LEARNING FITNESS LANDSCAPES FOR PROTEIN DESIGN

A PREPRINT

Trenton Bricken[*] Duke University
Nathan Rollins Harvard Medical School
Nikki Thadani Harvard Medical School
Debora Marks Harvard Medical School

January 2, 2022

ABSTRACT

Recent advances allow statistical models to predict protein function from sequence data alone. But even with an excellent model for protein function, it is still difficult to identify those sequences that have the highest predicted functionality and which also inhabit distant regions in sequence space. This paper proposes the use of variational inference and invertible neural networks to identify sequences that maximize predicted function and are very distinct. The method works for any model that predicts protein function. The sequences produced from the method may be directly assayed or used to seed directed evolution experiments in distinct regions of sequence space. More optimal protein design holds the promise to benefit almost every aspect of our lives not limited to the creation of better drugs, enzymes and crops [1] [2] [3] [4].

***K*eywords** Protein Design · Variational Inference · Invertible Networks · Normalizing Flows · Deep Learning

1 Background and Motivation

A protein function predictor (PFP) can be thought of as a black box model with a protein sequence as input and score for a particular desired function or set of functions as the output. There have been a number of recent models that show capabilities of not only predicting protein function with impressive accuracy but also are able to generalize reasonably well beyond their training data across protein space [3] [4] [5] [6] . However, provided with a successful PFP, it remains non-trivial to use this model to generate a set of sequences that not only have a high PFP score but also are diverse. This emphasis on diversity is important for a few reasons pertaining to the fact that PFPs are and will likely remain imperfect:

(i) Because of uncertainty around which protein sequence will actually be the most functional, we want to try those which are unique and unlikely to fail in the same ways;

(ii) To produce truly optimized proteins the computational protein design field still must rely upon directed evolution methods [7] [8]. Imagining a protein fitness landscape, one can think of directed evolution as performing a stochastic hill climbing search towards the nearest local maxima, this makes using a diverse set of starter sequences an advantageous way to increase the likelihood of discovering the global maximum.

Due to the very high dimensionality of the protein sequence space (an average protein is $\sim$300 amino acids long making it 20^{300} possible sequences [11]) it is infeasible to perform any provably optimal search for example, through using brute force or dead end elimination. As a result, we must rely upon ways to efficiently statistically approximate the fitness landscape. The naive approach to do so is Markov Chain Monte Carlo (MCMC) which has the guarentee that it will asymptotically converge to the true distribution [12]. However, for highly non-linear spaces, MCMC can struggle to mix well between different modes and take a very long time to reach stationarity [13].

Variational inference (VI) is another technique to approximate a distribution which has been successfully applied through the use of artificial neural networks in the field "Deep Learning" to a number of domains, particularly with

[*]correspondence to trenton.bricken@duke.edu

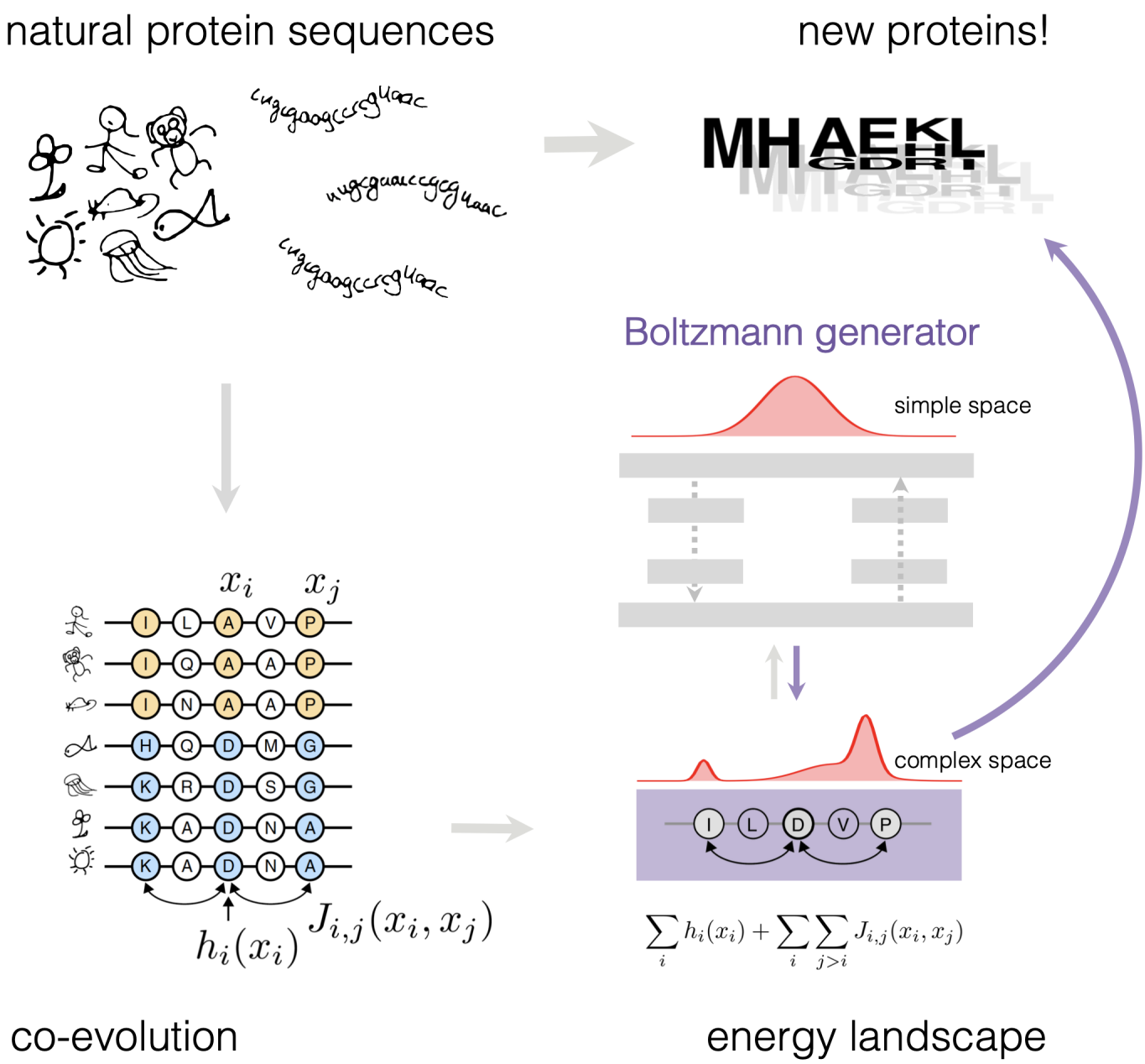

Figure 1: A graphical abstract of our methodology. In this example we use natural sequence variation to learn a PFP [9] [10].

Variational Auto-Encoders [14] [15]. The success of VI lies in the fact that it converts the problem of approximating the normalizing constant from one that MCMC solves with sampling, to one that is solved via optimization, which lends itself to speedups including stochastic optimization and distributed computing [14]. VI achieves its approximation of the normalizing constant by learning how to map a latent probability distribution such as a multivariate Gaussian, onto a more complex distribution over data. Because the latent probability distribution is only being transformed, it remains normalized, removing the need to explicitly approximate the normalizing constant.

A useful abstract analogy to the approach of VI is to imagine the distribution you are trying to learn is a crumbled up ball of paper and it is being approximated by putting a sheet on top and pressing it downwards so that it covers all of the

contours of the paper ball as tightly as possible. More formally, VI minimizes the the Kullback-Leibler divergence (KLD) between the intractable distribution being approximated $p(x)$ and the simple latent distribution chosen $q(x)$.

$$D_{KL}[q(x)||p(x)] = \int_x q(x) \ln \frac{q(x)}{p(x)} dx$$

KLD is an asymmetric distance metric ($D_{KL}[q(x)||p(x)] \neq D_{KL}[p(x)||q(x)]$). KLD is always positive and equals 0 when $q(x) = p(x)$.

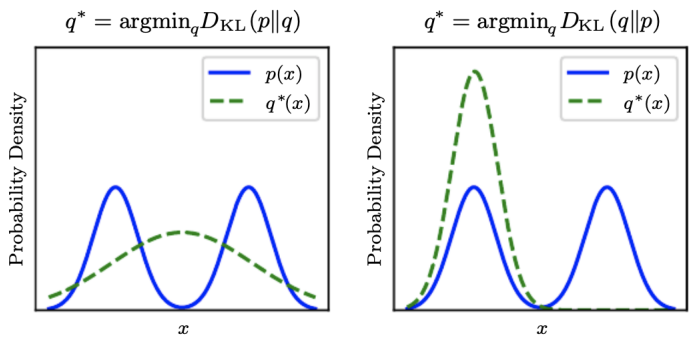

Figure 2: This figure shows how Kullback-Leibler Divergence is asymmetric and this changes the optimal solution. On the left, the latent space q tries to put its high probability regions everywhere that p has high probability, as a result resorting to blurring together the modes. On the right, the latent space q will try to avoid low probability regions of p, as a result resorting to sticking to only one of the modes (the right mode could have been chosen as easily as the left one has been). This example is taken from Goodfellow et al.'s Deep Learning textbook where it is Figure 3.6 [16].

This work is inspired by a novel formulation of VI which uses invertible neural networks to achieve a tighter bound on the PFP parameterized fitness landscape by allowing for the KLD to be minimized in two directions. This approach was pioneered by Noe et al. in their Boltzmann Generator paper where they used it to approximate the energy landscape of molecular dynamics simulations and accelerated the run time by up to a factor of one hundred [13]. Here, by approximating well the PFP parameterized fitness landscape we can use our model to identify diverse, locally optimal protein sequences to use for later experimentation and hopefully the discovery of highly performant proteins.

2 Related Work

Beyond MCMC there have been a few additional approaches developed recently in the literature. Most relevant is the Adaptive Sampling research of Listgarten et al. which uses a combination of VI and annealing of the PFP to move away from known protein sequences towards discovering more functional ones [17] [18].

Formally, Adaptive Sampling seeks to learn a generative model of protein sequences that can be conditioned on the desired PFP functionality $p(x|S, \theta)$ where S is the set of desirable functionality scores, θ are the parameters for the generative model and $q(x|\phi)$ is a latent distribution to be learnt. This is achieved with the VI equation:

$$D_{KL}[p(x|S, \theta)||q(x|\phi)]$$

In order for training to be successful and variational inference to work here, a model of the data distribution $p(x|\theta)$ is necessary along with annealing of the score S that the model is conditioned on during training.

The use of VI provides a formal theoretical framework that the other methods to be mentioned lack. In addition, the use of the known protein sequences as the prior distribution ensures the method does not drift too far away into unknown regions of space where the PFP may fail to generalize. However, this model is only able to minimize VI in a single direction, which in reference to Figure 2 is the approach that encourages straddling across every mode. This straddling combined with the reliance on training data for the prior distribution prevents the model from capturing as much of the fitness landscape conditioned on the PFP as it otherwise would.

Another approach to the problem of finding the highest scoring and diverse sequences from the PFP is Deep Exploratory Networks [19]. These are generative neural networks trained to generate high scoring sequences. In order to avoid mode collapse where the model finds a local optima and generates sequences from it exclusively to maximize the score, there is a diversity penalty in the loss function. The method, like the others, relies upon the PFP being differentiable which most are not because they expect discrete protein sequences as input. Meanwhile, the generative network outputs a probability distribution over amino acids at each position in the sequence. To get around this issue the method uses two approaches, firstly it incorporates an entropy penalty into the loss function to favour the production of discrete sequences. However, this results in vanishing gradients. Secondly they use the straight through gradient estimator and sample a number of discrete sequences from the probability distribution [20]. These techniques are used successfully to show that having a non-differentiable PFP can be managed. The issue with Deep Exploratory Networks is that there is no statistical theory behind the sequences that are discovered and an arbitrary diversity penalty is required.

A final approach recently published frames the problem in the context of model based reinforcement learning [21]. In this setting, an agent is trained to propose sequences that are "tested" in the next experimental round. It is then re-trained on the results and generates its next set of predictions. This work assumes the underlying function being optimized for is unknown and trains a suite of different statistical models to try and approximate it. This is slightly different from our setting where we already have a PFP model that we believe is a good approximation of the ground truth protein functionality. As with Deep Exploratory Networks, the loss function must be trained with an explicit diversity metric. As a result, this method becomes almost identical to Deep Exploratory Networks except for it uses Proximal Policy Optimization, a more sample efficient version of the REINFORCE algorithm (outlined later) as its method of handling the discrete sequence issue [22] [23]. It is noteworthy that on the benchmarks used, their approach is able to significantly outperform Listgarten et al.'s Adaptive Sampling approach. Deep Exploratory Networks was published at almost the same time meaning comparisons between the two have not yet been performed.

3 Methods

Given a PFP that we treat as a black box model, we are trying to learn a protein sequence generator that maximizes the PFPs score while also finding as many local optima as possible. We use the Bayesian framework of VI to try and approximate the intractable PFP parameterized probability distribution with invertible neural networks enabling a tighter VI than otherwise obtainable.

Applying invertible neural networks to discrete sequences presents a number of challenges and novel computational solutions.

The Model

The novel form of variational inference used in this paper, thanks to the utilization of invertible networks, is able to approximate a tighter bound by minimizing Kullback-Leibler divergence (KLD) in two directions as outlined by this graphic:

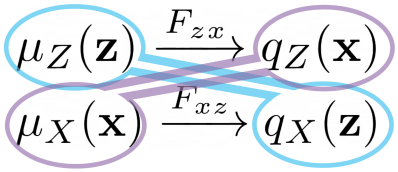

Figure 3: Kullback-Leibler Divergence is minimized between the probability distributions encompassed by the blue and purple outlines. The top row shows the latent space $\mu_Z(\mathbf{z})$ which is mapped into the protein space $q_Z(\mathbf{x})$. The bottom row shows the intractable distribution of protein sequences according to the PFP chosen $\mu_X(\mathbf{x})$ and its mapping into the latent variables $q_X(\mathbf{z})$.

We use the notation $\mu()$ to denote an "original" non-transformed probability distribution and $q()$ for a transformed distribution with a subscript for the specific random variable which the distribution originated from. $\mu_Z(\mathbf{z})$ is the latent space distribution which is transformed $\mu_Z(\mathbf{z}) \xrightarrow{F_{zx}} q_Z(\mathbf{x})$ by the invertible network into sequence space. Similarly, $\mu_X(\mathbf{x})$ is the intractable probability distribution parameterized by the PFP which is transformed $\mu_X(\mathbf{x}) \xrightarrow{F_{xz}} q_X(\mathbf{z})$ into latent space.

By invertibility, $F_{zx} = F_{xz}^{-1}$ and the change of variables formulae:

$$q_X(\mathbf{z}) = \mu_X(F_{zx}(\mathbf{z};\theta))|\det J_{xz}|$$
$$q_Z(\mathbf{x}) = \mu_Z(F_{xz}(\mathbf{x};\theta))|\det J_{zx}|$$

Where $J_{xz} = \dfrac{d\mathbf{x}}{d\mathbf{z}}$ is the Jacobian.

We define our PFP distribution by the Boltzmann distribution:

$$\mu_X(\mathbf{x}) = \frac{e^{-T\psi(\mathbf{x})}}{Z}$$

Where T is a temperature constant, $\psi(\mathbf{x})$ is the PFP function and Z is the normalizing constant.

The first KLD shown in blue in Figure 3 can be reduced in the following way to a tractable loss function that can be used to produce gradients that can train the invertible neural network.

$$D_{KL}[\mu_Z(\mathbf{z})||q_X(\mathbf{z})] = \int_{\mathbf{z}} \mu_Z(\mathbf{z}) \ln(\frac{\mu_Z(\mathbf{z})}{q_X(\mathbf{z})})d\mathbf{z}$$

$$= \int_{\mathbf{z}} \mu_Z(\mathbf{z}) \ln \mu_Z(\mathbf{z})d\mathbf{z} - \int_{\mathbf{z}} \mu_Z(\mathbf{z}) \ln q_X(\mathbf{z})d\mathbf{z}$$

$$= -H(\mu_Z(\mathbf{z})) + \int_{\mathbf{z}} \mu_Z(\mathbf{z})[-\ln \mu_X(F_{zx}(\mathbf{z};\theta))) - \ln|\det J_{xz}|]d\mathbf{z}$$

$$= -H(\mu_Z(\mathbf{z})) + \int_{\mathbf{z}} \mu_Z(\mathbf{z})[T\psi(F_{zx}(\mathbf{z};\theta)) + \ln Z - \ln|\det J_{xz}|]d\mathbf{z}$$

$$= -H(\mu_Z(\mathbf{z})) + \ln Z + \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}[T\psi(F_{zx}(\mathbf{z};\theta)) - \ln|\det J_{xz}|]$$

Applying the gradient ∇_θ to learn the parameters of the invertible network we are left with a lower bound:

$$= \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}[\nabla_\theta\psi(F_{zx}(\mathbf{z};\theta)) - \nabla_\theta \ln|\det J_{xz}|]$$

The other direction in purple is directly equivalent to performing likelihood maximization of the data distribution:

$$D_{KL}[\mu_X(\mathbf{x})||q_Z(\mathbf{x})] = \int_{\mathbf{x}} \mu_X(\mathbf{x}) \ln(\frac{\mu_X(\mathbf{x})}{q_Z(\mathbf{x})})d\mathbf{x}$$

$$= \int_{\mathbf{x}} \mu_X(\mathbf{x}) \ln \mu_X(\mathbf{x})d\mathbf{x} - \int_{\mathbf{x}} \mu_X(\mathbf{x}) \ln q_Z(\mathbf{x})d\mathbf{x}$$

$$= -H(\mu_X(\mathbf{x})) + \int_{\mathbf{x}} \mu_X(\mathbf{x})[-\ln \mu_Z(F_{xz}(\mathbf{x};\theta)) - \ln|\det J_{zx}|]d\mathbf{x}$$

$$= -H(\mu_X(\mathbf{x})) + \int_{\mathbf{x}} \mu_X(\mathbf{x})[\frac{1}{\sigma^2}||F_{zx}(\mathbf{x};\theta)||^2 + \ln Z - \ln|\det J_{zx}|]d\mathbf{x}$$

$$= -H(\mu_X(\mathbf{x})) + \ln Z + \mathbb{E}_{\mathbf{x}\sim\mu_X(\mathbf{x})}[\frac{1}{\sigma^2}||F_{zx}(\mathbf{x};\theta)||^2 - \ln|\det J_{zx}|]$$

$$= \mathbb{E}_{\mathbf{x}\sim\mu_X(\mathbf{x})}[\frac{1}{\sigma^2}\nabla_\theta||F_{zx}(\mathbf{x};\theta)||^2 - \nabla_\theta \ln|\det J_{zx}|]$$

Recalling Figure 2, the ability to minimize KLD in both directions here makes this VI a tighter bound that tries to simultaneously cover all of the modes of the data and cover them tightly. Minimizing $D_{KL}[\mu_Z(\mathbf{z})||q_X(\mathbf{z})]$ is both what makes this model unique and what allows for exploration across the space to find diverse local optima.

The model currently uses independent Gaussians as its prior in the latent space $\mu_Z(\mathbf{z})$, therefore making the mean field assumption $q(\mathbf{x}) = \prod_i q_i(x_i)$ but exploring more complex prior distributions is a compelling area for future research.

Discrete Sequences

One challenge of using this approach for protein sequences lies in their discrete nature. This not only makes it difficult for the neural network to learn a good representation of the sequences but also to incorporate any PFP.

We have experimented with dequantizing the discrete sequences by adding noise to them and also representing the sequences by a number of their continuous features such as charge, mass and hydrophobicity. An advantage of the continous representation approach is that the neural network can learn to generate values for each property that is absolute in its quantity. When the network is tasked with outputting a continuous representation of sequences, the values assigned to each amino acid are relative to each other because of the softmax function $\frac{\exp(x)}{\sum_x \exp(x)}$ necessary to convert them into a probability distribution. An additional consideration is that in order for the network to be invertible, the size of the latent space must be the same as the size of the protein. Without representing the amino acids in the sequence by their properties, each must be encoded in a "onehot" format where it is a length twenty vector of zeros and a one in a single position mapped in advance to this amino acid. This means for an average 300 length protein the latent space must be $300 * 20 = 6000$ dimensions making learning more difficult. For the amino acid property based representation it is only $300 * 4 = 1200$ in the case where four properties are used.

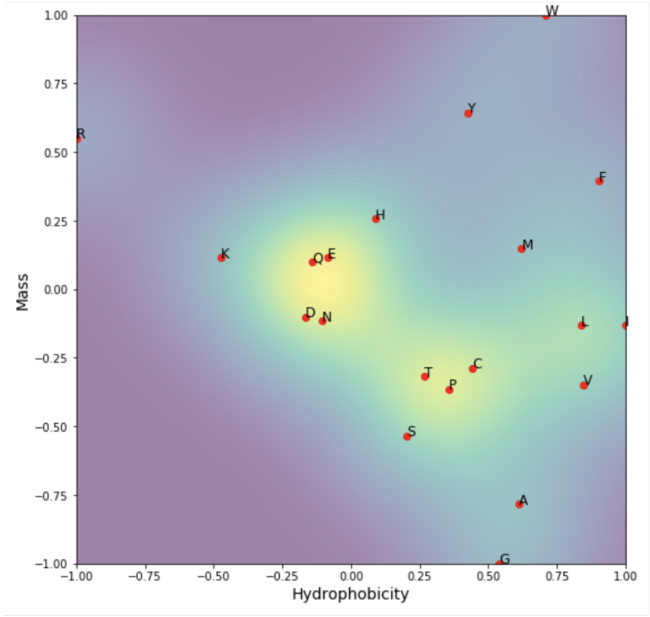

Figure 4: An example representation of all twenty amino acids in a continous space between their mass and hydrophobicity. Each amino acid is represented probabilistical by a two dimensional Gaussian with unit variance and mean centered at their properties. The colors represent the normalized probabilities of sampling from a particular region of amino acid space.

Aside from representing the sequences for the model, there is a difficulty in mapping the model outputs back to being the discrete sequences that most PFPs expect. Either training the model on the sequences or their continuous amino acid properties results in a probability distribution over over all twenty amino acids at each position in the sequence. Moreover, the most likely sequence alone cannot be taken because this operation is non-differentiable and would prevent training the neural network via backpropagation. However, this challenge is surmountable with ways of computing the gradient through discrete PFPs as is done in the Deep Exploratory Network and Reinforcement learning approaches [19] [21]. Outlining these approaches now in more detail:

(i) The entropy penalty uses the equation:

$$H(\mathbf{x}) = -\sum_{x_i \in \mathbf{x}} \sum_{a \in x_i} p(a) \ln p(a)$$

Where a is an amino acid from a particular position in the sequence x_i. This penalizes the amino acid probability distributions that are further away from being a "onehot" discrete sequence. One disadvantage is that the contribution of this penalty to the overall loss function must be weighted in proportion to the KLD losses.

(ii) REINFORCE is an on-policy reinforcement learning algorithm [22] that enables the expectation of gradients to be calculated from non-differentiable functions:

Taking the component of the loss function that corresponds to $D_{KL}[\mu_Z(\mathbf{z})||q_X(\mathbf{z})]$:

$$\mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}[\nabla_\theta \psi(F_{zx}(\mathbf{z};\theta))) - \nabla_\theta \ln|\det J_{xz}|]$$

And focusing on a non-differentiable PFP function $\psi()$:

$$\mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}[\nabla_\theta \psi(F_{zx}(\mathbf{z};\theta))] = \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}[\nabla_\theta \mathbb{E}_{\mathbf{s}\sim P(F_{zx}(\mathbf{z};\theta))}[\psi(s)]]$$

$$= \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}[\int_s \nabla_\theta p(s)\psi(s)ds]$$

$$= \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}[\int_s p(s)\nabla_\theta \ln p(s)\psi(s)ds]$$

$$= \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}[\mathbb{E}_{\mathbf{s}\sim P(F_{zx}(\mathbf{z};\theta))}[\nabla_\theta \ln p(\mathbf{s})\psi(s)]]$$

Where s here is a sample from the sequence probability distribution $\mathbf{x} = F_{zx}(\mathbf{z};\theta)$.

For the experiments run in this paper we did not require using either the entropy loss or REINFORCE because, while performing this research, we discovered that the Marks lab Evolutionary Couplings PFP, has a closed form solution to its expectation over a continuous sequence probability distribution allowing it to handle these inputs [9] [24] [10]. An outline of Evolutionary Couplings and formal description of its closed form expectation calculation are in the Appendix.

Invertible Networks

The use of invertible neural networks, also known in the literature as normalizing flows, while enabling a tighter VI bound, do introduce constraints on the expressivity of the neural network because of their restriction to invertible transformations.

As an example of an invertible transformation and how these can be used to map a simple distribution into a far more complex one, consider a vector $\mathbf{z}$ split in half as $\mathbf{z}_1$ and $\mathbf{z}_2$ with the following transformations applied [25]:

$$\mathbf{x}_1 = \mathbf{z}_1$$

$$\mathbf{x}_2 = (\mathbf{z}_2 * \sigma(\mathbf{z}_1)) + \mu(\mathbf{z}_1)$$

Where $\sigma()$ and $\mu()$ are arbitrarily complex and not necessarily invertible transformations of the data, in this case representing neural networks. To get back to where we started we simply perform:

$$\mathbf{z}_1 = \mathbf{x}_1$$

$$\mathbf{z}_2 = \frac{\mathbf{x}_2 - \mu(\mathbf{z}_1)}{\sigma(\mathbf{z}_1)}$$

Intuitively, this uses the one half of the split vector as a key to undo changes on the other half. We can learn complex transformations over time by applying the forward pass multiple times, each time alternating which half of the split vector is used as the key, with different $\sigma()$ and $\mu()$ functions learnt for each forward pass performed.

An advantage of this invertible function is that the Jacobian is lower triangular which makes its determinant only the product of the diagonal which is simply $\sigma(\mathbf{z_1})$.

$$\frac{d\mathbf{x}}{d\mathbf{z}} = \begin{bmatrix} I & 0 \\ \frac{d\mathbf{x_2}}{d\mathbf{z_1}} & \frac{d\mathbf{x_2}}{d\mathbf{z_2}} \end{bmatrix}$$

A number of more sophisticated invertible transformations have been invented including autoregressive transformations and monotonically increasing spline functions [26] [27] . Most recently, a discrete invertible network has been introduced which seems ideal for the discrete nature of the protein sequences [28]. A number of these we have attempted to implement to assess their ability to learn highly multimodal protein fitness landscapes, however, not always with success. The discrete invertible network module in particular resulted in us re-writing the codebase in PyTorch, as a result creating the first public, working implementation of the algorithm but some problems with it remain and correspondence with the original authors is ongoing[2].

The full list of invertible networks tried include: NICE, RealNVP, MAF, and Neural Splines [25] [29] [26] [27] [30]. The exact parameters used for each of these can be found in the Appendix.

Training

To benchmark performance of the model using Evolutionary Couplings we used a model trained to predict the evolutionary fitness of the e. Coli protein DYR. DYR is 155 amino acids long and has enough homologous protein sequences to build a good Evolutionary Couplings model to use as our PFP and predict protein fitness [9] [10] [24].

In order to provide sequences for the KLD training in the maximum likelihood direction $D_{KL}[\mu_X(\mathbf{x})||q_Z(\mathbf{x})]$ we ran an MCMC chain for a short period of time producing 10,000 samples to provide an initial approximation of this intractable distribution, as was done in the original Boltzmann Generator paper. Also from the paper, we first train the model using only this data driven direction for a number of epochs before training both of the KLD directions simultaneously. To discourage mode collapse we found the best results when the $D_{KL}[\mu_Z(\mathbf{z})||q_X(\mathbf{z})]$ direction had only half of the weight of $D_{KL}[\mu_X(\mathbf{x})||q_Z(\mathbf{x})]$.

When sampling from the model after it has been trained to produce protein sequences, we take the highest probability amino acid at each position from the amino acid probability distributions generated at each position.

4 Results

4.1 Toy Protein

To first assess model performance, we kept only the first six amino acids of the DYR protein. Six was chosen as the longest sequence that brute force search across all $20^6 = 64000000$ possible sequences could be run. In the context of this ground truth fitness landscape, we ran MCMC and our Boltzmann Generator using its onehot and amino acid feature representations to see how many local optima each was able to find. For this analysis we show only the results from one invertible network type, NICE, and for the amino acid feature representation, which performs better[3]. Because of the low dimensionality of the space both MCMC and the Boltzmann Generator were able to perform well but with some limitations of the Boltzmann Generator beginning to appear.

On a single CPU the MCMC was able to run 64 chains in parallel, for 15,000 steps and by keeping every 5th step, generate 192,000 samples in 4 seconds. The Boltzmann Generator required 15 minutes on a CPU and 4 minutes on a GPU using the most simple NICE architecture. Taking the same number of samples from the Boltzmann Generator as from MCMC, while the results are stochastic, we obtain statistically significant results at an alpha of 0.05 that for the highest scoring unique 10,000 sequences generated by each model, the Boltzmann Generator has slightly more diversity compared to MCMC. However, while capturing more diversity, the Boltzmann Generator can miss some of the local optima as is revealed by looking at the top 1,000 sequences from the brute force search. See these results outlined below.

Method	**Hamming Distance**, (95% CI)	**Accuracy**, (95% CI)
Boltmann Generator AA Features	0.698, (0.708-0.687)	0.930, (0.943-0.917)
MCMC	0.677, (0.686-0.668)	0.994, (0.994-0.994)
Natural Sequences	0.646	0.417

[2]The package is available at: https://github.com/TrentBrick/PyTorchDiscreteFlows

[3]As is discussed later we don't use the more powerful models because they prove to be very difficult to train.

Hamming distance is the mean of all pairwise sequences:

$$d(\mathbf{a}, \mathbf{b}) = \frac{\sum_i I(a_i, b_i)}{|\mathbf{a}|}$$

$$I(a_i, b_i) = \begin{cases} 1, & \text{if } a_i == b_i \\ 0, & \text{else} \end{cases}$$

Where $\mathbf{a}$ and $\mathbf{b}$ are both protein sequences of the same length.

We can visualize the top 10,000 sequences found by the Boltzmann and MCMC approaches in relation to the natural sequences using UMAP [31].

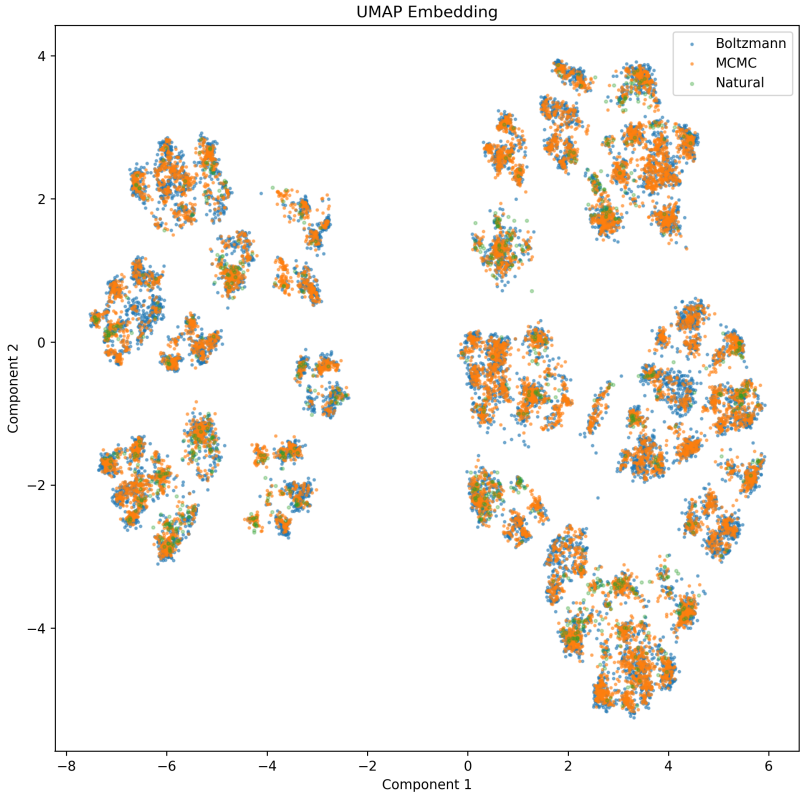

Figure 5: UMAP of the 10,000 highest scoring sequences across different generation approaches compared to the natural sequences. This graphic uses the amino acid feature Boltzmann approach. Note how the results are similar for this length six protein but there are places where the Boltzmann approach deviates into a slightly larger region.

In addition, Figure 6 shows how the Boltzmann Generator is able to learn a larger diversity of sequences across the space than MCMC but many of these are low scoring and not useful for the purposes of finding the local optima. In addition, the fact that the Boltzmann Generator energy distribution looks so normally distributed, combined with the UMAP figure where it deviates somewhat from the different sequence clusters, suggests that either the normal distribution prior used for VI or the amino acid feature representation may be constraining the number and size of the modes the model can approximate across the landscape.

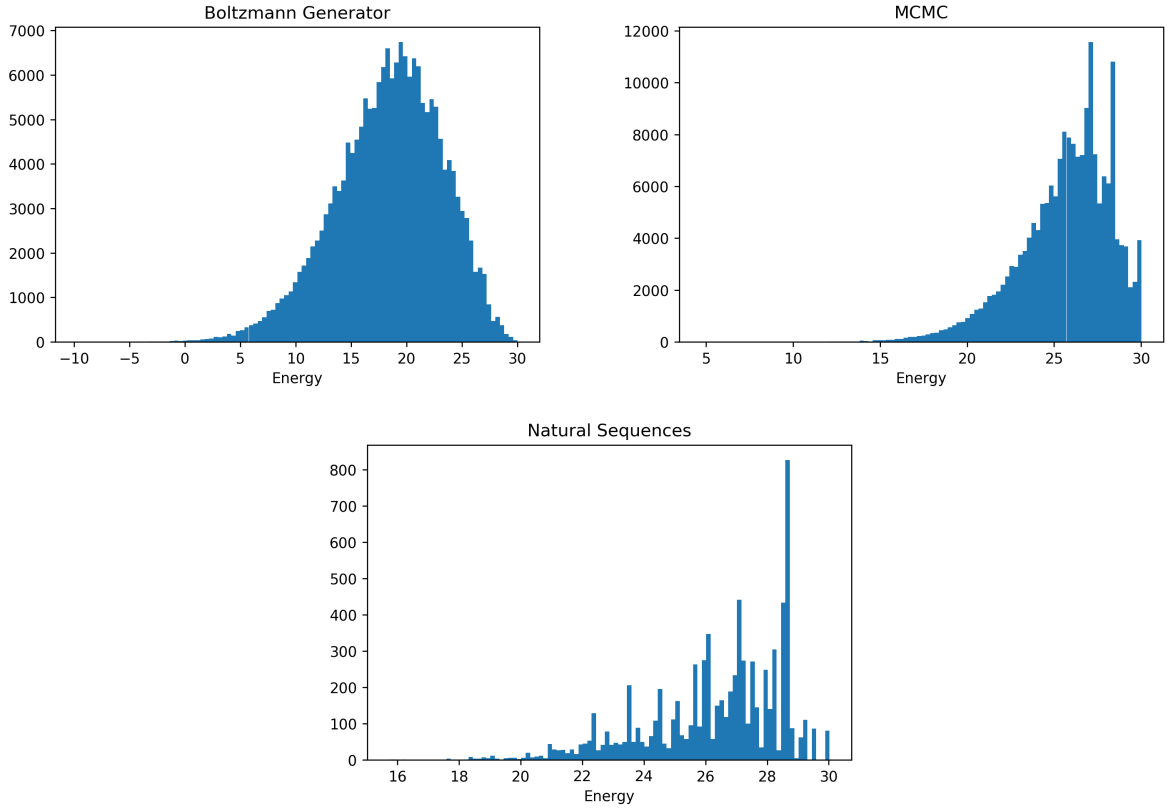

Figure 6: Histograms of the energies of sequences that each generation method captures in relation to the distribution of the natural sequences for the length 6 toy protein.

4.2 Full Length Protein

Getting reasonable results from the Boltzmann Generator for the length six protein, we scaled up our analysis to the full length 155 protein to see how the method would perform in a much higher dimensional space. Unfortunately, the model is unable to capture the space well. We hypothesize that this is due to limitations of the method introduced by either the mean field assumption on the latent space or the invertible neural networks failing to be sufficiently expressive for these highly multimodal fitness landscapes. Both of these potential issues are areas for future research.

MCMC run for 3,000 steps to produce 38,400 samples with random initial starting points also struggles to explore the space, however, using the 64 highest scoring natural sequences to initialize the chains run in parallel finds a number of high scoring solutions rapidly in $\sim$ 30 seconds on a single CPU. The Boltzmann Generator takes $\sim$10 minutes to run NICE on a GPU and fails to find these high scoring sequences for both the onehot and amino acid feature representation approaches. Using a more powerful invertible network like the state of the art Neural Splines [27] leads to highly unstable training that crashes[4].

Figure 7 shows the sequence energies generated by the different MCMC approaches and the amino acid feature representation Boltzmann Generator. The fact that the MCMC with a random initialization gets stuck between sequences at the 400 and 500 energy level as does the Boltzmann Generator suggests that these local optima are large attractors and difficult to escape from. Figures 8 and 9 further support this theory. In the case of the Boltzmann Generator, while with the length 6 protein it was able to capture a region of space larger than MCMC or the natural sequences, in this higher dimensional setting it struggles. This is true even for the more powerful Neural Splines which can be trained in a Maximum Likelihood $D_{KL}[\mu_X(\mathbf{x})||q_Z(\mathbf{x})]$ direction without breaking but fail to capture any more of the space than NICE.

[4]Not only are powerful invertible networks well documented to be difficult to train [29] but we hypothesize that the generative training of the model in the KLD direction $D_{KL}[\mu_Z(\mathbf{z})||q_X(\mathbf{z})]$ may generate very low probability sequences that produce gradients which break the model. This direction of training is not normally performed for invertible networks which tend to be used for density estimation and only perform the maximum likelihood equivalent direction $D_{KL}[\mu_X(\mathbf{x})||q_Z(\mathbf{x})]$.

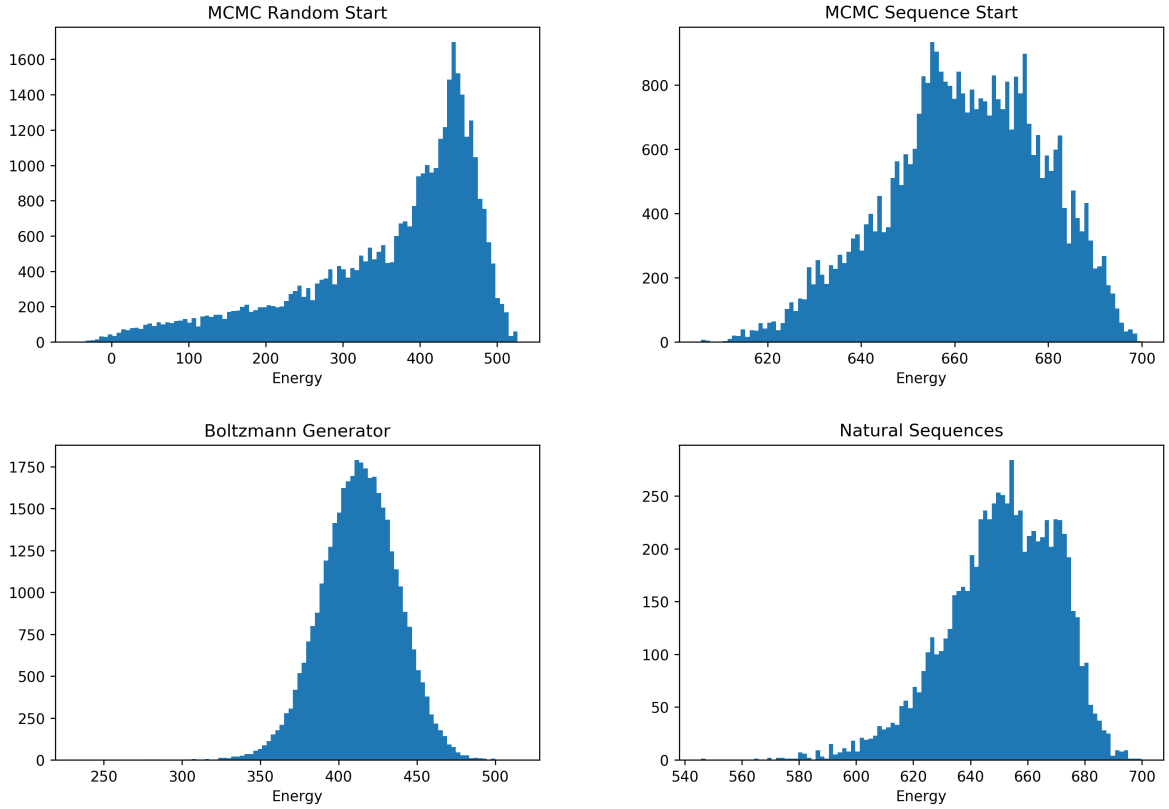

Figure 7: Histograms of the energies of sequences that each model captures for the full length DYR protein.

Considering the discrete representations of the sequences, for both the length 6 example and full length proteins, the Maximum Likelihood approach is able to capture a larger number of sequences than the amino acid feature representation approach which has less amino acid specificity. However, the model fails when the joint training in both VI directions occurs, seeing model collapse to a particular mode, at times the highest scoring mode and other times a lower scoring one as shown in Figure 10. These stark contrasts between Maximum Likelihood and joint training do not occur for the amino acid feature representation model.

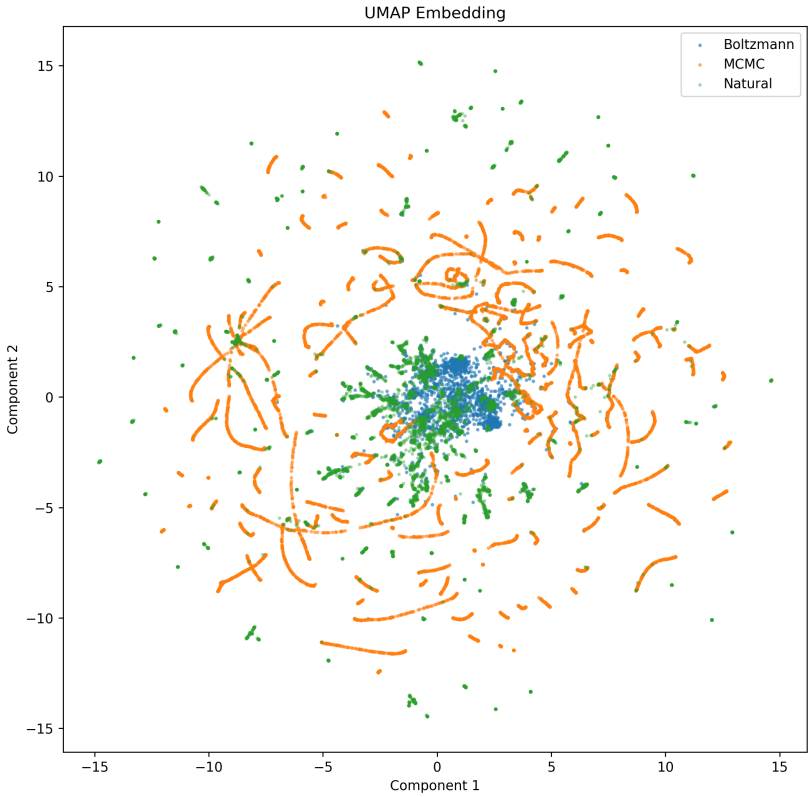

Figure 8: UMAP of the 10,000 highest scoring sequences for the full length DYR protein. This plot shows how the amino acid feature representation Boltzmann Generator approach captures regions of the largest cluster of sequences which happen to also be low scoring. The MCMC here has each chain initialized by one of the highest scoring natural sequences revealing that they are further away in the space than the central blob. The individual MCMC chains can actually be seen because they were not run long enough to mix in this high dimensional space.

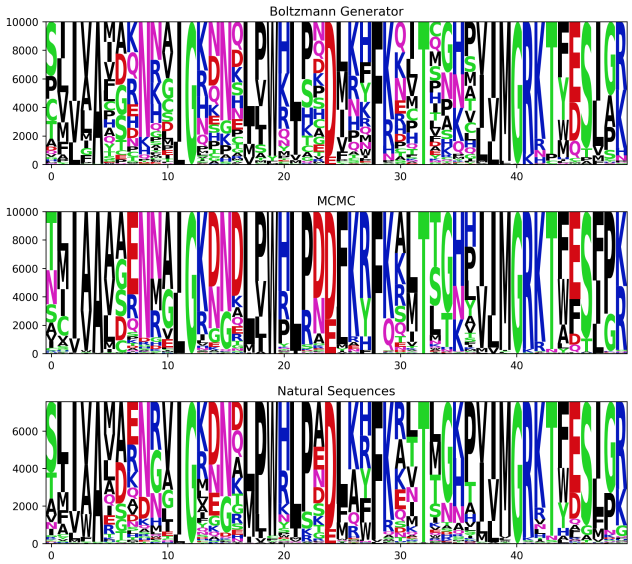

Figure 9: Sequence logos for the amino acid based Boltzmann Generator, MCMC (initalized with the highest scoring natural sequences) and the natural sequences. It is worth noting that the Boltzmann Generator seems to have a sequence logo that more closely matches the natural sequences than MCMC, further supporting the fact that the Boltzmann is able to capture a representative part of the sequence space but fails to extend far enough to capture the highest scoring more distant regions of space.

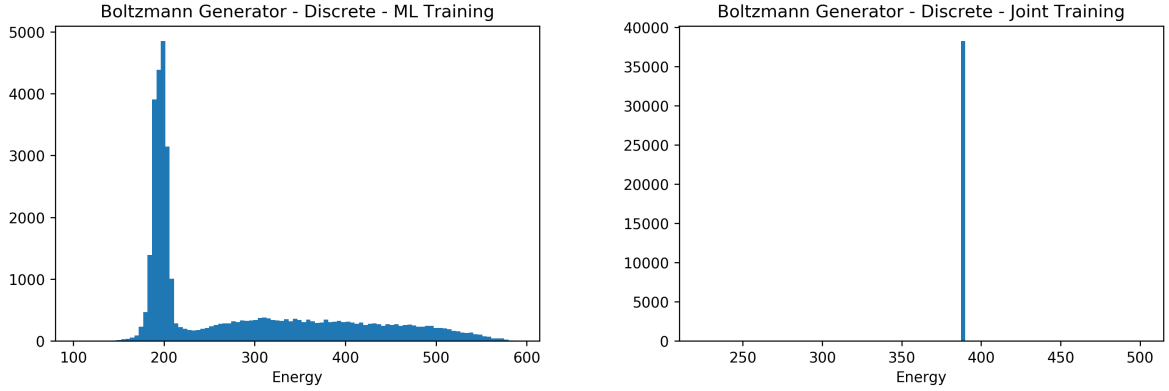

Figure 10: Histograms of the energies of the onehot Boltzmann model after ML training (left) and then additional joint training in both KLD directions (right). This mode collapse that occurs from training in both KLD directions occurs here even though the $D_{KL}[\mu_Z(\mathbf{z})||q_X(\mathbf{z})]$ direction that leads to mode collapse also only has half of the weighting as the Maximum Likelihood direction. Why the model concentrates so much on the 200 energy sequences during the ML training is unclear.

5 Conclusion

Our research on using the Boltzmann Generator to approximate protein fitness landscapes remains a work in progress. In order for the Boltzmann Generator to potentially become state of the art, further research into training with more powerful invertible networks, potentially even the discrete ones, and with more complex prior distributions needs to be explored. This is in addition to trying larger networks with more layers and alternative PFPs. Benchmarking our approach and MCMC against the other existing methods would also be useful as strangely, none of the other methods have compared themselves to MCMC. However, we are aware of the possibility that the Boltzmann Generator will always struggle to perform well on a discrete problem and one as multimodal as a protein fitness landscape.

Independent of whether or not the Boltzmann Generator is able to outperform existing methods, this work has yielded a number of contributions including:

(i) Discovering that the Evolutionary Couplings PFP has a closed form solution to its expectation over protein sequences;

(ii) Developing the first functional library for discrete normalizing flows;

(iii) Applying a novel formulation of variational inference to the new problem domain of discrete protein sequences.

Acknowledgements

I would like to thank Dr. Debora Marks, Dr. David Banks and Dr. Robert Thompson without whose mentorship, guidance and supervision none of my past, present or future research would be possible. I would also like to thank open source software contributors, including but not limited to: Numpy, Pandas, Scipy, Matplotlib, Pytorch, and Anaconda.

References

[1] Robert A. Langan, Scott E. Boyken, Andrew H. Ng, Jennifer A. Samson, Galen Dods, Alexandra M. Westbrook, Taylor H. Nguyen, Marc J. Lajoie, Zibo Chen, Stephanie Berger, Vikram Khipple Mulligan, John E. Dueber, Walter R. P. Novak, Hana El-Samad, and David Baker. De novo design of bioactive protein switches. *Nature*, 2019.

[2] Claire N. Bedbrook, Kevin K. Yang, J. Elliott Robinson, Elisha D. Mackey, Viviana Gradinaru, and Frances H. Arnold. Machine learning-guided channelrhodopsin engineering enables minimally invasive optogenetics. *Nature Methods*, 16(11):1176–1184, 2019.

[3] Ge Liu, Haoyang Zeng, Jonas Mueller, Brandon Carter, Ziheng Wang, Jonas Schilz, Geraldine Horny, Michael E Birnbaum, Stefan Ewert, and David K Gifford. Antibody complementarity determining region design using high-capacity machine learning. *Bioinformatics*, 11 2019. btz895.

[4] Adam Riesselman, Jung-Eun Shin, Aaron Kollasch, Conor McMahon, Elana Simon, Chris Sander, Aashish Manglik, Andrew Kruse, and Debora Marks. Accelerating protein design using autoregressive generative models. *bioRxiv*, 2019.

[5] 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.

[6] Surojit Biswas, Grigory Khimulya, Ethan C. Alley, Kevin M. Esvelt, and George M. Church. Low-n protein engineering with data-efficient deep learning. *bioRxiv*, 2020.

[7] Kevin M. Esvelt, Jacob C. Carlson, and David R. Liu. A system for the continuous directed evolution of biomolecules. *Nature*, 472(7344):499–503, 2011.

[8] Arjun Ravikumar, Garri A. Arzumanyan, Muaeen K.a. Obadi, Alex A. Javanpour, and Chang C. Liu. Scalable, continuous evolution of genes at mutation rates above genomic error thresholds. *Cell*, 175(7), 2018.

[9] Thomas A Hopf, John B Ingraham, Frank J Poelwijk, Charlotta P I Schärfe, Michael Springer, Chris Sander, and Debora S Marks. Mutation effects predicted from sequence co-variation. *Nature Biotechnology*, 35(2):128–135, 2017.

[10] Thomas A Hopf, Anna G Green, Benjamin Schubert, Sophia Mersmann, Charlotta P I Schärfe, John B Ingraham, Agnes Toth-Petroczy, Kelly Brock, Adam J Riesselman, Perry Palmedo, Chan Kang, Robert Sheridan, Eli J Draizen, Christian Dallago, Chris Sander, and Debora S Marks. The EVcouplings Python framework for coevolutionary sequence analysis. *Bioinformatics*, 35(9):1582–1584, 10 2018.

[11] Lukasz P. Kozlowski. Proteome-pi: proteome isoelectric point database. *Nucleic Acids Research*, 45(D1), 2016.

[12] Alan E. Gelfand and Adrian F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409, 1990.

[13] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457), 2019.

[14] 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, Feb 2017.

[15] Carl Doersch. Tutorial on variational autoencoders. *ArXiv*, abs/1606.05908, 2016.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[17] David H. Brookes and Jennifer Listgarten. Design by adaptive sampling. *CoRR*, abs/1810.03714, 2018.

[18] David H. Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design. *CoRR*, abs/1901.10060, 2019.

[19] Johannes Linder, Nicholas Bogard, Alexander B. Rosenberg, and Georg Seelig. Deep exploration networks for rapid engineering of functional dna sequences. *bioRxiv*, 2019.

[20] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.

[21] Christof Angermueller, David Dohan, David Belanger, Ramya Deshpande, Kevin Murphy, and Lucy Colwell. Model-based reinforcement learning for biological sequence design. In *International Conference on Learning Representations*, 2020.

[22] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

[24] Debora S. Marks, Lucy J. Colwell, Robert Sheridan, Thomas A. Hopf, Andrea Pagnani, Riccardo Zecchina, and Chris Sander. Protein 3d structure computed from evolutionary sequence variation. *PLoS ONE*, 6(12), Jul 2011.

[25] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *CoRR*, abs/1410.8516, 2014.

[26] George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. In *NIPS*, 2017.

[27] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *NeurIPS*, 2019.

[28] Dustin Tran, Keyon Vafa, Kumar Krishna Agrawal, Laurent Dinh, and Ben Poole. Discrete flows: Invertible generative models of discrete data. *CoRR*, abs/1905.10347, 2019.

[29] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.

[30] Andrej Karpathy. Pytorch normalizing flows, 2019.

[31] Leland McInnes and John Healy. Umap: Uniform manifold approximation and projection for dimension reduction. *ArXiv*, abs/1802.03426, 2018.

[32] Nathan J. Rollins, Kelly P. Brock, Frank J. Poelwijk, Michael A. Stiffler, Nicholas P. Gauthier, Chris Sander, and Debora S. Marks. Inferring protein 3d structure from deep mutation scans. *Nature Genetics*, 51(7):1170–1176, 2019.

[33] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *ICML*, 2015.

6 Appendix

6.1 Evolutionary Couplings Closed Form Expectation Proof

Evolutionary Couplings uses homologous proteins in a multiple sequence alignment to infer sitewise h_i and pairwise J_{ij} interactions between amino acids for a protein sequence of interest. This approach has been successful for predicting protein folds and protein design [9] [24] [10] [32]. The model is:

$$p(\mathbf{x}) = \frac{\exp(E(\mathbf{x}))}{Z}$$

$$E(\mathbf{x}) = \sum_i [h_i(x_i) + \sum_{i<j} J_{ij}(x_i, x_j)]$$

Where h is of a matrix dimension $L \times 20$ with L being the protein length and J is a tensor with dimension $L \times L \times 20 \times 20$ these can be thought of as lookup tables of parameters for each amino acid and position in the sequence.

h_i and J_{ij} are learnt using pseudolikelihood to avoid needing to compute the normalizing constant:

$$\log p(\mathbf{x}) = \sum_i \log p(x_i | \mathbf{x}_{-i}; h, J)$$

The fitness of a sequence $\mathbf{x}$ of dimension L with each amino acid mapping to a unique integer can be computed in a vectorized form where the sequence is converted to a onehot representation and flattened. Converting to onehot we have a matrix the dimensions $L \times 20$. Flattening we then have a vector with the dimension $L * 20$. We likewise flatten our sitewise matrix h and can get the sitewise score of this sequence by the dot product $h \cdot \mathbf{x}$. To obtain the pairwise J interactions we use our onehot sequence representation as a mask to eliminate pairwise interactions that are not present for this particular sequence. We flatten J into a matrix of shape $L * 20 \times L * 20$ and perform matrix multiplication with our onehot sequence $\mathbf{r} = J\mathbf{x}^T$ before applying applying elementwise multiplication and summing $(\sum_i x_i^T * r_i)/2$. We divide by two to account for the fact this J matrix is symmetric and we computed interactions $J_{ij}(x_i, x_j) + J_{ij}(x_j, x_i)$.

All together, we can compute the energy of a sequence in a vectorized form using the equation:

$$E(x) = h^T \mathbf{x} + (\sum_i x_i^T * (J\mathbf{x}^T)_i)/2$$

Using this vectorized equation, if the sequence represents a real probability distribution over each amino acid where at each position in the sequence x_i summing over the 20 amino acids with j we have $\sum_j x_{i,j} = 1$, then using the same equation as above we can compute the expectation over the probabilistic sequence $\mathbf{x}$ in a closed form.

6.2 Invertible Network Parameters

MAF, NeuralSplines, RealNVP were all unstable to train in both KLD directions. Approaches like clipping and transforming the energy of the PFP will need to be explored further but these invertible networks are well known to be unstable [29]. This left NICE, the least expressive of the networks as the most fully researched.

Except for the autoregressive invertible network MAF, the vectors were split in half by separating every even and odd amino acid. Which half was transformed and used as the invertible key was flipped after each transformation was applied.

NICE and RealNVP used four transformation blocks. The scaling and transformations in the networks were parameterized by vanilla feedforward neural networks with four layers each containing 32 hidden units and using leaky ReLU activation functions.

MAF also used four transformation blocks but with the MADE network architecture for its autoregressive functionality [33].

The Neural Splines used the same parameters except for the additional ones required, K=8 and B=3 [27].

Implementation of the invertible networks was aided by Andrej Karpathy's implementations which we were able to spot a bug in and contribute to [30].

The parameters used for each of the models implemented in PyTorch are as follows: Learning rate of 0.005; Gradient clipping of 1.0; Maximum Likelihood only training in the direction $D_{KL}[\mu_X(\mathbf{x})||q_Z(\mathbf{x})]$ for 10,000 epochs (this was important for ensuring the model was stable when trained jointly); Joint training for 5,000 epochs. Maximum Likelihood $D_{KL}[\mu_X(\mathbf{x})||q_Z(\mathbf{x})]$ batch size 256; Generator direction $D_{KL}[\mu_Z(\mathbf{z})||q_X(\mathbf{z})]$ batch size 1024; Generator direction weight of 0.5, Maximum Likelihood weight of 1.0. The MCMC chain for the Maximum Likelihood training was run for 3,000 steps across 64 chains taking the output at each 5th step with 10,000 data points randomly sampled from this.