conditioned on the sequences of 5000 previously unseen fragments and compared to the observed angles. The model was able to recreate the observed Ramachandran plots with minimal added noise (figure 6).

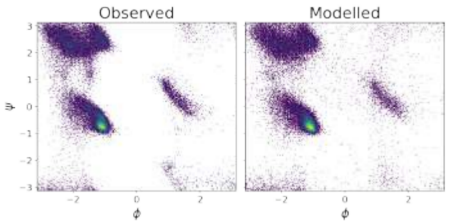



*Figure 6.* Observed and modelled aggregated Ramachandran plots

While most amino acids show angle distributions similar to the background in figure 6, glycine and proline are exceptions due to the nature of their side chains. The side chain of glycine is a single hydrogen atom, allowing the backbone to be exceptionally flexible, while the side chain of proline is covalently linked to the backbone restraining the conformational space. The modelled distribution of angles for these two unique cases, along with leucine to represent the general case, show that the model is able to capture specific amino acid properties (figure 7).

The left side of figure 8 shows a thin, smoothed coil representation of 100 samples from BIFROST conditioned on example 9-mer fragments that were observed to be either $\alpha$-helix, $\beta$-strand, or coiled. The right side shows distributions of backbone RMSDs of 5000 sampled fragments to the observed structure from BIFROST and as picked by Rosetta's fragment picker.

The RMSDs were generally distributed towards 0Å for the $\alpha$-helix case, showcasing BIFROSTs ability to predict this

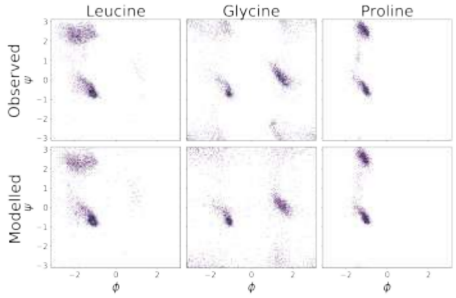



*Figure 7.* Amino acid specific Ramachandran plots

well defined secondary structure element. The model has more difficulty modelling $\beta$-strands and coils. However, the distributions of the RMSDs are nearly identical to those produced by the fragment picker. For coil fragments, the RMSDs were distributed around 3Å reflecting the inherent variability of those fragments.

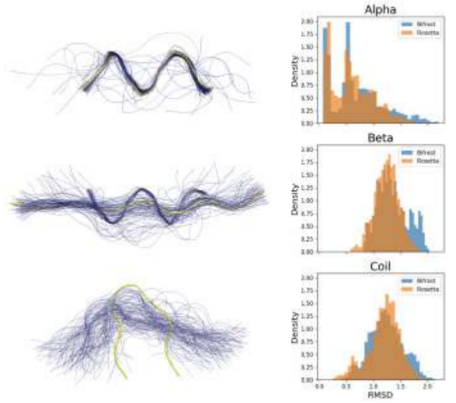



*Figure 8.* Left: 100 samples of backbone dihedral angles (blue) superimposed on the observed structures (yellow). For clarity, the backbones are represented as thin, smoothed coils instead of traditional cartoon representations. Right: Aggregated RMSDs of BIFROST-sampled conformations and conformations picked by Rosetta's fragment picker for sequences observed as $\alpha$-helix, $\beta$-strand, and coil respectively.

BIFROST was benchmarked against Rosetta's fragment picker (Gront et al., 2011) on all publicly available CASP13 regular targets. BIFROST generated fragment libraries with comparable precision and coverage to the fragment picker (figure 9).

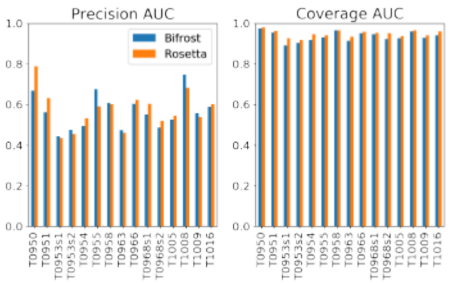



*Figure 9.* Comparison of fragment libraries generated by BIFROST, relying on just the amino acid sequences, against Rosetta's fragment picker, which uses external information and relies on ensemble predictions of secondary structure.

Finally, BIFROST enables efficient sampling of fragment libraries. The runtime of BIFROST and the fragment picker are compared in figure 10 on a set of nine proteins of varying lengths. Both runtimes roughly scale linearly with protein length, but BIFROST has a smaller constant term than the fragment picker.

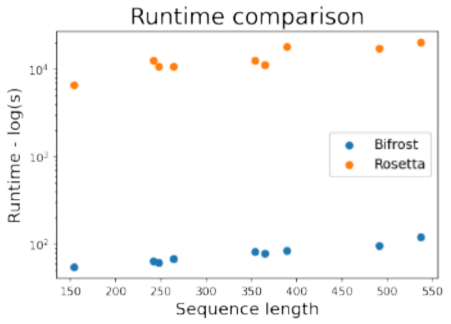



*Figure 10.* Runtime comparison between Rosetta's fragment picker and BIFROST on a set of nine proteins of varying lengths.

## 5. Discussion

BIFROST is a deep, generative model of local protein structure conditioned on sequence that provides a probabilistic approach to generating fragment libraries.

The quality of the generated fragment libraries is on par with Rosetta's fragment picker, despite using much less information, such as an ensemble of secondary structure predictors. Due to the probabilistic nature of BIFROST, distributions tend to be slightly wider than those resulting

from picking structural fragments from the PDB based on sequence similarity. This wider distribution plausibly reflects the dynamic nature of protein structure, which is not captured in the experimental data provided by static X-ray structures.

The model was estimated using SVI, relying on the ELBO variational objective. As the ELBO provides a lower bound on the log evidence (Kingma & Welling, 2014), we can evaluate the probability of a specific local structure given the sequence, simply by evaluating the ELBO. Evaluating the probability of fragments is crucial for correct sampling of the conformational space, for example in the case of equilibrium simulations of protein dynamics (Boomsma et al., 2014). The probabilities assigned by BIFROST can be used to decide how often a fragment should be sampled in the folding process. In contrast, existing methods do not provide an explicit measure of fragment confidence.

In this paper the focus was kept on fragments of nine residues for ease of comparison to the fragment picker. However, the DMM architecture of BIFROST allows generation of fragments of arbitrary length but with an observed drop-off in performance as the length of fragments are increased (data not shown).

Existing methods rely heavily on the availability of multiple sequence alignments (MSA) and other information, such as secondary structure predictors. As MSAs are not available for orphan proteins or synthetic proteins, the need for pure sequence based models is evident.

## 6. Acknowledgements

We acknowledge funding from the Innovation Fund Denmark under the grant "Accelerating vaccine development through a deep learning and probabilistic programming approach to protein structure prediction". We thank Wouter Boomsma for help with the wrapped student T distribution

## References

Alford, R. F., Leaver-Fay, A., Jeliazkov, J. R., O'Meara, M. J., DiMaio, F. P., Park, H., Shapovalov, M. V., Renfrew, P. D., Mulligan, V. K., Kappel, K., Labonte, J. W., Pacella, M. S., Bonneau, R., Bradley, P., Dunbrack, R. L., Das, R., Baker, D., Kuhlman, B., Kortemme, T., and Gray, J. J. The Rosetta all-atom energy function for macromolecular modeling and design. *Journal of Chemical Theory and Computation*, 13(6):3031–3048, jun 2017. ISSN 15499626. doi: 10.1021/acs.jctc.7b00125.

Bengio, Y. and Frasconi, P. An input output HMM architecture. *Neural Information Processing Systems*, pp. 427–434, 1995. ISSN 15322092. doi: 10.1093/europace/euq350.

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., and Bourne, P. E. The Protein Data Bank, jan 2000. ISSN 03051048.

Best, R. B. Computational and theoretical advances in studies of intrinsically disordered proteins. *Current Opinion in Structural Biology*, 42:147–154, feb 2017. ISSN 0959440X. doi: 10.1016/j.sbi.2017.01.006.

Bhattacharya, D., Adhikari, B., Li, J., and Cheng, J. FRAGSION: Ultra-fast protein fragment library generation by IOHMM sampling. *Bioinformatics*, 32(13):2059–2061, 2016. ISSN 14602059. doi: 10.1093/bioinformatics/btw067.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P. A., Horsfall, P., and Goodman, N. D. Pyro: Deep Universal Probabilistic Programming. *J. Mach. Learn. Res.*, 20: 28:1—-28:6, 2019.

Boomsma, W., Mardia, K. V., Taylor, C. C., Ferkinghoff-Borg, J., Krogh, A., and Hamelryck, T. A generative, probabilistic model of local protein structure. *Proceedings of the National Academy of Sciences of the United States of America*, 105(26):8932–8937, 2008. ISSN 00278424. doi: 10.1073/pnas.0801715105.

Boomsma, W., Frellsen, J., and Hamelryck, T. *Probabilistic models of local biomolecular structure and their applications*. Springer, 2012. doi: 10.1007/978-3-642-27225-7_10.

Boomsma, W., Tian, P., Frellsen, J., Ferkinghoff-Borg, J., Hamelryck, T., Lindorff-Larsen, K., and Vendruscolo, M. Equilibrium simulations of proteins using molecular fragment replacement and NMR chemical shifts. *Proceedings of the National Academy of Sciences of the United States of America*, 111(38):13852–13857, 2014. ISSN 10916490. doi: 10.1073/pnas.1404948111.

Bystroff, C., Thorsson, V., and Baker, D. HMMSTR: A hidden Markov model for local sequence-structure correlations in proteins. *Journal of Molecular Biology*, 301(1):173–190, 2000. ISSN 00222836. doi: 10.1006/jmbi.2000.3837.

Camproux, A. C., Tuffery, P., Chevrolat, J. P., Boisvieux, J. F., and Hazout, S. Hidden Markov model approach for identifying the modular framework of the protein backbone. *Protein Engineering*, 12(12):1063–1073, 1999. ISSN 02692139. doi: 10.1093/protein/12.12.1063.

Chikenji, G., Fujitsuka, Y., and Takada, S. Shaping up the protein folding funnel by local interaction: Lesson from a structure prediction study. *Proceedings of the National Academy of Sciences*, 103(9):3141–3146, feb 2006. ISSN 0027-8424. doi: 10.1073/pnas.0508195103.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734. Association for Computational Linguistics (ACL), jun 2014. ISBN 9781937284961. doi: 10.3115/v1/d14-1179.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., and Bengio, Y. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, volume 2015-Janua, pp. 2980–2988, 2015.

De Oliveira, S. H., Shi, J., and Deane, C. M. Building a better fragment library for de novo protein structure prediction. *PLoS ONE*, 10(4), 2015. ISSN 19326203. doi: 10.1371/journal.pone.0123998.

Edgoose, T., Allison, L., and Dowe, D. L. An MML classification of protein structure that knows about angles and sequence. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pp. 585–596, 1998. ISSN 2335-6928.

Eswar, N., Webb, B., Marti-Renom, M. A., Madhusudhan, M., Eramian, D., Shen, M.-y., Pieper, U., and Sali, A. Comparative protein structure modeling using modeller. *Current Protocols in Bioinformatics*, 15(1):5.6.1–5.6.30, sep 2006. ISSN 1934-340X. doi: 10.1002/0471250953.bi0506s15.

Fraccaro, M., Sønderby, S. K., Paquet, U., and Winther, O. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems*, pp. 2207–2215, 2016.

Gront, D., Kulp, D. W., Vernon, R. M., Strauss, C. E., and Baker, D. Generalized fragment picking in rosetta: Design, protocols and applications. *PLoS ONE*, 6(8): 23294, 2011. ISSN 19326203. doi: 10.1371/journal.pone.0023294.

Hamelryck, T., Kent, J. T., and Krogh, A. Sampling realistic protein conformations using local structural bias. *PLoS Computational Biology*, 2(9):e131, sep 2006. ISSN 1553-7358. doi: 10.1371/journal.pcbi.0020131.

Hamelryck, T., Mardia, K., and Ferkinghoff-Borg, J. (eds.). *Bayesian Methods in Structural Bioinformatics*. Statistics for Biology and Health. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27224-0. doi: 10.1007/978-3-642-27225-7.

Jones, D. T. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of Molecular Biology*, 292(2):195–202, 1999. ISSN 00222836. doi: 10.1006/jmbi.1999.3091.

Jones, T. A. and Thirup, S. Using known substructures in protein model building and crystallography. *The EMBO journal*, 5(4):819–22, apr 1986. ISSN 0261-4189.

Kalev, I. and Habeck, M. HHfrag: HMM-based fragment detection using HHpred. *Bioinformatics*, 27(22): 3110–3116, 2011. ISSN 13674803. doi: 10.1093/bioinformatics/btr541.

Karplus, K. SAM-T08, HMM-based protein structure prediction. *Nucleic Acids Research*, 37(SUPPL. 2):492–497, 2009. ISSN 03051048. doi: 10.1093/nar/gkp403.

Khurana, S., Laurent, A., Hsu, W.-N., Chorowski, J., Lancucki, A., Marxer, R., and Glass, J. A Convolutional Deep Markov Model for Unsupervised Speech Representation Learning. In *Proceedings of Interspeech*, jun 2020.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

Kingma, D. P. and Welling, M. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019. ISSN 1935-8237. doi: 10.1561/2200000056.

Krishnan, R. G., Shalit, U., and Sontag, D. Structured inference networks for nonlinear state space models. In *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pp. 2101–2109, sep 2017.

Leaver-Fay, A., O'Meara, M. J., Tyka, M., Jacak, R., Song, Y., Kellogg, E. H., Thompson, J., Davis, I. W., Pache, R. A., Lyskov, S., Gray, J. J., Kortemme, T., Richardson, J. S., Havranek, J. J., Snoeyink, J., Baker, D., and Kuhlman, B. Scientific benchmarks for guiding macromolecular energy function improvement. In *Methods in Enzymology*, volume 523, pp. 109–143. Academic Press Inc., 2013. ISBN 9780123942920. doi: 10.1016/B978-0-12-394292-0.00006-0.

Leman, J. K., Mueller, R., Karakas, M., Woetzel, N., and Meiler, J. Simultaneous prediction of protein secondary structure and transmembrane spans. *Proteins: Structure, Function and Bioinformatics*, 81(7):1127–1140, jul 2013. ISSN 08873585. doi: 10.1002/prot.24258.

Lennox, K. P., Dahl, D. B., Vannucci, M., Day, R., and Tsai, J. W. A Dirichlet process mixture of hidden Markov models for protein structure prediction. *Annals of Applied Statistics*, 4(2):916–942, 2010. ISSN 19326157. doi: 10.1214/09-AOAS296.

Levinthal, C. How to fold graciously. *Mössbauer Spectroscopy in Biological Systems Proceedings*, 24(41):22–24, 1969. ISSN 1041-1135. doi: citeulike-article-id:380320.

Li, S. C., Bu, D., Xu, J., and Li, M. Fragment-HMM: A new approach to protein structure prediction. *Protein Science*, 17(11):1925–1934, 2008. ISSN 09618368. doi: 10.1110/ps.036442.108.

Liebschner, D., Afonine, P. V., Baker, M. L., Bunkóczi, G., Chen, V. B., Croll, T. I., Hintze, B., Hung, L.-w., Jain, S., McCoy, A. J., Moriarty, N. W., Oeffner, R. D., Poon, B. K., Prisant, M. G., Read, R. J., Richardson, J. S., Richardson, D. C., Sammito, M. D., Sobolev, O. V., Stockwell, D. H., Terwilliger, T. C., Urzhumtsev, A. G., Videau, L. L., Williams, C. J., and Adams, P. D. Macromolecular structure determination using X-rays, neutrons and electrons: recent developments in Phenix. *Acta Crystallographica Section D Structural Biology*, 75(10):861–877, oct 2019. ISSN 2059-7983. doi: 10.1107/S2059798319011471.

Mardia, K. V. and Jupp, P. E. *Directional Statistics*. Wiley Series in Probability and Statistics. John Wiley and Sons, Inc., Hoboken, NJ, USA, jan 2008. ISBN 9780470316979. doi: 10.1002/9780470316979.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library, 2019. ISSN 23318422.

Pewsey, A., Lewis, T., and Jones, M. C. The wrapped t family of circular distributions. *Australian and New Zealand Journal of Statistics*, 49(1):79–91, 2007. ISSN 1467842X. doi: 10.1111/j.1467-842X.2006.00465.x.

Ramachandran, G. N., Ramakrishnan, C., and Sasisekharan, V. Stereochemistry of polypeptide chain configurations, 1963. ISSN 00222836.

Rohl, C. A., Strauss, C. E., Misura, K. M., and Baker, D. Protein structure prediction using rosetta. *Methods in Enzymology*, 383:66–93, 2004. ISSN 00766879. doi: 10.1016/S0076-6879(04)83004-0.

Roy, A., Kucukural, A., and Zhang, Y. I-TASSER: A unified platform for automated protein structure and function prediction. *Nature Protocols*, 5(4):725–738, 2010. ISSN 17502799. doi: 10.1038/nprot.2010.5.

Šali, A. and Blundell, T. L. Comparative protein modelling by satisfaction of spatial restraints. *Journal of Molecular Biology*, 234(3):779–815, dec 1993. ISSN 00222836. doi: 10.1006/jmbi.1993.1626.

Santos, K. B., Trevizani, R., Custodio, F. L., and Dardenne, L. E. Profrager Web Server: Fragment libraries generation for protein structure prediction. In *Proceedings of the International Conference on Bioinformatics and Computational Biology (BIOCOMP)*, pp. 38, 2015.

Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D. T., Silver, D., Kavukcuoglu, K., and Hassabis, D. Protein structure prediction using multiple deep neural networks in the 13th Critical Assessment of Protein Structure Prediction (CASP13). *Proteins: Structure, Function and Bioinformatics*, 87(12):1141–1148, 2019. ISSN 10970134. doi: 10.1002/prot.25834.

Simons, K. T., Kooperberg, C., Huang, E., and Baker, D. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions. *Journal of Molecular Biology*, 268(1):209–225, 1997. ISSN 00222836. doi: 10.1006/jmbi.1997.0959.

Song, Y., Dimaio, F., Wang, R. Y. R., Kim, D., Miles, C., Brunette, T., Thompson, J., and Baker, D. High-resolution comparative modeling with RosettaCM. *Structure*, 21 (10):1735–1742, 2013. ISSN 09692126. doi: 10.1016/j.str.2013.08.005.

Toyer, S., Cherian, A., Han, T., and Gould, S. Human pose forecasting via deep Markov models. In *DICTA 2017 - 2017 International Conference on Digital Image Computing: Techniques and Applications*, volume 2017-Decem, pp. 1–8. Institute of Electrical and Electronics Engineers Inc., jul 2017. ISBN 9781538628393. doi: 10.1109/DICTA.2017.8227441.

Trevizani, R., Dio, F. L. C., Santos, K. B. D., and Dardenne, L. E. Critical features of fragment libraries for protein structure prediction. *PLoS ONE*, 12(1), 2017. ISSN 19326203. doi: 10.1371/journal.pone.0170131.

Wang, G. and Dunbrack, R. L. PISCES: Recent improvements to a PDB sequence culling server. *Nucleic Acids Research*, 33(SUPPL. 2), 2005. ISSN 03051048. doi: 10.1093/nar/gki402.

Wang, T., Qiao, Y., Ding, W., Mao, W., Zhou, Y., and Gong, H. Improved fragment sampling for ab initio protein structure prediction using deep neural networks. *Nature Machine Intelligence*, 1(8):347–355, aug 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0075-7.

Zhao, F., Li, S., Sterner, B. W., and Xu, J. Discriminative learning for protein conformation sampling. *Proteins: Structure, Function and Genetics*, 73(1):228–240, oct 2008. ISSN 08873585. doi: 10.1002/prot.22057.

Zhao, F., Peng, J., Debartolo, J., Freed, K. F., Sosnick, T. R., and Xu, J. A probabilistic and continuous model of protein conformational space for template-free modeling. *Journal of Computational Biology*, 17(6):783–798, 2010. ISSN 10665277. doi: 10.1089/cmb.2009.0235.

Zhi-Xuan, T., Soh, H., and Ong, D. Factorized inference in deep Markov models for incomplete multimodal time series. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):10334–10341, apr 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i06.6597.

# 16 Paper 4: Ancestral protein sequence reconstruction using a tree-structured Ornstein-Uhlenbeck variational autoencoder

**Authors:** Lys Sanz Moreta, Ola Rønning, Ahmad Salim Al-Sibahi, Jotun Hein, Douglas Theobald, and Thomas Hamelryck.

**Motivation:** This work presents a new model (Draupnir) for Ancestral protein reconstruction along a given phylogenetic tree. Draupnir explicitly represents the evolutionary process by embedding a tree-structured Ornstein–Uhlenbeck process [163] in the latent space of a Variational Autoencoder [119]. Current models of evolution are factorized, treating protein sites as independent. This assumption impedes modelling the coevolution of protein residues necessary for capturing amino acids interactions in the protein 3D structure. This work presents the first implementation of a non-factorized model that allows for simultaneously modelling of substitutions, insertions and deletions [39, 36] and allows to account for co-evolving sites in the ancestor's predictions.

Under review as a conference paper at ICLR 2022

# Ancestral protein sequence reconstruction using a tree-structured Ornstein-Uhlenbeck variational autoencoder

**Anonymous authors**
Paper under double-blind review

## Abstract

We introduce a deep generative model for representation learning of biological sequences that, unlike existing models, explicitly represents the evolutionary process. The model makes use of a tree-structured Ornstein-Uhlenbeck process, obtained from a given phylogenetic tree, as an informative prior for a variational autoencoder. We show the model performs well on the task of ancestral sequence reconstruction of single protein families. Our results and ablation studies indicate that the explicit representation of evolution using a suitable tree-structured prior has the potential to improve representation learning of biological sequences considerably. Finally, we briefly discuss extensions of the model to genomic-scale data sets and the case of a latent phylogenetic tree.

## 1 Introduction

Representation learning of biological sequences is important for data exploration and downstream tasks such as protein design (Detlefsen et al., 2020; Alley et al., 2019). Deep generative models such as variational autoencoders (VAEs) (Kingma & Welling, 2013; 2019) have been especially useful for this purpose (Riesselman et al., 2018; Greener et al., 2018). However, current models do not take evolutionary information fully into account, i.e., by relating the sequences belonging to a protein family in a phylogenetic tree and incorporating parameterized evolutionary models (Durbin et al., 1998). To address this problem, we replace the standard multivariate Gaussian prior of a conventional VAE with a tree-structured prior that takes into account a given evolutionary tree. We propose a prior based on the Ornstein-Uhlenbeck Gaussian process on a tree (Hansen, 1997; Jones & Moriarty, 2013). We apply the model to a classic problem in phylogenetics, namely the inference of ancestral sequences.

Ancestral sequence reconstruction (ASR), i.e., the inference of ancestral sequences given their descendants or leaf sequences (Pauling et al., 1963; Yang et al., 1995; Koshi & Goldstein, 1996; Joy et al., 2016; Hochberg & Thornton, 2017; Selberg et al., 2021), has important applications including protein engineering (Cole & Gaucher, 2011; Spence et al., 2021), modeling tumour evolution (El-Kebir et al., 2015), evaluating virus diversity and vaccine design (Gaschen et al., 2002), understanding drug mechanisms (Wilson et al., 2015) and reconstructing ancient proteins *in vitro* (Chang et al., 2002; Wilson et al., 2015; Hochberg & Thornton, 2017).

As input, we assume a set of $n_S$ known, aligned leaf sequences and their phylogenetic tree. The task we want to address is the inference of the $n_A \leq n_S - 1$ unknown, ancestral sequences (Joy et al., 2016). We show that our probabilistic model, called Draupnir, is about on par with or better than the accuracy of established ASR methods for a standard experimentally-derived data set (Alieva et al., 2008; Randall et al., 2016) and several simulated data sets. In addition, we show that Draupnir is capable of capturing coevolution among sequence positions, unlike conventional ASR methods.

The paper is organised as follows. In Background, we briefly discuss evolution of biological sequences, ancestral sequence reconstruction and the tree-structured Ornstein-Uhlenbeck

process. In Related Work, we discuss deep generative models of biological sequences. In Methods, we describe the Draupnir model, the inference of ancestral sequences and the setup of the benchmarking experiments. In Results, we discuss the quality of the latent representations, compare the accuracy of Draupnir with state-of-the-art phylogenetic methods for ASR, and present the results of ablation experiments. We end with a brief discussion of future work, including extending the method to genomic-scale data sets and the case of a latent phylogenetic tree.

## 2 Background

### 2.1 Protein sequences and evolution

Biological molecules such as proteins and nucleic acids (DNA, RNA) can be characterised by sequences of characters from an alphabet of size $n_C$, where typically $n_C = 21$ for proteins and $n_C = 5$ for nucleic acids (Durbin et al., 1998). These alphabets include one character that represents a gap, which is useful in aligning related sequences in a multiple sequence alignment (MSA). In the course of evolution, mutations arise that cause changes in these sequences, including character substitutions, deletions and insertions. A set of $n_S$ known, homologous, extant sequences and their evolutionary relationships are naturally represented as $n_S$ leaf nodes in a binary tree or phylogeny, where the $n_A$ internal or ancestral nodes represent unknown, ancestral sequences (Joy et al., 2016). Among the internal nodes, the root node is the most ancient node.

Edges between two nodes in the tree are labelled by positive real numbers that represent the time difference or the amount of change between them. Such a labelled binary tree naturally defines a $(n_S + n_A) \times (n_S + n_A)$ matrix containing the pairwise distances between all nodes, called the *patristic distance matrix*, $\mathbf{T}$. In the context of biological sequences, the field of phylogenetics is concerned with the inference of the tree topology, the labels of the tree's edges and the composition of the ancestral sequences, making use of methods based on heuristics (such as maximum parsimony) or probabilistic, evolutionary models (Joy et al., 2016).

### 2.2 Ancestral protein reconstruction

The ASR problem amounts to inferring the composition of the $n_A$ ancestral sequences from the $n_S$ extant sequences, making use of a tractable model of evolution (Joy et al., 2016). Typically, the phylogenetic tree that relates the sequences is assumed known. Standard methods to do this typically assume independent (factorized) evolution of the characters in the sequence, which is a computationally convenient but unrealistic assumption. For example, in proteins, amino acids are involved in an intricate 3-dimensional network of interactions that can lead to strong dependencies between amino acids far part in the sequence. This phenomenon is called *epistasis* (Hochberg & Thornton, 2017), which requires coevolutionary models that go beyond the factorized assumption. Nonetheless, it has been possible to infer ancestral sequences and subsequently resurrect functional ancient, ancestral proteins *in vitro* (Hochberg & Thornton, 2017). The aim of this work is to go beyond the assumption of independent, factorized evolution by using a model of evolution that features continuous, latent vector representations of the protein sequences. This allows us to formulate the ASR problem in the context of a deep generative model.

### 2.3 The Ornstein-Uhlenbeck process on a phylogenetic tree

Typically, ASR of biological sequences is done using factorised evolutionary models that represent substitutions, insertions and deletions of the discrete characters in the sequences (Joy et al., 2016). In contrast, Draupnir aims to model the evolution of latent, continuous representations or underlying traits of the sequences. A simple diffusive process allowing for an equilibrium distribution is the Ornstein-Uhlenbeck (OU) process (Hansen, 1997; Jones & Moriarty, 2013). As the OU process is a Gaussian process, it has a Gaussian equilibrium distribution, as well as Gaussian marginal distributions.

We use an OU process on a phylogenetic tree (TOU process) (Hansen, 1997; Jones & Moriarty, 2013) to put the latent representations under the control of a parameterized evolutionary model. Apart from the mean, which for our purposes can be assumed to be zero, the TOU process has three parameters: the variation unattributable to the phylogeny or the intensity of specific variation $\sigma_n$, the characteristic length scale of the evolutionary dynamics $\lambda$, and the intensity of inherited variation $\sigma_f$. The covariance function for the corresponding multivariate Gaussian distribution is then given by (Hadjipantelis et al., 2012; Jones & Moriarty, 2013),

$$\Sigma_{k,l} = \sigma_f^2 \exp\left(\frac{-T_{k,l}}{\lambda}\right) + \sigma_n^2 \delta_{k,l} \tag{1}$$

where $T_{k,l}$ is the patristic distance between nodes $k$ and $l$ in the tree, and the Kronecker delta $\delta_{k,l} = 1$ if $k = l$, and 0 otherwise.

The TOU process and related diffusive processes on trees are well-established evolutionary models that have been used to model the evolution of continuous traits, such as body mass or length (Joy et al., 2016). For example, Lartillot (2014) proposes a phylogenetic Kalman filter for ancestral trait reconstruction of low-dimensional, continuous traits; Tolkoff et al. (2018) propose phylogenetic factor analysis, in which a latent variable under the control of a small number independent univariate Brownian diffusion processes is related to observed traits through a loading matrix; Horta et al. (2021) use a multivariate TOU process and Markov chain Monte Carlo to model both continuous traits and sequences of discrete characters. To represent the latter, they make use of a pairwise Potts model.

## 3 Related work

### 3.1 Representation learning of biological sequences

A VAE (Kingma & Welling, 2013; 2019) is a probabilistic, generative model featuring latent vectors or representations, $\{\mathbf{z}\}_{n=1}^N$, that are independently sampled from a prior distribution, $\mathbf{z}_n \sim \pi(\mathbf{z}_n)$. The latent vectors are passed to a neural network (the *decoder*) with parameters $\theta$, leading to a likelihood, $\mathbf{x}_n \sim p_\theta(\mathbf{x}_n \mid \mathrm{NN}_\theta(\mathbf{z}_n))$, for the data, $\{\mathbf{x}\}_{n=1}^N$. The prior is typically a standard multivariate Gaussian distribution, but other priors have been used, such as distributions on the Poincaré ball to recover hierarchical structures (Mathieu et al., 2019). The posterior distribution $p(\mathbf{z}_n \mid \mathbf{x}_n)$ is intractable, but can be approximated with a variational distribution or *guide*, $q_\phi(\mathbf{z}_n \mid \mathrm{NN}_\phi(\mathbf{x}_n))$, involving a second neural network (the *encoder*). Point estimates of the parameters $\theta$ and $\phi$ are obtained by maximizing the *evidence lower bound* (ELBO),

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_q\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})}\right)\right],$$

using stochastic gradient ascent (Hoffman et al., 2013).

VAEs are increasingly used for representation learning of biological sequences (Detlefsen et al., 2020). Riesselman et al. (2018) use a VAE with biologically motivated priors to evaluate the stability of mutants and to explore new regions of sequence space. Greener et al. (2018) use autoencoders to design metal-binding proteins and novel protein folds. Ding et al. (2019) show that the latent representations obtained with a VAE can capture evolutionary relationships between sequences. The above models do not represent the phylogenetic tree explicitly, but typically aim to condition on some evolutionary information by training on pre-computed MSAs - an approach that has been called *evo-tuning* (Rao et al., 2019; Detlefsen et al., 2020). Hawkins-Hooker et al. (2021) use a VAE with a convolutional encoder and decoder, combining upsampling and autoregression, without relying on a MSA.

The above models assume that the latent vectors factor independently, which is computationally convenient but unrealistic if the sequences are related to each other in a phylogeny. A more realistic approach thus uses a prior $\pi(\{\mathbf{z}\}_{n=1}^N \mid \tau, \kappa)\pi(\kappa)$ that conditions the latent vectors on a given phylogenetic tree, $\tau$, and an evolutionary model with latent

parameters, $\kappa$. Because the latent vectors do not factor independently anymore, mini-batch training can include the sequences but not the latent vectors, which limits the possible size of the data sets. Nonetheless, we show here that such a model is both computationally tractable and practically useful for realistic data sets concerning single protein families.

# 4 Methods

## 4.1 The Draupnir model

The pseudocode of the Draupnir model is given in Algorithm 1; Figure 1 shows the corresponding graphical model. A summary of the variables, their dimensions and the notation is given in Tables 1 and 2 in Appendix A.1.

As inputs, we assume (a) a set of $n_S$ aligned sequences, each with length $n_L$, organized in the $n_S \times n_L$ matrix, $\mathbf{S}$, and (b) information on their phylogenetic tree in the form of their $(n_S + n_A) \times (n_S + n_A)$ patristic distance matrix, $\mathbf{T}$.

The latent matrix $\mathbf{Z}$ of the model is a matrix with $n_S$ rows (one for each leaf sequence) and $n_Z$ columns, where $n_Z$ is the size of the latent representation of the sequences. In all experiments, $n_Z = 30$. Each column of $\mathbf{Z}$ (with $n_S$ elements) is sampled from a univariate OU process on the phylogenetic tree, representing the evolution of a hidden trait underlying the sequences along the tree. Each row of $\mathbf{Z}$ corresponds to the latent vector of a standard VAE. However, unlike a standard VAE, the latent vectors do not factorize independently.

For each of the $n_Z$ TOU processes, the parameters of the TOU process, corresponding to $\kappa$ in Section 3.1, are sampled from a suitable prior distribution. The TOU process has three parameters: $\sigma_f$, $\lambda$ and $\sigma_n$. As we use $n_Z$ tree OU processes (one for each column of $\mathbf{Z}$), we need to sample $n_Z$ sets of these three parameters.

For each of the three parameters, $\sigma_f, \lambda, \sigma_n$, we sample a hyperparameter $(\alpha_1, \alpha_2, \alpha_3)$ from a half-normal distribution with scale parameter equal to one. These hyperparameters serve as scale parameter for the half-normal priors over $\sigma_f, \lambda$ and $\sigma_n$. Given the parameters of the TOU process obtained from the prior described above, an $n_S \times n_S$ covariance matrix can be calculated based on the patristic distance matrix of the leaves, $\mathbf{T}^{(S,S)}$. We need one such covariance matrix for each of the $n_Z$ columns of the matrix of latent representations, $\mathbf{Z}$. The element $k, l$, with $k, l \in 1, ..., n_S$, of covariance matrix $h$, with $h \in 1, ..., n_Z$, is given by (Hadjipantelis et al., 2012; Jones & Moriarty, 2013),

$$C_{h,k,l} = \sigma_{f,h}^2 \exp(-T_{k,l}/\lambda_h) + \sigma_{n,h}^2 \delta_{k,l}. \tag{2}$$

As decoder, we use a bidirectional gated recurrent unit (GRU, Cho et al. (2014)) with length equal to the alignment length, $n_L$. The input at each position $i$ of the GRU for sequence $\mathbf{S}_{k,:}$ is a concatenated vector, consisting of the latent vector $\mathbf{Z}_{k,:}$ representing sequence $k$, and the BLOSUM embedding $\mathbf{E}_{i,:}$, which is the result of applying a fully connected neural network, $\mathrm{NN}_\theta^{(1)}$, to the BLOSUM vector $\mathbf{V}_{i,:}$ describing the amino acid preferences at position $i$ in the MSA (see Section 4.2). For each of the $n_S$ sequences and for each position $i$, the GRU states are mapped to a logit vector that specifies the probabilities of the $n_C$ characters using another fully connected neural network, $\mathrm{NN}_\theta^{(2)}$. The architecture of the networks is given in Appendix A.2.

## 4.2 BLOSUM embeddings

A BLOSUM matrix $\mathbf{B}$ is an $n_C \times n_C$ substitution matrix used for sequence alignment, where each row contains the log-odds scores of replacing a given character with any of the other characters (Henikoff & Henikoff, 1992). Each position (column) in the MSA is represented by the weighted average of the BLOSUM vectors of the characters in that column (see Algorithm 2). The averaged BLOSUM vectors only need to be precomputed once. In the model, the BLOSUM vectors are processed into BLOSUM embeddings by a neural network

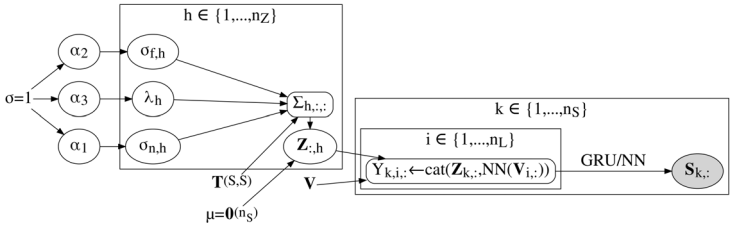



Figure 1: Draupnir as a graphical model. For notation and information on variables and their dimensions see Tables 1 and 2 in Appendix A.1 and A.2. Random variables are shown as ellipses, while deterministic quantities are shown as rounded boxes and observed random variables are shown as shaded ellipses. Parameters of priors and other given quantities are shown without boxes. The model contains three plates, respectively corresponding to the number of dimensions of the leaf sequence-specific latent vector $\mathbf{Z}_{\mathbf{k},:}$ ($n_Z$=30), the number of leaf sequences ($n_S$) and the alignment length ($n_L$). "cat" indicates the concatenation of two vectors. $\alpha$ are the hyperparameters and $\sigma_n, \sigma_f, \lambda$ are the parameters of the TOU processes. $\mathbf{\Sigma}_{h,:,:}$ is the covariance matrix that is used to sample component $h$ of the $n_S$ latent vectors from a multivariate Gaussian distribution (with mean $\mathbf{0}^{(n_S)}$). $\mathbf{T}^{(S,S)}$ is the patristic distance matrix containing the distances between the leaf sequences. $\mathbf{Y}_{k,:}$ is the input vector for the GRU that produces the likelihood parameters for leaf sequence $k$, $\mathbf{S}_{k,:}$. $\mathbf{V}$ represents the MSA as an $n_L \times n_C$ matrix of averaged BLOSUM vectors. NN denotes a fully connected neural network.

to provide position-specific information on the MSA, while the latent variables provide sequence-specific information (see Algorithm 2).

### 4.3 Model implementation and training

Draupnir was implemented in the deep probabilistic programming language Pyro (Bingham et al., 2019) and trained using stochastic variational inference with Pyro's *AutoDelta* guide by optimizing the ELBO (Kingma & Welling, 2019), resulting in maximum a posteriori (MAP) estimates for all parameters. We use Adam (Kingma & Ba, 2014) as the optimizer using the default values. From the MAP estimates, we can sample the latent representations of the ancestral nodes (see Section 4.4 and equation 5 in the Appendix). These latent representations are then subsequently decoded to their respective ancestral sequences. We also use a custom guide to calculate a variational posterior (Draupnir-variational, see Appendix A.6). Training details can be found in Appendix A.4. All programs were executed on an Intel(R) Xeon(R) Gold 6136 CPU @ 3.00GHz machine with a Quadro RTX 6000 GPU.

### 4.4 Inference of the ancestral sequences

In this section, for ease of notation, let $\mathbf{z} \equiv \mathbf{Z}^{(A,S)}_{:,h}$ denote one of the $h \in \{1, \ldots, n_z\}$ columns of the latent representation matrix for both ancestral and leaf sequences, $\mathbf{Z}^{(A,S)}$. First, note that $\mathbf{z}$ can be partitioned as $\mathbf{z} = (\mathbf{z}^{(S)}, \mathbf{z}^{(A)})$, where $\mathbf{z}^{(S)}$ and $\mathbf{z}^{(A)}$ denote the latent representations of the leaf and ancestral sequences, respectively. The prior $p(\mathbf{z})$ is a multivariate Gaussian distribution with parameters,

$$\mu = \begin{pmatrix} \mu^{(S)} \\ \mu^{(A)} \end{pmatrix} = \begin{pmatrix} \mathbf{0}^{(n_Z)} \\ \mathbf{0}^{(n_A)} \end{pmatrix}, \mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma}^{(S,S)} & \mathbf{\Sigma}^{(S,A)} \\ \mathbf{\Sigma}^{(A,S)} & \mathbf{\Sigma}^{(A,A)} \end{pmatrix}, \mathit{\Lambda} = \begin{pmatrix} \mathit{\Lambda}^{(S,S)} & \mathit{\Lambda}^{(S,A)} \\ \mathit{\Lambda}^{(A,S)} & \mathit{\Lambda}^{(A,S)} \end{pmatrix},$$

where $\mathit{\Lambda} \equiv \mathbf{\Sigma}^{-1}$. The covariance matrices $\mathbf{\Sigma}^{(S,S)}, \mathbf{\Sigma}^{(A,A)}, \mathbf{\Sigma}^{(A,S)} \equiv \left(\mathbf{\Sigma}^{(S,A)}\right)^T$ are respectively obtained from the distance matrices concerning distances within the leaves,

**Algorithm 1** The Draupnir model

**Require:** Multiple sequence alignment $\mathbf{S}$, patristic distance matrix $\mathbf{T}$

**for** $j \in [1, 2, 3]$ **do** ▷ Hyperpriors over the TOU process parameters
  $\alpha_j \sim \mathcal{HN}(1)$

**for** $h \in [1, ..., n_Z]$ **do** ▷ Priors over the parameters of the $n_Z$ TOU processes
  $\sigma_{f,h} \sim \mathcal{HN}(\alpha_0)$
  $\sigma_{n,h} \sim \mathcal{HN}(\alpha_1)$
  $\lambda_h \sim \mathcal{HN}(\alpha_2)$

**for** $h \in [1, ..., n_Z]$ **do** ▷ Kernels for the $n_Z$ TOU processes
  **for** $k, l \in \{1, ..., n_S\}$ **do**
    $C_{h,k,l} \leftarrow \sigma_{f,h}^2 \exp(-T_{k,l}^{(S,S)}/\lambda_h) + \sigma_{n,h}^2 \delta_{k,l}$

**for** $h \in [1, \ldots, n_Z]$ **do** ▷ Prior over tree-strucured latent matrix $\mathbf{Z}$
  $\mathbf{Z}_{:,h} \sim \mathcal{MVN}(\mathbf{0}^{(n_S)}, \mathbf{C}_{h,:,:})$

**for** $i \in [1, \ldots, n_L]$ **do** ▷ BLOSUM embeddings
  $\mathbf{E}_{i,:} \leftarrow \mathrm{NN}_\theta^{(1)}(\mathbf{V}_{i,:})$

**for** $k \in [1, \ldots, n_S]$ **do** ▷ Input vector $\mathbf{Y}$ for GRU
  **for** $i \in [1, \ldots, n_L]$ **do**
    $\mathbf{Y}_{k,i,:} \leftarrow \mathrm{cat}(\mathbf{Z}_{k,:}, \mathbf{E}_{i,:})$ ▷ Concatenate sequence- and position-specific vectors

**for** $k \in [1, \ldots, n_S]$ **do** ▷ Likelihood parameters (logits) $\mathbf{L}$ from GRU
  $\mathbf{H}_{k,:,:} \leftarrow \mathrm{GRU}_\theta(\mathbf{Y}_{k,:,:})$ ▷ Bidirectional GRU states
  **for** $i \in [1, \ldots, n_L]$ **do**
    $\mathbf{L}_{k,i,:} \leftarrow \mathrm{NN}_\theta^{(2)}(\mathbf{H}_{k,i,:})$
    $S_{k,i} \sim \mathrm{Categorical}(\mathbf{L}_{k,i,:})$ ▷ Likelihood at position $i$ in sequence $k$

**Algorithm 2** Pre-computation of weighted averaged BLOSUM vectors

**Require:** Multiple sequence alignment $\mathbf{S}$

$\mathbf{V} \leftarrow \mathbf{0}^{(n_L \times n_c)}$ ▷ Initialize BLOSUM weighted average $\mathbf{V}$
**for** $i \in [1, \ldots, n_L]$ **do** ▷ Position in sequence alignment
  **for** $k \in [1, \ldots, n_S]$ **do** ▷ Index of leaf sequence
    $r \leftarrow S_{k,i}$ ▷ Character at position $i$ in leaf sequence $k$
    $\mathbf{V}_{i,:} \leftarrow \mathbf{V}_{i,:} + \mathbf{B}_{r,:}$ ▷ Add BLOSUM vector corresponding to the amino acid
  $\mathbf{V}_{\mathbf{i},:} \leftarrow \frac{1}{n_S} \mathbf{V}_{\mathbf{i},:}$ ▷ Average

within the ancestors and between the ancestors and the leaves, $\mathbf{T}^{(S,S)}, \mathbf{T}^{(A,A)}$ and $\mathbf{T}^{(A,S)}$, and the TOU process parameters (see Eq. 2).

As $p(\mathbf{z})$ is a multivariate Gaussian distribution, we can easily obtain the conditional distribution of $\mathbf{z}^{(A)}$ given $\mathbf{z}^{(S)}$ as follows (see Bishop (2006), page 689),

$$p(\mathbf{z}) = \mathcal{MVN}(\mathbf{z} \mid \mathbf{\Sigma}) \Rightarrow p\left(\mathbf{z}^{(A)} \mid \mathbf{z}^{(S)}\right) = \mathcal{MVN}\left(\mathbf{z}^{(A)} \mid \mu_{A|S}, \left(\Lambda^{(A,A)}\right)^{-1}\right),$$

with

$$\mu_{A|S} = \mu_A - \left(\Lambda^{(A,A)}\right)^{-1} \Lambda^{(A,S)} \left(\mathbf{z}^{(S)} - \mu^{(S)}\right) = -\left(\Lambda^{(A,A)}\right)^{-1} \Lambda^{(A,S)} \mathbf{z}^{(S)}.$$

As $\mu_{A|S}$ corresponds to the MAP of $\mathbf{z}^{(A)}$ for any given values of $\mathbf{z}^{(S)}$ and the TOU process parameters when the ancestral sequences are not observed, the MAP estimate of the latent representation of the ancestral sequences is given by,

$$\mathbf{z}^{(A),\text{MAP}} = -\left(\Lambda^{(A,A)}\right)^{-1} \Lambda^{(A,S)} \mathbf{z}^{(S),\text{MAP}}.$$

In addition, a Gaussian approximation of the posterior of $\mathbf{z}^{(A)}$ based on the MAP estimates is given by,

$$\mathbf{z}^{(A)} \sim \mathcal{MVN}\left(\mathbf{z}^{(A)} \mid \mathbf{z}^{(A),\text{MAP}}, \left(\Lambda^{(A,A)}\right)^{-1}\right). \tag{3}$$

The reconstructed ancestral sequences are subsequently obtained by applying the GRU decoder to the MAP estimates of their latent representations, as explained in Section 4.1.

### 4.5 Benchmarking

In order to assess the accuracy of the ASR we benchmark Draupnir (MAP and Marginal) against state-of-the-art phylogenetic methods. We selected methods that perform ASR using a given tree topology and given patristic distances, including one Bayesian method (PhyloBayes, Lartillot et al. (2013)) and three maximum likelihood based methods (PAML, Yang (2007); FastML, Ashkenazy et al. (2012); and IQTree, Nguyen et al. (2015)). We apply the ASR methods to both protein sequences, and to their corresponding DNA sequences followed by subsequent translation to protein sequences. For Draupnir, we use the protein sequences, which is the harder problem. We use eleven data sets with different numbers of leaves (from 19 to 800), different alignment lengths (from 63 to 558) and with or without gaps (10 and 1 data set respectively). The data sets include eight simulated data sets generated using the software EvolveAGene (Hall, 2016) and three data sets with experimentally determined ancestral sequences. Note that the simulated data sets were obtained from factored evolution models. We included a large data set with 800 leaves. For prediction with Draupnir, we either use i) the most likely sequence (Draupnir-MAP in Fig. 3), using Equation 4, ii) samples from the marginal distribution (Draupnir-Marginal in Fig. 3; we report the average identity of 50 samples), using Equation 5 or iii) samples from the variational posterior using an amortised guide (Draupnir-Variational, see Appendix A.6), using Equation 6. Details on the data sets and training can be found in Appendix A.3 and A.4, respectively.

## 5 Results

### 5.1 Latent representations and coevolution

In order to inspect the quality of the latent representations of the sequences, we use the $\beta$-lactamase family with 32 leaf sequences. We visualize t-SNE projections (Van der Maaten & Hinton, 2008) of the latent representations and compare the results with the structure of the phylogenetic tree (see Figure 2). The result indicates that the latent representations represent the structure of the tree and its different clades (subtrees) well, indicating that the TOU process performs well as an informative prior on evolutionary relationships. In Appendix A.6, we show how marginalizing over the latent representations allows Draupnir (marginal and variational) to model coevolution among sites.

### 5.2 Benchmarking Results

In Figure 3, we compare the accuracy of Draupnir (MAP and marginal) with state-of-the-art ASR methods by plotting the average percent identity between the ancestral sequences as reconstructed by Draupnir and the true sequences. The true ancestral sequences were either experimentally determined or simulated. The description and origin of the data sets can be found in Appendix A.3. Tables with benchmark results are shown in Appendix A.5.

Figure 2: Results for the $\beta$-lactamase family with 32 leaves. *Left*: t-SNE projection of the latent representations of the ancestral and leaf nodes. *Right*: The phylogenetic tree. Both plots are coloured according to clade membership.

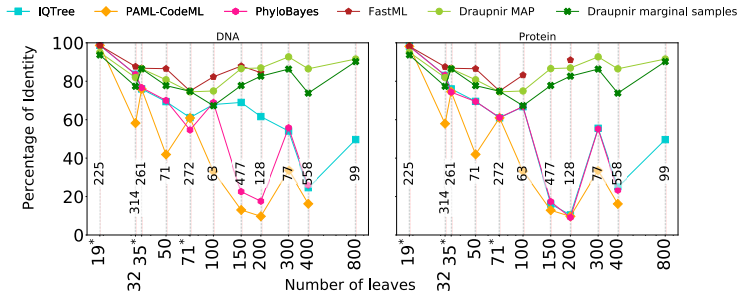
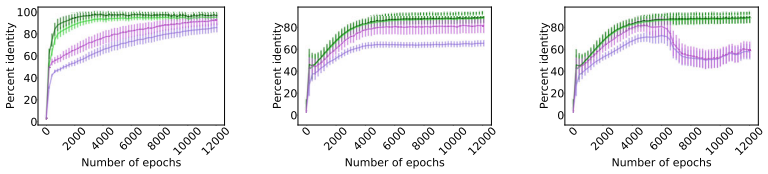



Figure 3: Comparison of the average percentage identity (y-axis) between predicted and true ancestral sequences for Draupnir (MAP and marginal) and ASR methods for data sets with different number of leaves (x-axis; experimental data sets are indicated with an asterisk). Missing points indicate that the ASR method failed to produce results on the given hardware. The alignment size is shown on the dotted lines. We compare with ASR methods using the DNA sequences (*left*) and the corresponding Protein sequences, subsequently translated to protein sequences (*right*). Tables with detailed results for Draupnir-marginal and Draupnir-MAP can be found in Appendix A.5. For the benchmarking settings see Appendix A.8.

### 5.3 Ablation studies

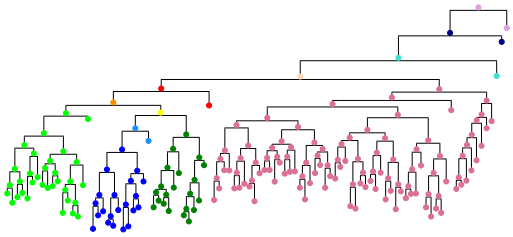
In the first ablation study, we investigate the influence of the BLOSUM embeddings by removing them as input to the GRU. Overall, the absence of the BLOSUM embeddings slows down convergence and sometimes make the learning process unstable, but ultimately does not strongly affect accuracy (see Figure 5).

In the second ablation study, we investigate the influence of the tree-structured prior by comparing with a standard VAE with a Gaussian prior. We do this by using diagonal unit covariance matrices for each of the $n_Z$ columns of the latent matrix $\mathbf{Z}$. The rest of the model was identical. We then compare the latent representations obtained for the leaf nodes. The results (see Figure 7) indicate that the standard VAE is not capable of reconstructing the evolutionary relationships well: sequences belonging to the same clade often end up far apart in latent space. This indicates that the influence of the tree-structures prior is substantial. A quantitative analysis of this ablation study can be found in Appendix A.7.




(a) Leaves, stable case (b) Ancestors, stable case (c) Ancestors, unstable case.

Figure 5: BLOSUM embedding ablation study for the 800 leaves data set. For every 100 training epochs, the average percent identity and standard deviation are plotted for all leaves (training set) or ancestors (test set), respectively. The results obtained with the BLOSUM embedding are shown in dark green (MAP) and light green (marginal, see Equation 3). The results without the BLOSUM embeddings are shown in pink (MAP) and purple (marginal).



(a) Phylogenetic tree coloured by clade membership

(b) Standard VAE, leaves (c) Leaves (d) Leaves and ancestors

Figure 7: Bottom: t-SNE projections of the latent representations for the SRC-Kinase SH3 domain with 100 leaves, obtained from a standard VAE (*left*) and Draupnir-marginal (*center* and *right*), colored by clade. Note that only the latter model can be used to infer the latent representations of the ancestral sequences. *Top*: the corresponding tree.

## 6 Discussion and future work

Draupnir demonstrates the potential value of incorporating evolutionary information and evolutionary models explicitly in deep generative models for representation learning of biological sequences. We point out that it is possible to extend the model with additional information beyond sequences, for example backbone angles describing protein structure (Golden et al., 2017) or measurements of protein stability. In future work, extending the model to genomic-size data can be done using inducing points for Gaussian processes, as explored in Jazbec et al. (2021) and Vikram et al. (2019). The case of a latent phylogenetic tree can be addressed using a coalescent point process prior (Lambert & Stadler, 2013; Vikram et al., 2019). Finally, in the large data case, the current simple network architectures can be improved with more expressive compositions such as an MSA transformer (Rao et al., 2021) or a deconvolutional model for sequences (Hawkins-Hooker et al., 2021). Finally,

we point out that the learned parameters of the TOU processes might offer interpretable information on the evolutionary process.

## REFERENCES

Naila O Alieva, Karen A Konzen, Steven F Field, Ella A Meleshkevitch, Marguerite E Hunt, Victor Beltran-Ramirez, David J Miller, Jörg Wiedenmann, Anya Salih, and Mikhail V Matz. Diversity and evolution of coral fluorescent proteins. *PLoS ONE*, 3(7):e2680, 2008.

Ethan C Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature methods*, 16(12):1315–1322, 2019.

Haim Ashkenazy, Osnat Penn, Adi Doron-Faigenboim, Ofir Cohen, Gina Cannarozzi, Oren Zomer, and Tal Pupko. FastML: a web server for probabilistic reconstruction of ancestral sequences. *Nucleic acids research*, (W1):W580–W584, 2012.

Ahmet Bakan, Anindita Dutta, Wenzhi Mao, Ying Liu, Chakra Chennubhotla, Timothy R Lezon, and Ivet Bahar. Evol and prody for bridging protein sequence evolution and structural dynamics. *Bioinformatics*, 30:2681–2683, 2014.

Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The journal of machine learning research*, 20(1):973–978, 2019.

Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

Belinda SW Chang, Karolina Jönsson, Manija A Kazmi, Michael J Donoghue, and Thomas P Sakmar. Recreating a functional ancestral archosaur visual pigment. *Molecular biology and evolution*, (9):1483–1489, 2002.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Megan F Cole and Eric A Gaucher. Utilizing natural diversity to evolve protein function: applications towards thermostability. *Current opinion in chemical biology*, (3):399–406, 2011.

Nicki Skafte Detlefsen, Søren Hauberg, and Wouter Boomsma. What is a meaningful representation of protein sequences? *arXiv preprint arXiv:2012.02679*, 2020.

Xinqiang Ding, Zhengting Zou, and Charles L Brooks III. Deciphering protein evolution and fitness landscapes with latent space models. *Nature communications*, (1):1–13, 2019.

Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

Mohammed El-Kebir, Layla Oesper, Hannah Acheson-Field, and Benjamin J Raphael. Reconstruction of clonal trees and tumor composition from multi-sample sequencing data. *Bioinformatics*, (12):i62–i70, 2015.

Brian Gaschen, Jesse Taylor, Karina Yusim, Brian Foley, Feng Gao, Dorothy Lang, Vladimir Novitsky, Barton Haynes, Beatrice H Hahn, Tanmoy Bhattacharya, et al. Diversity considerations in HIV-1 vaccine selection. *Science*, (5577):2354–2360, 2002.

Michael Golden, Eduardo García-Portugués, Michael Sørensen, Kanti V Mardia, Thomas Hamelryck, and Jotun Hein. A generative angular model of protein structure evolution. *Molecular biology and evolution*, 34(8):2085–2100, 2017.

Joe G Greener, Lewis Moffat, and David T Jones. Design of metalloproteins and novel protein folds using variational autoencoders. *Scientific reports*, pp. 1–12, 2018.

Pantelis Z Hadjipantelis, Nick S Jones, John Moriarty, David Springate, and Christopher G Knight. Ancestral inference from functional data: statistical methods and numerical examples. *arXiv preprint arXiv:1208.0628*, 2012.

Barry G Hall. Comparison of the accuracies of several phylogenetic methods using protein and DNA sequences. *Molecular biology and evolution*, 22(3):792–802, 2005.

Barry G Hall. Effects of sequence diversity and recombination on the accuracy of phylogenetic trees estimated by kSNP. *Cladistics*, (1):90–99, 2016.

Thomas F Hansen. Stabilizing selection and the comparative analysis of adaptation. *Evolution*, 51(5):1341–1351, 1997.

Alex Hawkins-Hooker, Florence Depardieu, Sebastien Baur, Guillaume Couairon, Arthur Chen, and David Bikard. Generating functional protein variants with variational autoencoders. *PLoS computational biology*, (2):e1008736, 2021.

Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the national academy of sciences*, (22):10915–10919, 1992.

Georg KA Hochberg and Joseph W Thornton. Reconstructing ancient proteins to understand the causes of structure and function. *Annual Review of Biophysics*, pp. 247–269, 2017.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of machine learning research*, 14(5), 2013.

Edwin Rodríguez Horta, Alejandro Lage-Castellanos, and Roberto Mulet. Ancestral sequence reconstruction for co-evolutionary models. *arXiv preprint arXiv:2108.03801*, 2021.

Metod Jazbec, Matt Ashman, Vincent Fortuin, Michael Pearce, Stephan Mandt, and Gunnar Rätsch. Scalable Gaussian process variational autoencoders. In *International Conference on Artificial Intelligence and Statistics*, pp. 3511–3519. PMLR, 2021.

Nick S Jones and John Moriarty. Evolutionary inference for function-valued traits: Gaussian process regression on phylogenies. *Journal of the Royal Society Interface*, 10(78):20120616, 2013.

Jeffrey B Joy, Richard H Liang, Rosemary M McCloskey, T Nguyen, and Art FY Poon. Ancestral reconstruction. *PLoS computational biology*, 12(7):e1004763, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and trends in machine learning*, 12:307–392, 2019.

Jeffrey M Koshi and Richard A Goldstein. Probabilistic reconstruction of ancestral protein sequences. *Journal of molecular evolution*, 42(2):313–320, 1996.

Amaury Lambert and Tanja Stadler. Birth–death models and coalescent point processes: The shape and probability of reconstructed phylogenies. *Theoretical population biology*, 90:113–128, 2013.

Nicolas Lartillot. A phylogenetic Kalman filter for ancestral trait reconstruction using molecular data. *Bioinformatics*, pp. 488–496, 2014.

Nicolas Lartillot, Nicolas Rodrigue, Daniel Stubbs, and Jacques Richer. PhyloBayes MPI: Phylogenetic reconstruction with infinite mixtures of profiles in a parallel environment. *Systematic biology*, (4):611–615, 2013.

Emile Mathieu, Charline Le Lan, Chris J Maddison, Ryota Tomioka, and Yee Whye Teh. Continuous hierarchical representations with Poincaré variational auto-encoders. *arXiv preprint arXiv:1901.06033*, 2019.

Jaina Mistry, Sara Chuguransky, Lowri Williams, Matloob Qureshi, Gustavo A Salazar, Erik LL Sonnhammer, Silvio CE Tosatto, Lisanna Paladin, Shriya Raj, Lorna J Richardson, et al. Pfam: The protein families database in 2021. *Nucleic Acids Research*, 49(D1):D412–D419, 2021.

Faruck Morcos, Andrea Pagnani, Bryan Lunt, Arianna Bertolino, Debora S Marks, Chris Sander, Riccardo Zecchina, José N Onuchic, Terence Hwa, and Martin Weigt. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences*, 108:E1293–E1301, 2011.

Lam-Tung Nguyen, Heiko A Schmidt, Arndt Von Haeseler, and Bui Quang Minh. IQ-TREE: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular biology and evolution*, 32(1):268–274, 2015.

Linus Pauling, Emile Zuckerkandl, T Henriksen, and R Lövstad. Chemical paleogenetics. *Acta chemica Scandinavica*, 17:S9–S16, 1963.

Ryan N Randall, Caelan E Radford, Kelsey A Roof, Divya K Natarajan, and Eric A Gaucher. An experimental phylogeny to benchmark ancestral sequence reconstruction. *Nature communications*, 7(1):1–6, 2016.

Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Xi Chen, John Canny, Pieter Abbeel, and Yun S Song. Evaluating protein transfer learning with TAPE. *Advances in neural information processing systems*, 32:9689, 2019.

Roshan Rao, Jason Liu, Robert Verkuil, Joshua Meier, John F Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives. MSA transformer. *bioRxiv*, 2021.

Adam J Riesselman, John B Ingraham, and Debora S Marks. Deep generative models of genetic variation capture the effects of mutations. *Nature methods*, 15(10):816–822, 2018.

Avery GA Selberg, Eric A Gaucher, and David A Liberles. Ancestral sequence reconstruction: From chemical paleogenetics to maximum likelihood algorithms and beyond. *Journal of Molecular Evolution*, (3):157–164, 2021.

Matthew A Spence, Joe A Kaczmarski, Jake W Saunders, and Colin J Jackson. Ancestral sequence reconstruction for protein engineers. *Current Opinion in Structural Biology*, pp. 131–141, 2021.

Max R Tolkoff, Michael E Alfaro, Guy Baele, Philippe Lemey, and Marc A Suchard. Phylogenetic factor analysis. *Systematic biology*, (3):384–399, 2018.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, (11), 2008.

Sharad Vikram, Matthew D Hoffman, and Matthew J Johnson. The LORACs prior for VAEs: Letting the trees speak for the data. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 3292–3301. PMLR, 2019.

C Wilson, RV Agafonov, M Hoemberger, S Kutter, A Zorba, J Halpin, V Buosi, R Otten, D Waterman, DL Theobald, et al. Using ancient protein kinases to unravel a modern cancer drug's mechanism. *Science*, 347(6224):882–886, 2015.

Ziheng Yang. PAML 4: Phylogenetic analysis by maximum likelihood. *Molecular biology and evolution*, 24(8):1586–1591, 2007.

Ziheng Yang, Sudhir Kumar, and Masatoshi Nei. A new method of inference of ancestral nucleotide and amino acid sequences. *Genetics*, 141(4):1641–1650, 1995.

# A Appendix

## A.1 Notation and variables

| Name | Description |
|---|---|
| $n_Z$ | Number of TOU processes, length of latent vector (30) |
| $n_L$ | Alignment length |
| $n_S$ | Number of leaf sequences |
| $n_A$ | Number of ancestral sequences ($n_A = n_S - 1$) |
| $n_C$ | Number of character types |
| $\mathbf{S}$ | Sequence alignment matrix of leaf sequences |
| $\mathbf{Z}^{(S)} \equiv \mathbf{Z}$ | Matrix of latent representations of the leaf sequences Note: We use $\mathbf{Z}$ for notational convenience where possible. |
| $\mathbf{Z}^{(A)}$ | Matrix of latent representations of the ancestral sequences |
| $\mathbf{Z}^{(A,S)}$ | Matrix of latent representations of the leaf and ancestral sequences |
| $\mathbf{T}$ | Patristic distance matrix (given) |
| $\mathbf{T}^{(S,S)}$ | Patristic distance submatrix of distances between leaf sequences (given) |
| $\mathbf{T}^{(A,A)}$ | Patristic distance submatrix of distances between ancestral sequences (given) |
| $\mathbf{T}^{(A,S)}$ | Patristic distance submatrix of distances between leaf and ancestral sequences (given) |
| $\mathbf{B}$ | BLOSUM matrix (given) |
| $\alpha$ | Vector of OU hyperprior parameters |
| $\sigma_{\mathbf{f}}$ | Vector of intensities of inherited variation (TOU process) |
| $\sigma_{\mathbf{n}}$ | Vector of intensities of specific variation (TOU process) |
| $\lambda$ | Vector of characteristic lenght-scales (TOU process) |
| $\mathbf{C}$ | Tensor of $n_Z$ TOU process covariance matrices |
| $\mathbf{V}$ | Matrix of weighted BLOSUM vectors |
| $\mathbf{E}$ | Matrix of BLOSUM embeddings in the model |
| $\mathbf{F}$ | Tensor of BLOSUM embeddings in the guide |
| $\mathbf{Y}$ | Input tensor to GRU |
| $\mathbf{H}$ | State tensor of GRU |
| $\mathbf{L}$ | Tensor of logits of the $n_C$ sequence characters |
| $\theta$ | Parameters of the neural networks and the GRU |
| $\mathbf{0}^{(d)}$ | $d$-dimensional vector of zeros |
| $\mathbf{0}^{(m\times n)}$ | $(m \times n)$-dimensional matrix of zeros |
| GRU | Gated recurrent unit |
| NN | Fully connected neural network |
| $\mathcal{HN}$ | Half-normal distribution |
| $\mathcal{MVN}$ | Multivariate Gaussian distribution |
| cat | Concatenation of two vectors. |

Table 1: Variables and notation used for the Draupnir model.

| Name | Dimensions |
|---|---|
| $\alpha$ | 3 |
| $\sigma_{\mathbf{f}},\ \sigma_{\mathbf{n}},\ \lambda$ | $n_Z$ |
| $\mathbf{T}$ | $(n_S + n_A) \times (n_S + n_A)$ |
| $\mathbf{T}^{(A,A)}$ | $n_A \times n_A$ |
| $\mathbf{T}^{(S,S)}$ | $n_S \times n_S$ |
| $\mathbf{T}^{(A,S)}$ | $n_A \times n_S$ |
| $\mathbf{C}$ | $n_Z \times n_S \times n_S$ |
| $\mathbf{Z}^{(S)} \equiv \mathbf{Z}$ | $n_S \times n_Z$ |
| $\mathbf{Z}^{(A)}$ | $n_A \times n_Z$ |
| $\mathbf{Z}^{(A,S)}$ | $(n_S + n_A) \times n_Z$ |
| $\mathbf{Y}$ | $n_S \times n_L \times (n_Z + n_C)$ |
| $\mathbf{E}$ | $n_L \times n_C$ |
| $\mathbf{F}$ | $n_S \times n_L \times n_C$ |
| $\mathbf{H}$ | $n_S \times n_L \times 60$ |
| $\mathbf{L}$ | $n_S \times n_L \times n_C$ |
| $\mathbf{B}$ | $n_C \times n_C$ |
| $\mathbf{S}$ | $n_S \times n_L$ |
| $\mathbf{V}$ | $n_L \times n_C$ |

Table 2: Dimensions of variables

### A.2 Draupnir settings

**Neural network architecture** The Draupnir model contains three neural networks (see Algorithm 1, Figure 1 and Figure 8): a fully connected network, $\mathrm{NN}_\theta^{(1)}$, that maps the pre-computed BLOSUM vectors, $\mathbf{V}$, to BLOSUM embeddings, $\mathbf{E}$; a bidirectional $\mathrm{GRU}_\theta$ (Cho et al., 2014) with a single layer that takes as input the BLOSUM embeddings and the latent vector of the $k-$th leaf sequence, $\mathbf{Z}_{k,:}$, and a second fully connected network, $\mathrm{NN}_\theta^{(2)}$, that maps the $\mathrm{GRU}_\theta$ states to the logit vectors of the sequence characters. The dimensionality of the state of the bidirectional $\mathrm{GRU}_\theta$ is $2 \times 60$.

In the guide, we re-use the neural network architectures described above: $\mathrm{NN}_\phi^{(2)}$ and $\mathrm{GRU}_\phi$ are identical to $\mathrm{NN}_\theta^{(2)}$ and $\mathrm{GRU}_\theta$, respectively, except that the output size of $\mathrm{NN}_\phi^{(2)}$ is $(2{\times}n_Z)$ instead of $n_C$, corresponding to the length of the mean vector and standard deviation vector of the latent representation. $\mathrm{NN}_\phi^{(1)}$ is identical to $\mathrm{NN}_\theta^{(1)}$.

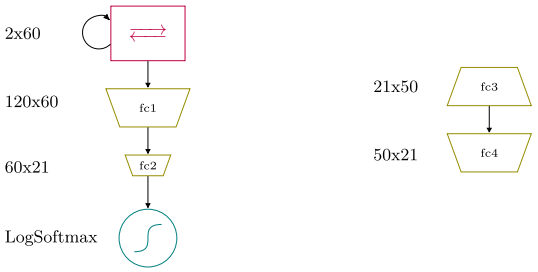



Figure 8: Architectures and dimensions of the neural networks used in Draupnir. *Left*: Architecture of the bidirectional $\mathrm{GRU}_\theta$ (red) and $\mathrm{NN}_\theta^{(2)}$. *Right*: Architecture of $\mathrm{NN}_\theta^{(1)}$. "fc" indicates a fully connected layer.

**Additional Draupnir settings** We chose BLOSUM62 as the base substitution matrix, except for the CFP datasets (71 and 35 leaves) where we use PAM70 due to the presence of special amino acids.

### A.3 Data sets

Table 3: Descriptions of the eleven data sets used for benchmarking and the leaves-only data set used in the co-evolutionary analysis. All data sets contain insertions and deletions (gaps), except the one in italic (top), which only contains substitutions. Data sets labelled with an asterisk only contain the sequence of the root node.

| Dataset | Number of leaves | Alignment length | Source |
|---|---|---|---|
| **Datasets with experimentally determined ancestral sequences** | | | |
| *Randall's Coral fluorescent proteins (CFP) benchmark* | 19 | 225 | Randall et al. (2016) |
| *Coral fluorescent proteins (CFP) Faviina subclade | 35 | 261 | *allfav* root node sequence from Alieva et al. (2008) |
| *Coral fluorescent proteins (CFP) subclade | 71 | 272 | *allcor* root node sequence from Alieva et al. (2008) |
| **EvolveAGene4 (Hall, 2005) simulations** | | | |
| Simulation $\beta$-Lactamase | 32 | 314 | GenBank accession no. AF309824 |
| Simulation Calcitonin | 50 | 71 | NBCI CCDS7820.1 |
| Simulation SRC-Kinase SH3 domain | 100 | 63 | GenBank BC011566.1 |
| Simulation Sirtuin | 150 | 477 | NBCI CCDS44412.1 |
| Simulation SRC-Kinase SH3 domain | 200 | 128 | GenBank BC011566.1 |
| Simulation PIGBOS | 300 | 77 | NBCI CCDS81884.1 |
| Simulation Insulin | 400 | 558 | NCBI BC011566.1 |
| Simulation SRC-Kinase SH3 domain | 800 | 99 | GenBank BC011566.1 |
| **Leaves only data set** | | | |
| PF00400 | 125 | 138 | PFAM family no. PF00400 |

### A.4 Training

Table 4: Training settings and running times (Intel(R) Xeon(R) Gold 6136 CPU @ 3.00GHz, Quadro RTX 6000 GPU).On the largest dataset (800 leaves), we make use of a Reduce on plateau learning scheduler combined with plating to further improve the accuracy results.

| Dataset | Epochs | Plate size | Learning rate scheduler | Model parameters | Running times |
|---|---|---|---|---|---|
| **Datasets with experimentally determined ancestral sequences** | | | | | |
| Randall's Coral fluorescent proteins (CFP) benchmark, 19 leaves | 16600 | 19 | No | 52055 | 53 min 39 s |
| Coral fluorescent proteins (CFP) Faviina subclade, 35 leaves | 23000 | 35 | No | 55181 | 1 h 36 min 1 s |
| Coral fluorescent proteins (CFP) clade, 71 leaves | 23000 | 71 | No | 52535 | 1 h 4 min |
| **EvolveAGene4 (Hall, 2005) simulations** | | | | | |
| Simulation $\beta$-Lactamase, 32 leaves | 15000 | 32 | No | 52445 | 1 h 41 min 39 s |
| Simulation Calcitonin, 50 leaves | 18700 | 50 | No | 52985 | 2 h 10 min 1 s |
| Simulation SRC-Kinase SH3 domain, 100 leaves | 21600 | 100 | No | 54485 | 2 h 42 min 10 s |
| Simulation Sirtuin, 150 leaves | 20000 | 150 | No | 55985 | 4 h 6 min 23 s |
| Simulation SRC-Kinase SH3 domain, 200 leaves | 22000 | 200 | No | 57485 | 49 min 21 s |
| Simulation PIGBOS, 300 leaves | 18000 | 300 | No | 60485 | 44 min 1 s |
| Simulation Insulin, 400 leaves | 18400 | 400 | No | 63485 | 3 h 32 min 9 s |
| Simulation SRC-Kinase SH3 domain, 800 leaves | 25000 | 50 | Yes | 141005 | 3 h 39 min 29 s |
| **Leaves only data set** | | | | | |
| PF00400, 125 leaves - Marginal | 23000 | 125 | No | 55235 | 52 min 48 s |
| PF00400, 125 leaves - Variational | 23000 | 125 | No | 137487 | 1 h 21 min 39 s |

### A.5 Benchmarking tables

Table 5: Benchmarking results using protein sequences. The table shows the means and the standard deviation of the percent identity for all the predicted ancestral sequences and their corresponding true sequences. In the case of Draupnir-MAP, the means and standard deviations are calculated using the most likely sequence of each ancestral node. In the case of Draupnir-marginal samples, they are calculated using 50 samples per ancestral node. "Not available" indicates the ASR method failed to produce results for that data set on the given hardware. The results for the standard ASR methods were obtained using the amino acid sequences.

| | Number of leaves | Alignment length | Draupnir MAP | Draupnir marginal samples | PAML-CodeML | PhyloBayes | FastML | IQTree |
|---|---|---|---|---|---|---|---|---|
| Randall's Coral fluorescent proteins (CFP) | 19 | 225 | 95.03±1.29 | 93.67±0.85 | 98.14±1.3 | 98.09±1.03 | 98.17±1.31 | 98.27±1.07 |
| Coral fluorescent proteins (CFP) clade | 71 | 272 | 74.49±1.07 | 74.78±1.09 | 60.96±0.8 | 61.17±0.85 | 74.94±1.07 | 60.96±0.8 |
| Coral fluorescent proteins (CFP) Faviina subclade | 35 | 261 | 86.49±0.98 | 86.46±1.11 | 74.56±1.14 | 74.33±0.86 | 86.76±1.27 | 76.01±1.16 |
| Simulation Beta-Lactamase | 32 | 314 | 81.92±7.97 | 77.37±7.08 | 57.91±22.39 | 83.12±6.13 | 87.52±6.28 | 83.07±6.01 |
| Simulation Calcitonin, 50 | 50 | 71 | 80.77±9.22 | 77.73±7.33 | 41.94±21.21 | 69.45±8.88 | 86.52±6.07 | 69.54±8.83 |
| Simulation SRC-Kinase SH3 domain, 100 | 100 | 63 | 74.94±10.46 | 67.27±8.92 | 33.24±17.33 | 66.78±9.35 | 83.21±8.71 | 66.76±9.17 |
| Simulation Sirtuin SIRT1, 150 | 150 | 477 | 86.59±4.9 | 77.78±3.98 | 12.87±7.16 | 17.36±1.15 | Not available | 16.09±1.36 |
| Simulation SRC-Kinase SH3 domain, 200 | 200 | 128 | 86.92±5.84 | 82.65±4.85 | 9.69±7.99 | 9.10±1.64 | 91.09±4.54 | 10.44±2.71 |
| Simulation PIGB Opposite Strand regulator | 300 | 77 | 92.69±4.08 | 86.34±3.43 | 33.57±10.39 | 55.20±6.12 | Not available | 55.53±5.95 |
| Simulation Insulin Factor like | 400 | 558 | 86.48±4.06 | 73.81±3.16 | 16.17±8.37 | 23.30±1.74 | Not available | 25.22±1.61 |
| Simulation SRC-Kinase SH3 domain, 800 | 800 | 99 | 91.57±4.3 | 90.24±3.63 | Not available | Not available | Not available | 49.63±3.6 |

Table 6: Benchmarking results using DNA sequences. The table is similar to Table 5 but the reconstructions for the standard ASR methods were obtained using DNA instead of amino acid sequences. The DNA sequences were subsequently translated into protein sequences before comparison.

| | Number of leaves | Alignment length | Draupnir MAP | Draupnir marginal samples | PAML-CodeML | PhyloBayes | FastML | IQTree |
|---|---|---|---|---|---|---|---|---|
| Randall's Coral fluorescent proteins (CFP) | 19 | 225 | 95.03±1.29 | 93.67±0.85 | 98.69±0.82 | 98.82±0.83 | 98.59±0.77 | 98.69±0.76 |
| Coral fluorescent proteins (CFP) clade | 71 | 272 | 74.49±1.07 | 74.78±1.09 | 60.88±0.85 | 54.65±0.71 | 74.94±1.07 | 61.10±0.79 |
| Coral fluorescent proteins (CFP) Faviina subclade | 35 | 261 | 86.49±0.98 | 86.46±1.11 | 76.01±1.16 | 76.67±1.13 | 86.76±1.27 | 76.01±1.16 |
| Simulation Beta-Lactamase | 32 | 314 | 81.92±7.97 | 77.37±7.08 | 58.21±22.68 | 83.61±6.15 | 87.66±6.3 | 83.65±6.13 |
| Simulation Calcitonin, 50 | 50 | 71 | 80.77±9.22 | 77.73±7.33 | 41.88±21.14 | 70.01±8.92 | 86.55±6.29 | 69.42±8.89 |
| Simulation SRC-Kinase SH3 domain, 100 | 100 | 63 | 74.94±10.46 | 67.27±8.92 | 33.06±17.56 | 68.90±9.24 | 82.31±9.9 | 67.90±9.31 |
| Simulation Sirtuin SIRT1, 150 | 150 | 477 | 86.59±4.9 | 77.78±3.98 | 13.02±7.25 | 22.54±2.08 | 87.92±8.4 | 68.98±0.79 |
| Simulation SRC-Kinase SH3 domain, 200 | 200 | 128 | 86.92±5.84 | 82.65±4.85 | 9.72±8.17 | 17.60±2.53 | 84.48±8.5 | 61.62±2.71 |
| Simulation PIGB Opposite Strand regulator | 300 | 77 | 92.69±4.08 | 86.34±3.43 | 33.64±10.35 | 55.79±5.23 | Not available | 54.10±5.03 |
| Simulation Insulin Factor like | 400 | 558 | 86.48±4.06 | 73.81±3.16 | 16.23±8.4 | 26.02±2.22 | Not available | 24.63±3.94 |
| Simulation SRC-Kinase SH3 domain, 800 | 800 | 99 | 91.57±4.3 | 90.24±3.63 | Not available | Not available | Not available | 49.66±3.68 |

### A.6 Coevolution analysis

Conventional ASR methods use models that assume factorized evolution (Horta et al., 2021), that is, they assume that each site evolves independently of all other sites in the sequence. In reality, some sites are coupled in evolution, resulting in dependencies between sites. This phenomenon is called epistasis (Hochberg & Thornton, 2017). Draupnir is a model that goes beyond factorized evolution, and thus potentially models coevolving sites. Here, we analyze to what extend this is indeed the case.

In order to evaluate modelling of coevolution, we make use of direct coupling analysis (DCA) (Morcos et al., 2011). DCA identifies coevolving pairs of residues that directly influence each other by calculating a quantity called Direct Information (DI), which is obtained by fitting a Markov random field. We calculated the DI using *ProDy* ((Bakan et al., 2014)). As data set, we used 125 sequences from the WB40 domain of the PF00400 family from the PFAM data base Mistry et al. (2021).

The DIs of the leaf sequences serve as ground truth and are compared with sequences sampled from Draupnir at the root node in three different ways (see below). If coevolution is at least partially modelled, the DIs of the leaf sequences will be similar (but not completely identical) to the DIs of the sampled sequences at the root node. We sample sequences from Draupnir using three different methods.

The first method (MAP) simply uses the MAP estimates of the probability vectors at each position:

$$\mathbf{a} \sim \prod_{i=1}^{n_L} p\left(a_i \mid \boldsymbol{\theta}_i^{(\mathrm{MAP})}\right), \tag{4}$$

where $\mathbf{a}$ is an ancestral sequence and $\boldsymbol{\theta}_i^{(\mathrm{MAP})}$ is the MAP estimate of the amino acid probability vector at position $i$ for that sequence. This baseline results in independent sites. Picking the most likely amino acid at each position according to the above expression corresponds to Draupnir -MAP in Section 5.2.

The second method (Draupnir-marginal) makes use of the MAP estimates of the leaf representations but marginalizes over the ancestral representations (see Section 4.4):

$$\mathbf{a} \sim \int \left( \prod_{i=1}^{n_L} p\left(a_i \mid \left[\boldsymbol{\theta}\left(\mathbf{z}^{(a)}\right)\right]_i\right)\right) p(\mathbf{z}^{(a)} \mid \mathbf{Z}^{(\mathrm{MAP})}) \mathrm{d}\mathbf{z}^{(a)}, \tag{5}$$

where the probability vectors $\boldsymbol{\theta}_i$ are obtained from a GRU applied to the latent representation $\mathbf{z}^{(a)}$ of the ancestral sequence $\mathbf{a}$ (see Algorithm 1). The last factor involves conditional Gaussian distributions (see Equation 3 and Section 4.4). Integrating over $\mathbf{z}^{(a)}$ introduces correlations along the sequence. Therefore, this method is in principle capable to model some coevolution as we integrate over the latent representations of the ancestors (while using a point estimate for the latent representations of the leaves). This corresponds to the "Draupnir marginal samples" in Section 5.2.

In the third method (Variational), we make use of a guide $q_\phi(\mathbf{Z} \mid \mathbf{S})$ to obtain a variational posterior over the latent representations $\mathbf{Z}$ (see Algorithm 3):

$$\mathbf{a} \sim \int \left( \prod_{i=1}^{n_L} p\left(a_i \mid \left[\boldsymbol{\theta}\left(\mathbf{z}^{(a)}\right)\right]_i\right)\right) p(\mathbf{z}^{(a)} \mid \mathbf{Z}) q_\phi(\mathbf{Z} \mid \mathbf{S}) \mathrm{d}\mathbf{z}^{(a)} \mathrm{d}\mathbf{Z}. \tag{6}$$

In this case, we marginalize over the latent representations of both leaves and ancestors. This method should capture the coevolutionary signal to a greater extent than the second method.

The results are shown in Figure 9. As expected the third method, (Variational) does best, while the first method (MAP) does worst. The Variational method improves considerably upon the Marginal method, indicating that replacing the MAP estimate with a variational distribution for the latent representations of the leaves has significant impact. The results indicate that Draupnir indeed to a significant extent can capture coevolutionary information.

**Algorithm 3** Architecture of the variational guide, $q_\phi(\mathbf{Z} \mid \mathbf{S})$. We use point estimates for the hyperparameters and parameters of the TOU processes, and multivariate diagonal Gaussian distributions for the latent representations. An asterisk indicates the value of a point estimate. $\delta(.)$ is the Dirac delta function.

**Require:** Multiple sequence alignment $\mathbf{S}$

**for** $j \in \{1, 2, 3\}$ **do** ▷ Point estimates of TOU hyperprior parameters
  $\alpha_j \sim \delta(\alpha_j^*)$

**for** $h \in \{1, ..., n_Z\}$ **do** ▷ Point estimates of the parameters of the $n_Z$ TOU processes
  $\sigma_{f,h} \sim \delta(\sigma_{f,h}^*)$
  $\sigma_{n,h} \sim \delta(\sigma_{n,h}^*)$
  $\lambda_h \sim \delta(\lambda_h^*)$

**for** $k \in [1, \ldots, n_S]$ **do**
  **for** $i \in [1, \ldots, n_L]$ **do**
    $r \leftarrow S_{k,i}$ ▷ Amino acid at position $i$ in leaf sequence $k$
    $\mathbf{F}_{k,i,:} \leftarrow \mathrm{NN}_\phi^{(1)}(\mathbf{B}_{r,:})$ ▷ BLOSUM embedding

**for** $k \in [1, \ldots, n_S]$ **do**
  $\mathbf{H}_{k,:,:} \leftarrow \mathrm{GRU}_\phi(\mathbf{F}_{k,:,:})$ ▷ Bidirectional GRU states
  $\mathbf{m}, \mathbf{c} \leftarrow \mathrm{NN}_\phi^{(2)}(\mathbf{H}_{k,-1,:})$ ▷ Mean and standard deviations of $\mathbf{Z}_{k,:}$
  $\mathbf{Z}_{k,:} \sim \mathcal{MVN}(\mathbf{m}, \mathrm{diag}(\mathbf{c}))$

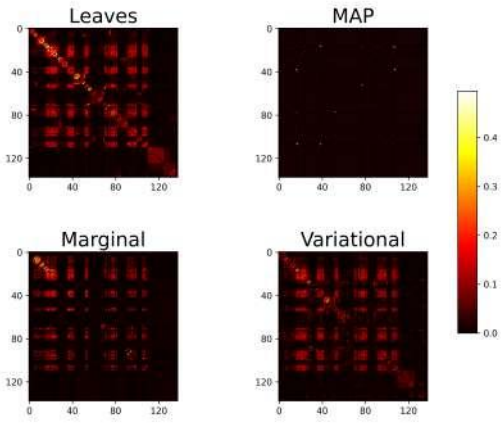



Figure 9: DI values for all position pairs of the WB40 data set obtained from the leaf sequences (upper left) and sequences sampled at the root node (other plots) using the MAP, Marginal and Variational methods. We sampled 125 root sequences, which is equal to the number of leaves. The correlation coefficients between the DIs of the leaves and the sampled sequences are 0.05 (MAP), 0.66 (Marginal) and 0.79 (Variational).

### A.7 Quantitative analysis of the latent space representations

In order to compare the standard VAE with Draupnir in a quantitative way (see Figure 7), we analyze the correlation between (a) the Euclidean distances between the latent representations of the leaves and (b) the corresponding branch lengths in the phylogenetic tree for both models. The results are shown in Figure 10.

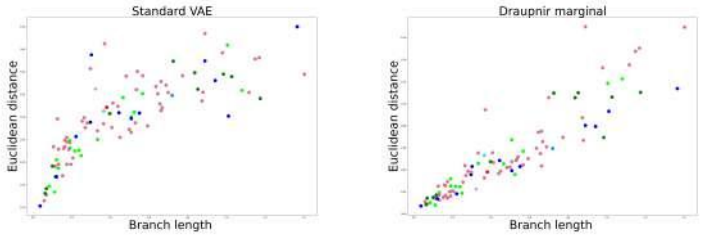



Figure 10: Comparison of the Euclidean distances between the latent representations of the leaves (y-axis) and the corresponding branch lengths in the phylogenetic tree (x-axis). We use the same color scheme as in Figure 2. We traverse the tree in level order and assign the colour of the clade of the first leaf. (*Left*) The standard VAE. The correlation coefficient is 0.79; the Spearman correlation coefficient is 0.85. (*Right*) Draupnir. The correlation coefficient is 0.91; the Spearman correlation coefficient is 0.94.

### A.8 Benchmark settings

**PAML-CodeML settings** PAML-CodeML (provided by Biopython 1.78) was used with the following settings:

| | |
|---|---|
| verbose | 2 (includes detailed information of the posterior probabilities per node) |
| runmode | 0 (utilize given tree) |
| seqtype | 2 (amino acids) |
| clock | 0 (no molecular clock, genes are evolving at different rates) |
| aaDist | 0 |
| getSE | 0 |
| RateAncestor | 1 |
| aaRateFile | WAG |
| method | 1 |
| model | 2 (more dn/ds (selection coefficient) ratios per branch) |
| fix_alpha | 0 (estimate gamma) |
| alpha | 0.5 (initial alpha value for gamma distribution) |
| fix_blength | 0 (keep given branch lengths) |

**PhyloBayes 4.1 settings**

pd -s -f -T *treefile* -cat -gtr -d *alignmentfile chainname*

In the case of PhyloBayes, as recommended, we run 2 Markov chains until the convergence criteria are met. The recommended convergence criteria are minimum effective sample size above 300 and *max diff* among the chains below 0.1. Both for evaluation of the convergence of the chains and for sampling the ancestral sequences we use 100 samples.

**FastML v3.11 settings**

perl fastml –MSA-file *alignmentfile* –seqType *aa* or *nuc* –SubMatrix *WAG or GTR* –indelReconstruction *ML* –Tree *treefile*

**IQ-Tree v2.0.3 settings**
For IQ-Tree (multicore version 2.0.3), model choice and settings are automatically optimized by using the options "-m TEST" and "-nt AUTO".

iqtree -s *alignmentfile* -m *TEST* -asr -te *treefile* -nt *AUTO*

# 17 Conclusions

During the development of this thesis, I have combined Bayesian statistics and machine learning techniques to design statistical models that improve the existing solutions over several biological issues. I have implemented a new Bayesian approach to protein superposition that can be used as an optimizable error function for protein structure prediction (Theseus-PP) [1, 3]. I have collaborated on the implementation of a new model (BIFROST) for sampling protein fragments that provide accurate local structure reconstruction and uncertainty over the predictions [6]. I have developed an alternative method for performing Ancestral Sequence Reconstruction, Draupnir, which for the first time is performed using a non-factorized evolution model [8].

Theseus-PP [1, 3] conveys a new solution for protein superposition that accounts for differences among regions with high variability versus conserved ones. The design of the algorithm allows to recognize partially correct protein predictions during the model training process. This approach challenges the conventional Kabsch-RMSD solution (see Section 11) that assumes invariability across the positions in the protein. Theseus-PP is expected to perform as an error likelihood function within models for protein structure prediction such as BIFROST [6] to boost their prediction results.

BIFROST [6] is a deep probabilistic model of local structure designed to generate protein fragment libraries. The probabilistic nature of the model offers a confidence score that allows selecting which fragments will be considered for the final protein prediction. The model functions relying solely on the information gathered from the dihedral angles, without the need for Multiple Sequence Alignments or additional information that might not be available for training. The model offers quality predictions on par with ROSETTA [24, 24] at a fraction of the time.

Draupnir [8] approaches Ancestral Protein Reconstruction (ASR) by explicitly modelling the evolutionary process that takes place in a given phylogeny. This new approach relies on the usage of an informative prior over the latent space of an ordinary VAE [119] (see Section 10.2). The Ornstein-Uhlenbeck process is a type of stochastic process (see Section 12) that has been shown to capture the evolutionary process along a tree [163]. We show how Draupnir is on par with state-of-the-art ASR methods and can account for epistasis [164] by modelling co-evolving sites.

The next steps within the Phylogeny inference department could continue to challenge the current algorithms to capture more realistic and accurately the evolutionary information available from the large on-growing genomics field. Recently, new methods for non-discretized and non-Euclidean representations of the tree [165] have been developed. These new techniques are based on hyperbolic geometry and hierarchical clustering [166, 167, 168] and are designed to learn the hyperbolic manifold that describes the phylogenetic tree. Hierarchical clustering is the algorithm employed by UPGMA (see Section 6.3) to perform a heuristic approach to phylogeny reconstruction, which is based on average linkage. Hyperbolic geometry [169] is utilized as a continuous representation of the tree in pursuit of achieving an exponential growth in the number of nodes as the tree depth increases [170]. These new proposed methods open a new paradigm of solutions within Phylogenetics.

On the other hand, is important to note that UPGMA [55] based methods give an unreliable solution over the phylogeny, whereas Neighbour Joining (NJ) [58] (see Section 6.3) offers more trustworthy predictions. The Neighbour Joining method offers non-ultrametric predictions over the branch lengths, which allows for a diversity of evolutionary rates across taxa without cutting back on computer performance. I suggest investigating an alternative model that utilizes this approach will offer a superior solution.

Furthermore, combining these new technologies with Gaussian processes would achieve simultaneous inference of the phylogenetic tree, in a fully continuous and differentiable fashion, and accurate ancestral sequence reconstruction. This approach will allow performing inference over the ancestor's sequences without relying on a fixed phylogeny. Regarding the problem of the scalability of the Gaussian process, there are new advances in the field. Recently it has been proposed to sub-divide the stochastic process by partitioning the input data in a sequential manner [171]. This method reduces the computational complexity of the process from quadratic $\mathcal{O}(n^3)$ down to linear $\mathcal{O}(n)$, opening a new realm for stochastic processes.

# References

[1] Lys Sanz Moreta, Ahmad Salim Al-Sibahi, Douglas Theobald, William Bullock, Basile Nicolas Rommes, Andreas Manoukian, and Thomas Hamelryck. A probabilistic programming approach to protein structure superposition. In *2019 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–5. IEEE, 2019.

[2] Douglas L. Theobald and Deborah S. Wuttke. Accurate structural correlations from maximum likelihood superpositions. *PLoS Computational Biology*, 4(2), 2008.

[3] Lys Sanz Moreta, Ahmad Salim Al-Sibahi, and Thomas Hamelryck. Bayesian protein superposition using hamiltonian monte carlo. In *2020 IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 1–11. IEEE, 2020.

[4] Matthew D. Hoffman and Andrew Gelman. The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15:1593–1623, 2014.

[5] Deep Markov Model. Deep Markov Model. pages 1–21, 2019.

[6] Christian B Thygesen, Christian Skjødt Steenmans, Ahmad Salim Al-Sibahi, Lys Sanz Moreta, Anders Bundgård Sørensen, and Thomas Hamelryck. Efficient Generative Modelling of Protein Structure Fragments using a Deep Markov Model. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10258–10267. PMLR, 2021.

[7] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[8] Lys Sanz Moreta, Ola Rønning, Ahmad Salim Al-Sibahi, Jotun Hein, Douglas Theobald, and Thomas Hamelryck. Ancestral protein sequence reconstruction using a tree-structured ornstein-uhlenbeck variational autoencoder.

[9] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.

[10] Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in numpyro. *arXiv preprint arXiv:1912.11554*, 2019.

[11] Yuan Zhang, Peizhao Li, Feng Pan, Hongfu Liu, Pengyu Hong, Xiuwen Liu, and Jinfeng Zhang. Applications of alphafold beyond protein structure prediction. *bioRxiv*, 2021.

[12] Christian B Anfinsen. Principles that govern the folding of protein chains. *Science*, 181(4096):223–230, 1973.

[13] Peter D Sun, Christine E Foster, and Jeffrey C Boyington. Overview of protein structural and functional folds. *Current protocols in protein science*, 35(1):17–1, 2004.

[14] Ardala Breda, Napoleao Fonseca Valadares, Osmar Norberto de Souza, and Richard Charles Garratt. Protein structure, modelling and applications. In *Bioinformatics in tropical disease research: a practical and case-study approach [Internet]*. National Center for Biotechnology Information (US), 2007.

[15] Tim Harder, Wouter Boomsma, Martin Paluszewski, Jes Frellsen, Kristoffer E Johansson, and Thomas Hamelryck. Beyond rotamers: a generative, probabilistic model of side chains in proteins. *BMC bioinformatics*, 11(1):1–13, 2010.

[16] Wouter Boomsma, Kanti V Mardia, Charles C Taylor, Jesper Ferkinghoff-Borg, Anders Krogh, and Thomas Hamelryck. A generative, probabilistic model of local protein structure. *Proceedings of the National Academy of Sciences*, 105(26):8932–8937, 2008.

[17] Brian Kuhlman and Philip Bradley. Advances in protein structure prediction and design. *Nature Reviews Molecular Cell Biology*, 20(11):681–697, 2019.

[18] Elena A Ponomarenko, Ekaterina V Poverennaya, Ekaterina V Ilgisonis, Mikhail A Pyatnitskiy, Arthur T Kopylov, Victor G Zgoda, Andrey V Lisitsa, and Alexander I Archakov. The size of the human proteome: the width and depth. *International journal of analytical chemistry*, 2016, 2016.

[19] Raymond C Stevens. The cost and value of three-dimensional protein structure. *Drug Discovery World*, 4(3):35–48, 2003.

[20] Andrei N Lupas, Joana Pereira, Vikram Alva, Felipe Merino, Murray Coles, and Marcus D Hartmann. The breakthrough in protein structure prediction. *Biochemical Journal*, 478(10):1885–1890, 2021.

[21] Wenlin Li, R Dustin Schaeffer, Zbyszek Otwinowski, and Nick V Grishin. Estimation of uncertainties in the global distance test (gdt_ts) for casp models. *PloS one*, 11(5):e0154786, 2016.

[22] Robin Pearce and Yang Zhang. Toward the solution of the protein structure prediction problem. *Journal of Biological Chemistry*, page 100870, 2021.

[23] Jad Abbass and Jean-Christophe Nebel. Rosetta and the journey to predict proteins' structures, 20 years on. *Current Bioinformatics*, 15(6):611–628, 2020.

[24] Kim T Simons, Charles Kooperberg, Enoch Huang, and David Baker. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and bayesian scoring functions. *Journal of molecular biology*, 268(1):209–225, 1997.

[25] Kim T Simons, Rich Bonneau, Ingo Ruczinski, and David Baker. Ab initio protein structure prediction of casp iii targets using rosetta. *Proteins: Structure, Function, and Bioinformatics*, 37(S3):171–176, 1999.

[26] Faruck Morcos, Andrea Pagnani, Bryan Lunt, Arianna Bertolino, Debora S Marks, Chris Sander, Riccardo Zecchina, José N Onuchic, Terence Hwa, and Martin Weigt. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences*, 108(49):E1293–E1301, 2011.

[27] Martin Weigt, Robert A White, Hendrik Szurmant, James A Hoch, and Terence Hwa. Identification of direct residue contacts in protein–protein interaction by message passing. *Proceedings of the National Academy of Sciences*, 106(1):67–72, 2009.

[28] Bryan Lunt, Hendrik Szurmant, Andrea Procaccini, James A Hoch, Terence Hwa, and Martin Weigt. Inference of direct residue contacts in two-component signaling. *Methods in enzymology*, 471:17–41, 2010.

[29] Sheng Wang, Siqi Sun, Zhen Li, Renyu Zhang, and Jinbo Xu. Accurate de novo prediction of protein contact map by ultra-deep learning model. *PLoS computational biology*, 13(1):e1005324, 2017.

[30] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[31] Eduard Porta-Pardo, Victoria Ruiz-Serra, and Alfonso Valencia. The structural coverage of the human proteome before and after alphafold. *bioRxiv*, 2021.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[33] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[34] Michael Eisenstein et al. Artificial intelligence powers protein-folding predictions. *Nature*, 599(7886):706–708, 2021.

[35] Minkyung Baek, Frank DiMaio, Ivan Anishchenko, Justas Dauparas, Sergey Ovchinnikov, Gyu Rie Lee, Jue Wang, Qian Cong, Lisa N Kinch, R Dustin Schaeffer, et al. Accurate prediction of protein structures and interactions using a 3-track network. *bioRxiv*, 2021.

[36] Rainer Merkl and Reinhard Sterner. Ancestral protein reconstruction: techniques and applications. *Biological chemistry*, 397(1):1–21, 2016.

[37] Wayne J Becktel and John A Schellman. Protein stability curves. *Biopolymers: Original Research on Biomolecules*, 26(11):1859–1877, 1987.

[38] Talking Glossary of Genetic Terms | NHGRI.

[39] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 4 1998.

[40] Joseph Felsenstein. The number of evolutionary trees. *Systematic zoology*, 27(1):27–33, 1978.

[41] David Sankoff and Robert J Cedergren. Simultaneous comparison of three or more sequences related by a tree. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison/edited by David Sankoff and Joseph B. Krustal*, 1983.

[42] Walter M Fitch. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416, 1971.

[43] RL Graham and LR Foulds. Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Mathematical Biosciences*, 60(2):133–142, 1982.

[44] Joseph Felsenstein. Cases in which parsimony or compatibility methods will be positively misleading. *Systematic zoology*, 27(4):401–410, 1978.

[45] Daniele Catanzaro. The minimum evolution problem: Overview and classification. *Networks: An International Journal*, 53(2):112–125, 2009.

[46] DG Kendall, FR Hodson, and P Tautu. Mathematics in the archaeological and historical sciences. *Edinburgh Uni*, 1971.

[47] Luigi L Cavalli-Sforza and Anthony WF Edwards. Phylogenetic analysis. models and estimation procedures. *American journal of human genetics*, 19(3 Pt 1):233, 1967.

[48] Walter M Fitch and Emanuel Margoliash. Construction of phylogenetic trees. *Science*, 155(3760):279–284, 1967.

[49] Michael Bulmer. Use of the method of generalized least squares in reconstructing phylogenies from sequence data. 1991.

[50] Ranajit Chakraborty. Estimation of time of divergence from phylogenetic studies. *Canadian journal of genetics and cytology*, 19(2):217–223, 1977.

[51] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

[52] Richard Desper and Olivier Gascuel. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. In *International Workshop on Algorithms in Bioinformatics*, pages 357–374. Springer, 2002.

[53] Richard Desper and Olivier Gascuel. Theoretical foundation of the balanced minimum evolution method of phylogenetic inference and its relationship to weighted least-squares tree fitting. *Molecular Biology and Evolution*, 21(3):587–598, 2004.

[54] David L Swofford. Paupˆ*: Phylogenetic analysis using parsimony (and other methods) 4.0 b8. *Sinauer, Sunderland, MA*, 2001.

[55]

[56] Emile Zuckerkandl and Linus Pauling. Molecules as documents of evolutionary history. *Journal of theoretical biology*, 8(2):357–366, 1965.

[57] Akashnil Dutta, Albert Wang, Mashaal Sohail, Guo-Liang Chew, and Sara Baldwin. 6.047/6.878 lecture 20: Molecular evolution and phylogenetics. 2012.

[58] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.

[59] Joseph Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of molecular evolution*, 17(6):368–376, 1981.

[60] Laurent Gatto, Daniele Catanzaro, and Michel C Milinkovitch. Assessing the applicability of the gtr nucleotide substitution model through simulations. *Evolutionary Bioinformatics*, 2:117693430600200020, 2006.

[61] Thomas H Jukes, Charles R Cantor, et al. Evolution of protein molecules. *Mammalian protein metabolism*, 3:21–132, 1969.

[62] Masami Hasegawa, Hirohisa Kishino, and Taka-aki Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *Journal of molecular evolution*, 22(2):160–174, 1985.

[63] Koichiro Tamura and Masatoshi Nei. Estimation of the number of nucleotide substitutions in the control region of mitochondrial dna in humans and chimpanzees. *Molecular biology and evolution*, 10(3):512–526, 1993.

[64] Joseph Felsenstein and Gary A Churchill. A hidden markov model approach to variation among sites in rate of evolution. *Molecular biology and evolution*, 13(1):93–104, 1996.

[65] Simon Tavaré et al. Some probabilistic and statistical problems in the analysis of dna sequences. *Lectures on mathematics in the life sciences*, 17(2):57–86, 1986.

[66] Bret Larget and Donald L Simon. Markov chain monte carlo algorithms for the bayesian analysis of phylogenetic trees. *Molecular biology and evolution*, 16(6):750–759, 1999.

[67] Nicolas Lartillot. The bayesian approach to molecular phylogeny, 2020.

[68] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[69] Ziheng Yang. Paml 4: phylogenetic analysis by maximum likelihood. *Molecular biology and evolution*, 24(8):1586–1591, 2007.

[70] Tal Pupko, Itsik Pe, Ron Shamir, and Dan Graur. A fast algorithm for joint reconstruction of ancestral amino acid sequences. *Molecular biology and evolution*, 17(6):890–896, 2000.

[71] Thomas Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, (53):370–418, 1763.

[72] Sharon Bertsch McGrayne. *The theory that would not die*. Yale University Press, 2011.

[73] Andrew I Dale. Bayes or laplace? an examination of the origin and early applications of bayes' theorem. *Archive for History of Exact Sciences*, pages 23–47, 1982.

[74] Gael M Martin, David T Frazier, and Christian P Robert. Approximating bayes in the 21st century. *arXiv preprint arXiv:2112.10342*, 2021.

[75] Danielle Navarro. A personal essay on bayes factors. 2020.

[76] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[77] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[78] D. Heckerman. Bayesian Graphical Models and Networks. *International Encyclopedia of the Social & Behavioral Sciences*, pages 1048–1052, 1 2001.

[79] Christopher M Bishop. Pattern recognition. *Machine learning*, 128(9), 2006.

[80] Kevin Murphy. An introduction to graphical models. *Rap. tech*, 96:1–19, 2001.

[81] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[82] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

[83] Alperen Degirmenci. Introduction to hidden markov models. *Harvard University*, pages 1–5, 2014.

[84] Arnoldo Frigessi and Bernd Heidergott. Markov Chains. In Miodrag Lovric, editor, *International Encyclopedia of Statistical Science*, pages 772–775. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[85] Arnoldo Frigessi and Bernd Heidergott. Markov chains., 2011.

[86] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[87] John-Paul Hosom. Speech Recognition. *Encyclopedia of Information Systems*, pages 155–169, 2003.

[88] Christian Kohlschein. An introduction to hidden markov models. In *Probability and Randomization in Computer Science. Seminar in winter semester*, volume 2007, 2006.

[89] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[90] Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

[91] Leonard E Baum et al. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3(1):1–8, 1972.

[92] Lloyd R Welch. Hidden markov models and the baum-welch algorithm. ieee information theory society newsletter 53 (4)(december 2003).

[93] Maya R Gupta and Yihua Chen. *Theory and use of the EM algorithm*. Now Publishers Inc, 2011.

[94] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[95] Frederick Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976.

[96] Jüri Lember and Alexey Koloydenko. The adjusted viterbi training for hidden markov models. *Bernoulli*, 14(1):180–206, 2008.

[97] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[98] Ruslan Leont'evich Stratonovich. Conditional markov processes. In *Non-linear transformations of stochastic processes*, pages 427–453. Elsevier, 1965.

[99] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[100] Todd K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, 1996.

[101] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

[102] Roger Eckhardt, Stan Ulam, and Jhon Von Neumann. the monte carlo method. *Los Alamos Science*, 15:131, 1987.

[103] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43, 2003.

[104] Nicholas Metropolis et al. The beginning of the monte carlo method. *Los Alamos Science*, 15(584):125–130, 1987.

[105] John Von Neumann. Various techniques used in connection with random digits. *John von Neumann, Collected Works*, 5:768–770, 1963.

[106] Andrew W Marshall. The use of multi-stage sampling schemes in monte carlo computations. Technical report, RAND CORP SANTA MONICA CALIF, 1954.

[107] Malvin H Kalos and Paula A Whitlock. *Monte carlo methods*. John Wiley & Sons, 2009.

[108] Carolyn J Anderson. Central limit theorem. *The Corsini Encyclopedia of Psychology*, pages 1–2, 2010.

[109] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[110] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

[111] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984.

[112] David Spiegelhalter, Andrew Thomas, Nicky Best, and Wally Gilks. Bugs 0.5: Bayesian inference using gibbs sampling manual (version ii). *MRC Biostatistics Unit, Institute of Public Health, Cambridge, UK*, pages 1–59, 1996.

[113] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[114] Peter Young. The leapfrog method and other symplectic algorithms for integrating newton's laws of motion, 2014.

[115] John R. Taylor. Classical mechanics. 2005.

[116] Charles W Fox and Stephen J Roberts. A tutorial on variational bayesian inference. *Artificial intelligence review*, 38(2):85–95, 2012.

[117] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[118] Tom Minka et al. Divergence measures and message passing. Technical report, Citeseer, 2005.

[119] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[120] Matthew D Hoffman and Matthew J Johnson. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, volume 1, 2016.

[121] Xitong Yang. Understanding the variational lower bound. *In: variational lower bound, ELBO, hard attention*, pages 1–4, 2017.

[122] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[123] Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial intelligence and statistics*, pages 814–822. PMLR, 2014.

[124] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[125] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[126] Edwin K. P. Chong and Stanislaw H. Żak. Gradient Methods. *An Introduction to Optimization*, pages 125–153, 10 2011.

[127] BK Lavine and TR Blank. Feed-forward neural networks. 2009.

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

[129] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[130] James C Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 65. John Wiley & Sons, 2005.

[131] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[132] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[133] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[134] David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *arXiv preprint arXiv:1301.1299*, 2013.

[135] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[136] Peter W Glynn and Pierre L'ecuyer. Likelihood ratio gradient estimation for stochastic recursions. *Advances in applied probability*, 27(4):1019–1053, 1995.

[137] George Casella and Christian P Robert. Rao-blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.

[138] Sheldon Ross. *Simualtion*.

[139] Paul Glasserman and Yu-Chi Ho. *Gradient estimation via perturbation analysis*, volume 116. Springer Science & Business Media, 1991.

[140] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[141] Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational bayes for non-conjugate inference. In *International conference on machine learning*, pages 1971–1979. PMLR, 2014.

[142] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28:2575–2583, 2015.

[143] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[144] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[145] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[146] Andrea Asperti and Matteo Trentin. Balancing reconstruction error and kullback-leibler divergence in variational autoencoders. *IEEE Access*, 8:199440–199448, 2020.

[147] Brooks Paige and Frank Wood. Inference networks for sequential monte carlo in graphical models. In *International Conference on Machine Learning*, pages 3040–3049. PMLR, 2016.

[148] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 32(5):922–923, 1976.

[149] Wolfgang Kabsch. A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 34(5):827–828, 1978.

[150] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(04):376–380, 1991.

[151] W Thompson d'Arcy. On growth and form. *Cambridge: Cambridge University Press*, 16:794, 1917.

[152] F James Rohlf and Dennis Slice. Extensions of the procrustes method for the optimal superimposition of landmarks. *Systematic biology*, 39(1):40–59, 1990.

[153] Evangelos A Coutsias, Chaok Seok, and Ken A Dill. Using quaternions to calculate rmsd. *Journal of computational chemistry*, 25(15):1849–1857, 2004.

[154] Jim Lawrence, Javier Bernal, and Christoph Witzgall. A purely algebraic justification of the kabsch-umeyama algorithm. *arXiv preprint arXiv:1902.03138*, 2019.

[155] Albert Einstein. *Investigations on the Theory of the Brownian Movement*. Courier Corporation, 1956.

[156] Jarosław Piasecki. Centenary of marian smoluchowski's theory of brownian motion. *Acta Physica Polonica B*, 38(5), 2007.

[157] Jean Perrin. *Les atomes*. Cnrs, 2014.

[158] Ping Li and Songcan Chen. A review on gaussian process latent variable models. *CAAI Transactions on Intelligence Technology*, 1(4):366–376, 2016.

[159] David JC MacKay et al. Introduction to gaussian processes. *NATO ASI series F computer and systems sciences*, 168:133–166, 1998.

[160] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[161] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257, 2006.

[162] Min Yang, Wenting Tu, Wenpeng Yin, and Ziyu Lu. Deep markov neural network for sequential data classification. *ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference*, 2:32–37, 2015.

[163] Nick S Jones and John Moriarty. Evolutionary inference for function-valued traits: Gaussian process regression on phylogenies. *Journal of the Royal Society Interface*, 10(78):20120616, 2013.

[164] Georg KA Hochberg and Joseph W Thornton. Reconstructing ancient proteins to understand the causes of structure and function. *Annual Review of Biophysics*, 46:247–269, 2017.

[165] Louis J Billera, Susan P Holmes, and Karen Vogtmann. Geometry of the space of phylogenetic trees. *Advances in Applied Mathematics*, 27(4):733–767, 2001.

[166] Ines Chami, Albert Gu, Vaggos Chatziafratis, and Christopher Ré. From trees to continuous embeddings and back: Hyperbolic hierarchical clustering. *arXiv preprint arXiv:2010.00402*, 2020.

[167] Hirotaka Matsumoto, Takahiro Mimori, and Tsukasa Fukunaga. Novel metric for hyperbolic phylogenetic tree embeddings. *Biology Methods and Protocols*, 6(1):bpab006, 2021.

[168] Maximilian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6341–6350. Curran Associates, Inc., 2017.

[169] James W Cannon, William J Floyd, Richard Kenyon, Walter R Parry, et al. Hyperbolic geometry. *Flavors of geometry*, 31(59-115):2, 1997.

[170] Yoshihiro Nagano, Shoichiro Yamaguchi, Yasuhiro Fujita, and Masanori Koyama. A Wrapped Normal Distribution on Hyperbolic Space for Gradient-Based Learning.

[171] Nick Terry and Youngjun Choe. Splitting gaussian processes for computationally-efficient regression. *Plos one*, 16(8):e0256470, 2021.

[172] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

# 18 Appendix

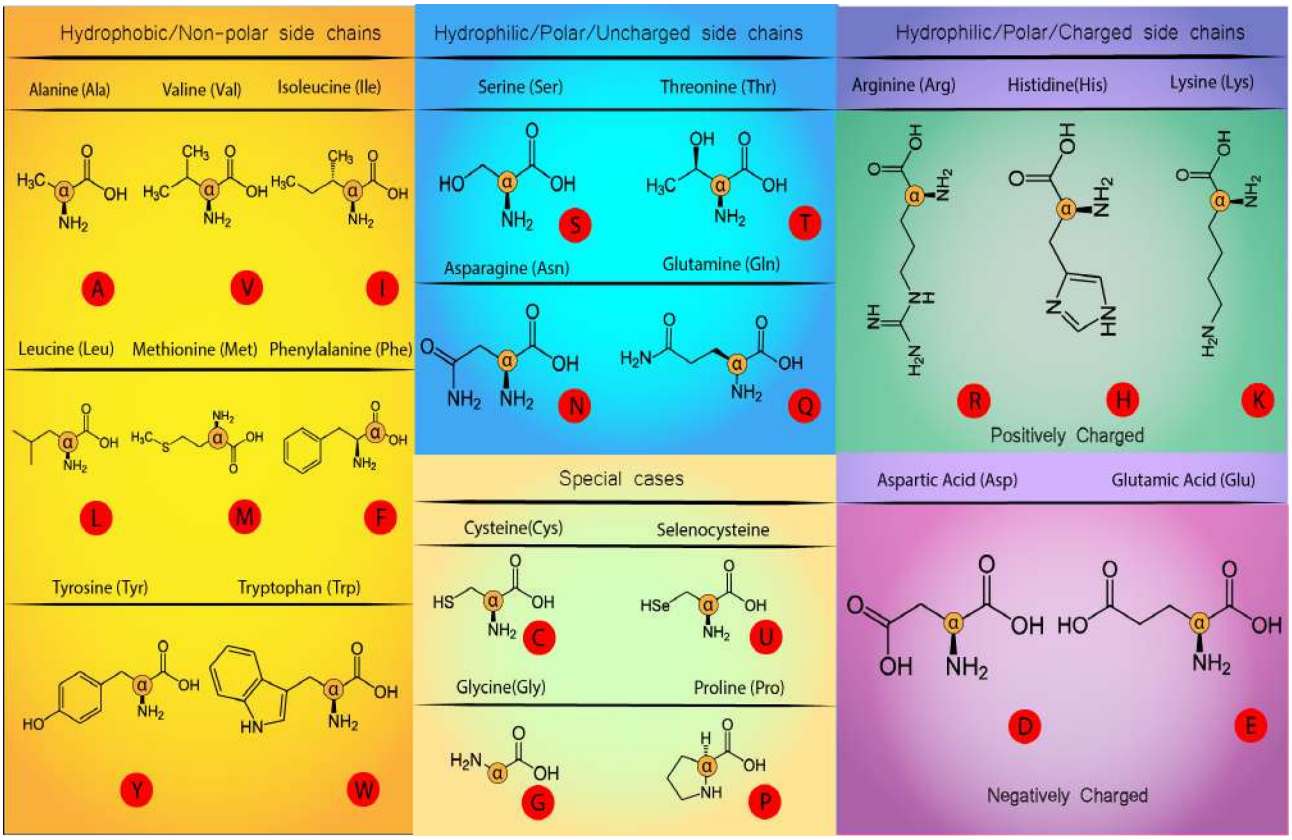



**Figure 26:** Overview of the most relevant existing aminoacids

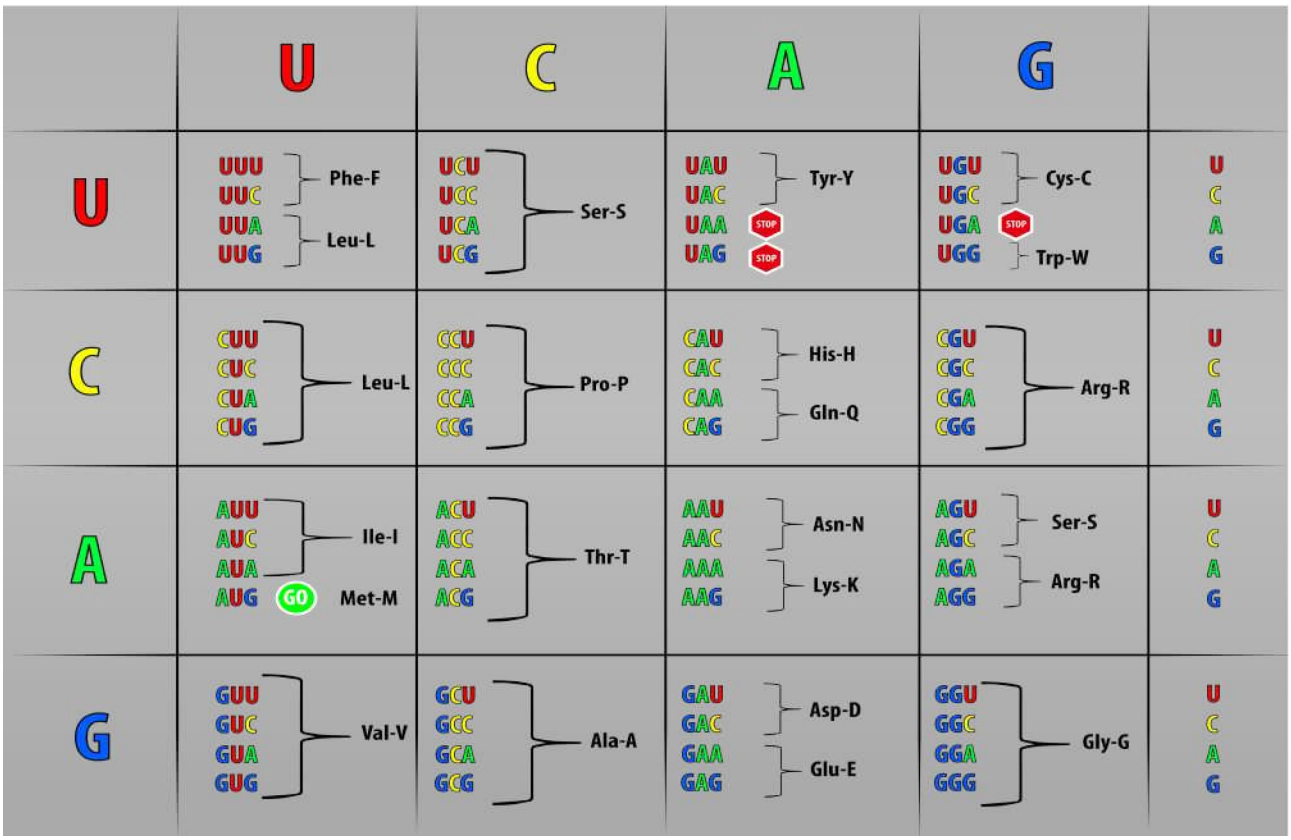



**Figure 27:** Codon-Amino acid equivalence table

# 19 Glossary

**Activation functions** Functions utilized the machine learning field perform an input transformation and are characterized by been easily differentiable.

- Linear activation functions: Functions that separate the data points using a linear hyper plane.
- Non-linear activation functions: Functions that separate the data points using a complex manifold. The also benefit from re-defining their inputs into a specific range of values. For example, sigmoid, ReLU, softplus etc.

. 37, 132

**Acyclic graph** Graphical model whose nodes never form a closed loop. 132

**AE** Autoencoder. 132

**CAVI** Coordinate Ascent Variational Inference. 132

**Closed-form expression** A closed-form solution or expression is a formula that can be expressed as a finite number of operations. Closed-form expressions offer closed-form solutions, which are regarded as exact solutions. 40, 132

**Conditional probability** Probability of a random variable $X$ occurring under some condition or random variable $Y$, $P(X \mid Y = y)$. The sum of the all the possible conditional probabilities, depending on the value of the conditional variable $Y$, adds up to 1. Meaning that,

$$P(X \mid Y) = \frac{P(X, Y)}{P(Y)} \tag{69}$$

- Discrete distributions : The conditional probability is a Probability Mass Function (PMF)

$$\sum_{x} P(X \mid Y) = \frac{\sum_{x} P(X, Y)}{P(Y)} = 1 \tag{70}$$

- Continuous : The conditional probability is a Probability Density Function (PDF)

$$\int_{-\infty}^{\infty} P(X \mid Y)\, dx = 1 \tag{71}$$

. 17, 132

**Cumulative distribution function (CDF)** Probability that a number would take a value less or equal to $t$.

$$F_X(t) = P(X \leq t)$$

The CDF of a continuous random variable gives the PDF from $-\infty$ to x.

$$P([-\infty < X < t] = \int_x^{-\infty} f_t(X)dt$$

. 132

**Cyclic graph** Graphical model whose nodes formed a closed loop. The number of vertices (nodes) equals the number of edges. Every node has 2 edges.. 132

**Derivative** Computes the change in slope or curvature of a target function (that maps 1D number to 1D number) with respect to the change in the inputs to the function. It can also be seen as,
• The rate of differentials that expresses the change of a function with respect to the change of a variable.
• The ratio of the differential of a function ($dy$) by the differential of a variable ($dx$). $f'(x) = \frac{dy}{dx}$.
• The limit of the function when the ratio of change of the variable ($\triangle x$) tends to 0.

$$f'(x) = lim_{\triangle x \to 0} \frac{f(x + \triangle x) - f(x)}{\triangle x} \tag{72}$$

. 132

**Differential** Rate of change of a variable, $dx$ or $\triangle x$ . 132

**ELBO** Evidence Lower Bound. 132

**Euler-Lagrange equations** Defines an equation for motion (energy) that can be interpreted for any type of coordinate space (cartesian, polar ...). See Section 9.2 . 31, 132

**Expectation-Maximization** Expectation Maximization (EM) is an unsupervised learning technique that iteratively calculates an expectation (conditional distribution or likelihood of the data under the current model parameters) and maximizes it (updates the parameters of the model to fit the data) [100] . 24, 132

**Expected value** The expected value of a random variable $\mathbb{E}[X]$ is the weighted average of a large number of independent realizations of $X$. It can also be expressed as the mean of an infinite population or *the mean or location of our probability distribution*. If the realizations are equiprobable (uniform distribution) then, the weights are equal across all realizations $p(x_i) = 1$ and the expected value is the arithmetic mean. If the realizations are not equiprobable the weights are proportional to the frequency of the random variable $p(X = x_i) = p(x_i)$. The expected value of a sampled variable $x$ from the distribution $p$ is equal to:

- Discrete probability case: $\mathbb{E}_{x \sim p} = \sum_i^N x_i p(x_i)$

- Continuous probability case : $\mathbb{E}_{x\sim p} = \int_{-\infty}^{\infty} x_i p(x_i) dx$

. 25, 132

**Features** In Machine Learning, a data feature consists of a vector or matrix of values that characterize the data. For example, we can describe a book with the descriptor vector of values $\hat{book} = (\text{rectangular} : 0.8, \text{paper} : 0.9, \text{size} : 10...)$ . 44, 132

**Gradient** Vector of first order partial derivatives of a target function/scalar field (that maps a n-dimensions vector to 1D number) with respect to its input variables [172]. The *euclidean gradient* points in the direction of the steepest ascent in Euclidean space, whilst the *natural gradient* used in Stochastic Variational Inference points in the direction of the steepest ascent in Riemannian space [101]. . 28, 38, 132

**HMC** Hamiltonian Monte Carlo. 132

**HMM** Hidden Markov Model. 132

**Integral** Approximation the Area Under the Curve (AUC) of function $f(x)$ between 0 and $x_m$. When $N$, the number of histogram bins under the curve, goes to $\infty$ , or what is the same, $\triangle x$ goes to 0.

$$\text{AUC} = \sum_{i}^{N} f(i)\triangle x = \int_{0}^{x_m} f(x).dx$$

We can also see the integral as the reverse of the differential. If we want to recover the original functional form $f(x)$ after a derivative, we can integrate

$$\frac{df(x)}{dx} = f'(x) \tag{73}$$

$$df(x) = f'(x)\,dx \tag{74}$$

$$f(x) = \int f'(x)\,dx \qquad \text{we can un-differentiate both sides} \tag{75}$$

. 23, 132

**Joint probability** Statistical measure that is used to calculate the probability of two events occurring simultaneously, in a bivariate case, when the variable $X$ takes realization $x$ and variable $Y$ realization $y$. The joint distribution is characterized by being non negative

$$P(X = x, Y = y) = P_{x,y}(X, Y) \geq 0 \tag{76}$$

and adds up to 1 when summing over all the possible values of the variables,

- Discrete variables case:

$$\sum_{x}\sum_{y} P(X, Y) = 1 \tag{77}$$

- Continuous variables case:

$$\int \int P(X,Y)\,dx\,dy = 1 \tag{78}$$

. 17, 19, 132

**KL** Kullback-Leibler. 32, 33, 132

**Marginal probability** In the bivariate case, unconditional probability of a random variable $X$ extracted from a joint distribution $p(X,Y)$ of 2 variables ($X$ and $Y$) where we consider that the events $X$ and $Y$ are independent. It can also be thought as the probability of an event irrespective of the outcome of another variable, both events occur simultaneously but independently. The marginal probability is calculated via the estimation of the expected value of a variable, which can be expressed as the collection of values of a variable $X$ as the variable $Y$ changes.
In the discrete case, we can calculate the *marginal probability mass function* by summing over all possible values of $Y$,

$$P_x(X) = \mathbb{E}[X] = \sum_{i=0}^{Y} P(X, Y = y_i) = \sum_{i=0}^{Y} P(X,Y) \tag{79}$$

whereas in the case of continous variables, we estimate the *marginal probability density function* via integration,

$$P_x(X) = \mathbb{E}[X] == \int_{-\infty}^{\infty} P(X,Y)\,dy \tag{80}$$

we can also restrict the range of values of the random variable $X$ to lie between the interval $[a,b]$,

$$P_x(X \in [a,b]]) = \mathbb{E}[X] == \int_{a}^{b} \int_{-\infty}^{\infty} P(X,Y)\,dy\,dx \tag{81}$$

The naming originates from marginalizing the probability values in a *joint probability table*, see Table 3, where the marginal probabilities will be the *marginals* or basically the rows/columns total sums (in the case of discrete variables).

**Table 3:** *Joint* and *marginal* probability table

| $X/Y$ | Brown | Green | Blue | $P_y(Y)$ |
|---|---|---|---|---|
| Car | 0.05 | 0.05 | 0.1 | 0.2 |
| Eye color | 0.3 | 0.1 | 0.1 | 0.5 |
| Pencil color | 0 | 0.1 | 0.2 | 0.3 |
| $P_x(X)$ | 0.35 | 0.25 | 0.4 | 1 |

. 17, 19, 23, 132

**MC** Markov Chain. 132

**MCMC** Markov Chain Monte Carlo. 132

**NN** Neural Network. 132

**Optimization** Identification of the value of the argument x that minimizes or maximizes a function f(x). 132

**Probability mass function (PMF), probability distribution function, distribution** Function that defines the probability of a discrete random variable being exactly equal to some value $t$. Describes the likelihood of obtaining a particular value when we sample (generate random variables) from it. It assigns positive probability to at most countably different values in the function's range.

$$P(X = t) = p_x(t) \sum_t p(t) = 1$$

. 132

**Probability distribution** Generalization term for probability density function (PDF) and probability mass function (PMF). 132

**Probability density function (PDF), density function, density** Function that defines the probability of a continuous random variable falling within a particular range of values. This probability is given by the integral of this variable's PDF over that range of values. It is obtained by calculating the area under the density function but above the x-axis and between the lowest and greatest values of the range. The probability density function is non-negative everywhere, and its integral over the entire value space is equal to 1. The function that complies with those characteristics is a *Lebesgue* function. The probability that a random variable is exactly equal to a value is technically 0 due to the infinitesimal nature of the integral.

$$p(X \in A) = \int_A p(x)\, dx \tag{82}$$

$$p([a < X < b] = \int_b^a f_x(X)\, dx$$

. 25, 26, 132

**Random element** Deterministic function defined on an event space that, maps the outcome of a random experiment (type of experiment that generates different outcomes under identical experimental conditions) to either:

- a) a real number (“random variable") from .
- b) a vector of numbers (“random vector").
- c) a set of numbers indexed by a continous space time (“stochastic process").
- d) a set of numbers indexed by 2 or more discrete spatial dimensions (“random field").
- e) trees or graphs.

. 132

**Random Variable** Result of mapping of a random element from a random experiment to $\sigma$-algebra or space of real numbers . Random Variables induce a probability distributions on their range. 19, 132

**Realization** Instance (i.e sample) of a random variable. 132

**Representation learning** Also referred as feature learning, comprises a series of automatized techniques used for learning the most relevant features from the raw data. This is necessary in order reduce the computational cost of working with the full raw data . 44, 132

**Reversibility** A *transition probability distribution* is reversible if the probability of transition from state $X_{n-1}$ to state $X_n$ is the same one as the one from $X_n$ to $X_{n-1}$ . 28, 132

**SVI** Stochastic Variational Inference. 132

**Unbiased estimator** The estimate of our parameter is as close as possible to the true value of that parameter, otherwise is biased and deviates from the true value. Biased estimators for example occur when our data sample is not representative of the population, it does not accurately represent the average in the sample. A simple case is when we attempt to estimate the average amount of sports practiced and we only consider the people that attend the gym from our population . 41, 132

**VAE** Variational Autoencoder. 132

**Variational Calculus** Calculus of the variation or change in a *Functional*'s value due to a small change in the input of the *Functional*, in line with the differential concept. It deals for example with derivatives, functional derivatives or integrals . 31, 132

**VI** Variational Inference. 132