THE JOURNAL OF PHYSICAL CHEMISTRY B
A JOURNAL OF THE AMERICAN CHEMICAL SOCIETY

pubs.acs.org/JPCB

Article

How Deep Learning Tools Can Help Protein Engineers Find Good Sequences

Published as part of The Journal of Physical Chemistry virtual special issue "Computational Advances in Protein Engineering and Enzyme Design".

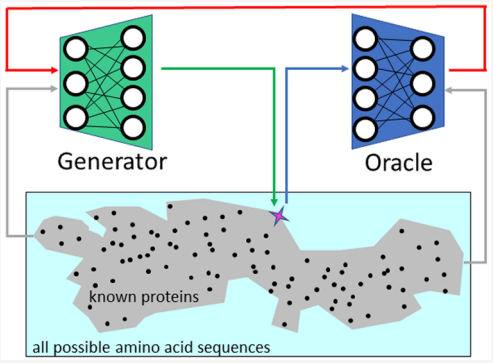
Margarita Osadchy and Rachel Kolodny*

Cite This: *J. Phys. Chem. B* 2021, 125, 6440–6450

Read Online

ACCESS | Metrics & More | Article Recommendations

ABSTRACT: The deep learning revolution introduced a new and efficacious way to address computational challenges in a wide range of fields, relying on large data sets and powerful computational resources. In protein engineering, we consider the challenge of computationally predicting properties of a protein and designing sequences with these properties. Indeed, accurate and fast deep network oracles for different properties of proteins have been developed. These learn to predict a property from an amino acid sequence by training on large sets of proteins that have this property. In particular, deep networks can learn from the set of all known protein sequences to identify ones that are protein-like. A fundamental challenge when engineering sequences that are both protein-like and satisfy a desired property is that these are rare instances within the vast space of all possible ones. When searching for these very rare instances, one would like to use good sampling procedures. Sampling approaches that are decoupled from the prediction of the property or in which the predictor uses only post-sampling to identify good instances are less efficient. The alternative is to use sampling methods that are geared to generate sequences satisfying and/or optimizing the predictor's desired properties. Deep learning has a class of architectures, denoted as generative models, which offer the capability of sampling from the learned distribution of a predicted property. Here, we review the use of deep learning tools to find good sequences for protein engineering, including developing oracles/predictors of a property of the proteins and methods that sample from a distribution of protein-like sequences to optimize the desired property.



INTRODUCTION

Engineering proteins calls for designing or identifying protein sequences that will fold into viable proteins to perform a useful technological or biomedical function.[1] The amazing range of biological functions performed by natural proteins leads one to believe that this can be accomplished. The challenge is finding such a sequence among the vast number of possible chains (e.g., there are 20^{200} possible chains of length 200).[1b] That there are so many possible sequences implies that only a minute fraction can be considered. This means that one can benefit greatly from optimizing the procedure of finding these sequences. Here, we review in-silico procedures and, more specifically, ones that rely on deep learning, to help with this task. Several recent reviews offer complementary perspectives on these topics.[1a,2]

Deep learning has shown significant progress in the past decade and has become a main driver of applications in many domains. We list a few: face/object recognition and analysis,[3] natural language and speech processing,[4] recommendation systems,[5] protein structure prediction,[6] GO playing, and autonomous vehicle control.[7] There are several factors that explain the tremendous advances in deep learning. The first two are large (and even huge) amounts of labeled data and powerful hardware that can process it in parallel and at high speed. The third factor is conceptual. Deep networks learn complex (hierarchical) features and a task over these features (e.g., classification, regression) in a single model with millions of parameters. These parameters are learned automatically when training the network by minimizing integrated and well-designed loss functions with the back-propagation algorithm and using improved optimization techniques.[8] This is very different from the traditional approach that applies machine learning models over hand-crafted features. Designing such hand-crafted features requires both effort and expert knowl-

Received: March 18, 2021
Revised: May 23, 2021
Published: June 9, 2021

https://doi.org/10.1021/acs.jpcb.1c02449
J. Phys. Chem. B 2021, 125, 6440–6450

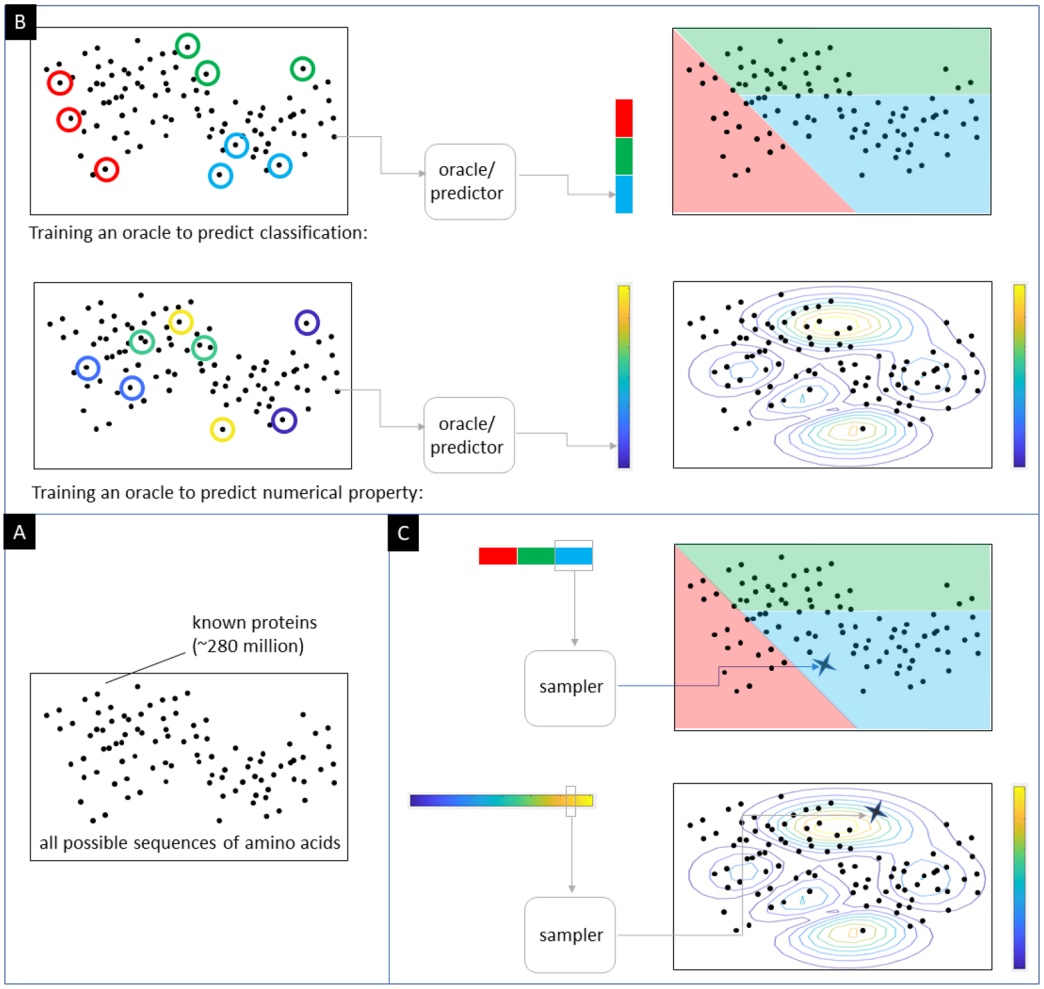


Figure 1. Designing deep network oracles to predict protein properties and samples to generate new sequences with predefined properties. (A) For the sake of illustration, we show the space of all possible amino acid sequences as a box. Among these sequences, some are protein-like (i.e., they are sequences that were selected by evolution to fold and function). Each point marked in this space represents a known protein sequences. (B) To train a deep network oracle, labeled samples are needed. Here, these are shown as points, within circles that correspond to their labels. The labels can be represented by a discrete set of values indicating a class (top; shown as the red/green/cyan classes) or by a continuous numerical value representing a property (bottom; the range of numbers is shown as a color scale). The trained oracle can then predict, given a protein sequence (or a point), the class (top) or the property (bottom). Hence, the oracle learned either a partitioning of protein sequence space (top, shown as differently colored regions) or a function over this space (bottom; shown as contour lines). (C) The goal of the sampler is to generate sequences that are not among the known proteins. We marked these with a star. Here, we show a conditional sampler: the top sampler generates a sequence that is defined by the class (top; here it generated a sequence that is not a known protein and belongs to the cyan class) or by the property desired (bottom; here the sample generated a sequence that is not a known protein in the region characterized by the yellow range).

edge, and success is not always guaranteed. The overall result is that deep models are much more accurate than traditional ones. Finally, implementing a tailored deep network is a relatively straightforward task due to widely available high-level open-source libraries (e.g., Keras, Pytorch, and TensorFlow).

Protein engineering can be implemented using two fundamental tools: (1) selection, to keep only promising leads that optimize a desired property, and (2) the generation of novel candidate sequences (sampling sequence space), including ones that can be selected to have the desired property. For the selection task, computational tools can be developed to predict if an amino acid sequence has some desired properties. This is analogous to the physical selection of fit sequences in an experiment. In computer science lingo, these tools are called "oracles", and deep learning can contribute to designing accurate and fast oracles. More intrestingly, deep networks can also contribute to the task of generating novel candidate sequences. Specifically, deep models offer many tools for sampling, either in the vicinity of a given sequence (akin to random mutagenesis) or even a more general sampling. Sampling is implemented via the so-called generative models and can be guided by the distribution learned from the data (implying it will generate only protein-like sequences) and tuned to optimize a desired property such as that predicted by a given oracle. These tools can then be used in an in-silico process that mimics the physical experiment of directed evolution: random mutagenesis could be replaced by a more sophisticated sampling of novel sequences, and an oracle could be used to evaluate the generated sequences in the selection step. By using the oracle's predictions to keep only promising leads, one can optimize the desired property.[9]

Here, we survey deep-learning-based computational tools that were designed for in-silico protein engineering. Our focus

https://doi.org/10.1021/acs.jpcb.1c02449
J. Phys. Chem. B 2021, 125, 6440−6450

is on deep learning methods for a better sampling of the sequence space, namely, the sampling of novel sequences that will have a desired property. To do this, different deep learning tools are used, each with its own nuiances. To review these topics, we first offer a high-level summary of oracle design, focusing on deep learning oracles. Then, we briefly describe the theoretical background of deep generative models that can be used for the sampling of protein sequence space. Finally, we survey studies that used these generative models for efficient sampling, emphasizing the innovative uses of oracles to guide generative model design.

FAST AND ACCURATE DEEP LEARNING ORACLES FOR PROTEIN DESIGN

The first computational tool needed for in-silico protein engineering is an oracle. The oracle, when given a protein sequence as input, predicts if this sequence encodes a valid protein and if it has a particular property. Oracles either predict the presence of a property for the input sequence (for example, if a label, such as a specific SCOP fold, should be assigned to it[10]) or predict a numerical estimate to score a property[11] (Figure 1). In other words, the oracle implements either a partitioning of sequence space according to the labels or a function over sequence space.

Oracles, or in-silico tools that predict protein properties, were developed long before deep learning emerged within this field. Surveying computational methods for predicting protein properties from their sequences is obviously far beyond the scope of this review. In fact, oracles vary in their approaches, including, for example, focused machine learning (ML) software tools developed on relatively small data sets (e.g., random forests as in Müller et al.[12]). Oracles are also tools that derive properties of homologous proteins found using a sensitive sequence search (e.g., with HHSearch[13] or HMMER[14]). Finally, oracles are also comprehensive scientific packages such as Rosetta[15] that can fold the protein in silico to predict its properties.

When machine learning, particularly deep learning, is used to design an oracle, this is typically viewed as a "supervised learning" task. To this end, one has to collect enough instances of proteins with this property (denoted "labeled sequences") and use them to train the oracle. The trained network, or the oracle, learns the statistical relationships within the set of sequences that have this desired property: namely, it encodes these in the parameters of the deep network. Notice that ML models can infer biological properties of unseen sequences without understanding the physical or biological mechnanisms.[16] In other words, neither the network nor the person who designs it needs to understand these mechanisms, but, of course, protein domain knowledge is invaluable when designing oracles. Another advantage that deep network oracles have with respect to physics-based tools, which fold the protein sequence to measure its properties, is that they are much faster. The difference in speed can be several orders of magnitude.[17] Thus, deep networks offer an attractive computational approach for oracle design.

Designing a deep learning oracle requires selecting the target function and choosing a meaningful representation of the protein sequence and the so-called architecture of the network. Any biologically meaningful target function can be studied if one can collect enough labeled data or sequences for which they have trusted measures of that target function. There are many examples of deep network oracles that have been designed for different target functions. The most well known ones are the oracles that predict protein structures from a sequence[6] including ones that shined in the last CASP experiment.[18] Other examples of deep network oracles include predicting the effects of single mutations,[11,17a] predicting protein function[19] (using the GO anthology[20] or EC numbers[21]), predicting if a sequence is an AMP,[22] predicting the protein family/superfamily/fold,[10,23] and predicting the protein stability, localization, or fluoresence intensity.[16,17,23b]

Most deep learning oracles for studing proteins use off-the-shelf architectures implemented with deep learning software libraries. The choice of the architecture is dictated by the representation (e.g., sequence, matrix/tensor, graph) and the amount of training data. Each architecture exploits specific structure and correlations in the input and provides a so-called inductive bias in learning. In the case of protein engineering, the input to the oracle is a sequence, so the architecture should be able to process sequential data of varying length. The varying length can be handled naturally by some deep architectures (e.g., fully convolutional networks) by using a fixed-length description that describes the protein as a whole, or by a simple heuristic that adds a padding symbol for augmenting shorter sequences. The most common choice for processing sequential data has been Recurrent Neural Networks (RNNs) and their variants, including LSTM,[24] attention models[25] that target longer sequences, and mLSTMs[26] that offer better expressiveness. For example, RNN-based oracles were used for predicting protein functions.[17a,19] Dilated CNNs[27] and dilated residual networks[28] are also common for modeling sequential data. CNN-based oracles were considered in references 16 and 17b. Finally, a transformer and its variants[4b] provide depth (in terms of network processing layers), high expressiveness, and sequential input/output. A nice property of all of these architectures is that they naturally lend themself to self-supervised training.

A common approach to leveraging a large set of unlabeled data is using self-supervised learning to train a network. This allows the network to learn statistical relationships within the input space. When trained on protein sequences in a self-supervised manner, the network is tasked with predicting parts in the sequence that were deliberately hidden from it (either the next amino acid, given a prefix,[17a] or masked amino acids[23b,29]). Thus, the self-supervised training does not require any property labels and can use all known protein sequences. The network learns meaningful features for "protein-like" sequences (i.e., sequences that survived evolution and can fold and function). Each amino acid in a protein sequence has its physical properties (e.g., hypdrophobicity, charge, or size), which is only implied in the sequence representation. Theoretically, one could imagine hand-crafting a representation of the protein that includes all (and only) relevant features. The advantage of learning and embedding is that the network learns these features from the data and on its own.[16] This step is also called pretraining due to how one plans to use the resulting network. This resulting network can be fine-tuned more quickly and more accurately to perform a specific downstream task using a smaller labeled set. Alternatively, it can be configured to output a fixed-length representation of the input sequence, which constitutes a new compact representation that encodes important and relevant properties. These can be used for visualization, additional analysis, or even as a new data set for training machine-learning oracles. This approach

https://doi.org/10.1021/acs.jpcb.1c02449
J. Phys. Chem. B 2021, 125, 6440−6450

was popularized in natural language modeling and is thus referred to as a language model.

Several language models were developed for protein sequences, with the goal of capturing semantically meaningful representations. These models include simple word embedding,[30] LSTM and its variants (e.g., ELMo[29a] and mLSTM[17a]), a transformer and its variants, and generalizations (e.g., Bert,[23b,29b,31] Albert, Transformer-XL, and XLNet[31]). Transformers offer a larger capacity than LSTM while keeping computations feasible due to parallel computations that are impossible in RNNs. Another advantage of BERT over LSTM is that it incorporates context from the entire sequence, while LSTM is directional. In ELMo,[32] this is compensated for by first pretraining on sequences and their reverse and then concatenating the output of unidirectional models applied in both directions. This does not allow the language model to use bidirectional context during training, but it allows supervised networks to combine context derived independently from both sides.[31] Indeed, language models of proteins based on transformers have been shown to perform better than LSTM models.[23b,29b,31,33] This could be attributed to both the larger capacity and the bidirectional context. However, one-directional models with an additional increase in capacity failed to outperform autoencoding (BERT) and bidirectional models,[29a] indicating that capacity is not the only requirement and bidirectional context plays an important role.

Representations, learned by a language model, are validated in two indirect ways: (1) comparing known relationships among amino acids or among proteins to the relationships among their representations and (2) testing the success of the language model in downstream tasks. For per amino acid, the relationships among their representations can be visualized in low dimensions after using a similarity-preserving projection such as PCA or t-SNE: Alley et al.,[17a] Rives et al.,[29b] and Filipavicius et al.[33] show that their language models successfully cluster amino acids of similar known properties (size, charge, and hydrophobicity). Bileschi et al.[17b] further show that the similarities among the amino acid representations is similar to BLOSUM62. For full sequences, Alley et al.[17a] show in their language model that proteins from similar organisms have similar representations, as do proteins with similar structural characteristics. Rives et al.[29b] show that homologous proteins have similar representations. Moreover, in Bileschi et al.[17b] the downstream task was identifying homologues, and they show that using distances among embedded sequences suffices for this task. Indeed, that a language model captures similarity information means that it includes information about the position of the sequence within the protein space. Many different downstream tasks were used to validate language models: these include a prediction of per residue structural properties (disorder and secondary structure),[29,33,34] protein classification (e.g., SCOP/enzyme class or GO label),[30,34,35] subcellular localization,[29a,33] thermostability,[11,16] or homology detection.[17b,35a] As Rao et al.[23b] argue, language models can be compared on the basis of their downstream task performance only if the same task and data sets are considered because the performance is due to both the language model and the oracle or otherwise to the specialized network that uses it. They curated five such tasks to compare language models, and their results suggest that different language models excel at different tasks, leaving more room for improvement on this front.

It should be noted that the overall prediction accuracy was not high in any of the language models. Some of the error is due to inherent ambiguity (or noise), but not all of it. While all protein language models are based on the assumptions that protein sequences are similar to sentences in natural language, one can argue that the similarity is in representation only but the structure of the correlations could be different. Consequently, building models with inductive bias that focuses on proteins is an important future research challenge.

QUICK INTRODUCTION TO GENERATIVE MODELS

Generative models summarize the information about the data set D using a finite set of parameters θ. The parameters are estimated by minimizing some notion of distance between the model distribution (from a chosen family, e.g., Gaussian distributions) and the data distribution. Generative models are used for three tasks, all employed in protein engineering: (1) density estimation: given a data point x, estimate the probability $p_\theta(x)$; (2) sampling: generate a new data point from the distribution $x_{\text{new}} \approx p_\theta(x)$; and (3) unsupervised representation learning: when the labeled data set is sparse or even unavailable, one can use generative models for unsupervised pretraining of the model to derive a meaningful representation for a data point x. We provide a brief overview of a few popular generative models.

Autoregressive Models. Protein sequences are ordered strings. For this type of multivariate random variable x, one can factorize the probability $p(x)$

$$p(x) = \prod_{i=1}^{n} p(x_i|x_1, x_2, \ldots, x_{i-1}) = \prod_{i=1}^{n} p(x_i|x_{<i})$$

where $\mathbf{x}_{<i} = [x_1, x_2, \ldots, x_{i-1}]$ denotes the vector of random variables with an index of less than i. Here, the distribution for the ith random variable depends on the values of all of the preceding random variables in the sequence $x_1, x_2, \ldots, x_{i-1}$. One can use parametrized functions (e.g., those implemented by neural networks) to predict the next element given all of the previous ones. For a categorical distribution (such as a protein sequence of amino acids), a softmax is used to transform a vector of K numbers (typically, for amino acids $K = 20$) into a vector of K probabilities (non-negative that sum to 1):

$$\begin{aligned}\text{softmax}(\vec{y}) &= \text{softmax}(y_1, \ldots, y_K) \\ &= \left(\frac{\exp(y_1)}{\sum_j \exp(y_j)}, \ldots, \frac{\exp(y_K)}{\sum_j \exp(y_j)}\right)\end{aligned}$$

To control sequence variability while sampling, it is common to incorporate a temperature factor T into the softmax, and picking an amino acid at a sequence position i is done with a temperature-controlled probability:

$$p(y_i) = \frac{\exp(y_i^{1/T})}{\sum_j \exp(y_j^{1/T})}$$

Autoregressive models are general and can represent any possible distribution over n random variables. Even though the space complexity for such a representation grows exponentially with n, density estimation in autoregressive models is both simple and efficient on modern hardware. However, sampling from an autoregressive model is a sequential procedure, which is slow. Teacher forcing is used in training and not in sampling;

https://doi.org/10.1021/acs.jpcb.1c02449
J. Phys. Chem. B 2021, 125, 6440−6450

therefore, sampling often requires various amendments (e.g., for reducing repetitions).[36] Finally, an autoregressive model does not directly learn the latent representations of the data.

The most popular examples of autoregressive models are recurrent networks (e.g., RNN, LSTM[24]), autoregressive convolutional networks (e.g., Wavenet[27]), and self-attention models (e.g., transformers[4b,37]). Indeed, autoregressive models were widely used in modeling protein sequences (e.g., refs 17a, 23b, 29b, 31, 34, and 38) and in sampling novel sequences.[12,35b,39]

Energy-Based Models. Any function $f_\theta: \mathcal{R}^d \rightarrow \mathcal{R}$ can be transformed to a distribution by exponentiation and normalization,

$$p_\theta(x) = \frac{\exp(-f_\theta(x))}{Z(\theta)}, \; Z(\theta) = \int_{\mathcal{R}^d} \exp(-f_\theta(x))\, dx$$

This provides no restriction on f_θ, but even when $f_\theta(x)$ is easy to compute (e.g., using a neural net), the density $p_\theta(x)$ is hard to normalize. Also, it is hard to train the model because one cannot run back-propagation through the generation process. Rather, to compute gradients, computationally expensive MCMC sampling is required. Once the model is trained, MCMC is also used for sampling.

Energy-based models offer a flexible class of models, but the above-mentioned computational issues make them less popular. Most known examples of energy-based models are restricted Boltzman machines (RBM)[38] and deep Boltzman machines.[40] In protein design, Tubiana et al.[23a] used RBMs to model the complex interactions among the residues in the sequences of specific protein families.

Variational Autoencoders. Variational autoencoders (VAE)[41] are a powerful class of deep generative models with latent variables that can capture a hidden structure in the underlying data. Let z and x denote the latent and observed variables, respectively. The joint distribution can be expressed as $p_\theta(x, z) = p(z|x)\, p(z)$. If a latent variable z encodes semantically meaningful information about x, then the generative process can be viewed as first generating the high-level semantic information $z \approx p(z)$ and then generating the sample given that information: $x \approx p(x|z)$.

Similar to autoencoders, VAE learns to encode semantically meaningful information in the latent space by encoding the training sample in a latent representation and decoding it back to the input. (Both the encoder and decoder are represented by deep neural networks.) The key modification that enables sampling is that instead of encoding an input as a single point, it encodes it as a distribution over the latent space. VAE is trained to minimize the negative evidence lower bound (ELBO) that comprises the data reconstruction term and a regularization term that forces the approximate distribution $q(z|x)$ produced by the encoder to stay close to the assumed distribution $p(z)$. The regularization term is expressed as the Kulback−Leibler divergence between the two distributions. Since for simple $p(z)$ and $p(x|z)$ we can still have a complex $p(x)$, in practice, these distributions are chosen to be Gaussian. The generation process in VAE is performed by first sampling in the latent representation (from a Gaussian distribution) and then decoding the sample by passing the latent variable through the decoder.

VAEs allow us to build flexible models, offer a simple sampling procedure, and are suitable for the unsupervised learning of latent space. They are less useful for evaluating likelihoods because their exact estimation is intractable. VAEs have been widely used in many domains, including protein sequence modeling and sampling.[11,42]

Normalizing Flow Models. Normalizing flow models[43] are latent variable models with tractable likelihoods and easy sampling mechanisms. This is an improvement over autoregressive models that provide tractable likelihoods but have no direct mechanism for learning features and are over VAEs that can learn feature representations but have intractable marginal likelihoods. The key idea in normalizing flows is to construct a generative model by mapping simple distributions (that are easy to sample and evaluate densities) to more complex distributions (learned via data) using a change in variables and invertible transformations. Such invertible transformations can be composed with each other, starting with a simple distribution of a latent variable (e.g., Gaussian) and applying a sequence of M invertible transformations by variable changes. The challenge is designing these invertible transforms.

Examples of normalizing flow models are NICE,[44] real-NVP,[45] inverse autoregressive flow,[46] and masked autoregressive flow.[47] While normalizing flows produced very realistic samples in the domain of face generation (GLOW[48]), we are unaware of their application in protein engineering.

Generative Adversarial Networks. Another alternative is generative adversarial networks (GANs) that express model training as a minimax two-player game and offer a (likelihood-free) model, purely for sampling purposes. A GAN is composed of a generator and a discriminator. The generator G_θ is a neural network that deterministically generates samples x from a random input latent variable z. The discriminator D_φ is a neural network that distinguishes between samples from the real data set vs those from the generator $G_\theta(z)$.

The generator and discriminator play a two-player minimax game, where the generator tries to fool the discriminator by generating samples that look indistinguishable from the real data and the discriminator tries to distinguish generated and real samples. Formally, the GAN objective can be expressed as

$$\min_\theta \max_\varphi L(G_\theta, D_\varphi) = E_{x\sim p_{data}}[\log D_\varphi(x)] + E_{z\sim p(z)}[\log(1 - D_\varphi(G_\theta(z)))]$$

where p_{data} is the distribution of real data and $p(z)$ is the latent distribution (e.g., Gaussian).

GANs are notoriously hard to train because of instability issues, causing the generator and discriminator loss to oscillate without converging. Because of the lack of a robust stopping criteria, it is difficult to know when to stop the training. Furthermore, it is sensitive to the choice of hyperparameters and experiences undesirable effects such as mode collapse (a stuck generator that produces one of a few types of samples repeatedly). Better loss formulations such as the Wasserstein GAN,[49] gradient penalty,[50] and optimization methods (e.g., ref 51) have been proposed recently. There are many variations of GAN that are applied to different problems showing impressive results, particularly in images.[52] In protein engineering, GANs have also been used to generate novel sequences.[23c,53]

USING DEEP NETWORKS FOR SAMPLING SEQUENCES IN PROTEIN ENGINEERING

Protein Engineering as an Optimization Problem for the Oracle's Function. Identifying sequences for which an

https://doi.org/10.1021/acs.jpcb.1c02449
J. Phys. Chem. B 2021, 125, 6440−6450

oracle (of a given property) will predict a high value can be, and was indeed, studied as an optimization problem (Figure 1C). Because this is an optimization, we focus on oracles which assign protein sequences numerical values for a property that they predict (i.e., oracles that are functions mapping from sequence space to real values). The challenge is identifying sequences that are assigned the highest value. For example, had the oracle been described as a simple closed-form differentiable function (which is rarely the case), the function could be differentiated to find the input that maximizes it. In a more practical settings, optimizing means searching the input space for an optimum of the function: if the function is differentiable, then one can use its gradients to speed up the search or otherwise use methods such as Monte Carlo sampling.[54] Often, complicated oracle functions encode a so-called "rugged" function, which is characterized by many local minima. A gradient-based optimization may be stuck in these many local minima. More importantly, these local minima may suggest that the oracle assigns comparable values to different sequences that lie in different regions of space (i.e., that are not merely small variants of each other). Bearing in mind that oracles are not perfect predictors, if there are multiple local minima with approximately the same values, then we would like to consider them all, not just the one that happens to be assigned the best value. Thus, we want the optimization to find all (or at least many) sequences that the oracle identified as having the property. For the sake of having a relatively succint sample set, we would like only a few representatives (or even just one representative) from separate regions of sequence space. Finally, the search must be constrained to the very small part of sequence space[55] with protein-like sequences.

If the oracle that predicts the property for a sequence is implemented as a deep neural network, then methods such as activation maximization (AM)[56] can be used to optimize its input. AM was first used to visualize an image that the oracle "prefers" to see for assigning the specific label. The input that maximizes the score of the network for that label does not necessarily lie near typical samples presented to the network. In images, this results in solutions that do not look like a natural image (various regularizers based on image statistical priors have been used to force the solution to be image-like). For protein (or DNA) sequences, this means that simply identifying inputs that maximize the score is not enough. Rather, one must condition the sequences to be protein-like, for example, by constraining the optimization to the solutions from the distribution of valid proteins. An implicit way to do this is using a generative model that was trained on a large set of proteins. The generator that produces a candidate solution to the AM already learned the distribution of protein seqeunces. This approach was taken by Killoran et al.[53b] and Costello and Martin.[57] Killoran et al.[53b] use a trained generator G from a GAN, which is expected to produce sequences with the property of being protein-like. The predictor network for the target property is pretrained on the corresponding labeled data set. The optimization is done over the latent variable z, which is converted to a protein-like sequence $x = G(z)$, for which the predictor estimates the predicted property $Pred(x)$. Costello and Martin[57] use a data set of proteins to train a hybrid model, combining a VAE with an autoregressive network added on top of the decoder. They add the autoregressive decoder to help the model train better on long sequences in the hope that a more informative latent representation will be formed. By using the decoder part of this pretrained network, a latent variable, sampled from the latent space (a Gaussian), is expected to be decoded into a protein-like sequence. For sequence design, they search over the latent space with the goal to minimize not only the squared distance between the predictor's output and the target value but also the sum-of-square distances between predictions of different properties and their target values.

Using a pretrained generator to map the latent space to protein-like sequences assumes that any z from the latent space will be decoded to a valid protein, even though in practice this is difficult to guarantee. To improve the VAE latent space, Costello and Martin[57] trained it with an improved loss that forces the latent space to be informative. (In some cases, ELBO loss for training VAE causes it to ignore the latent variables.[58]) There is, however, some evidence that VAEs trained on protein sequences do learn informative features. For example, Ding et al.[42b] showed that even with a simple network the latent space captures evolutionary relationships and that for the specific protein family cytochrome p450, for which they had a small labeled set, they could train a predictor of T_{50} (the temperature at which 50% of the protein is inactivated irreversibly after 10 min). In short, the above-mentioned studies attempt to find an optimal input sequence that when fed to a generator produces a novel valid sample on which the predictor excels.

An alternative to this approach is to iteratively refine the generator itself by retraining it on sets that are repeatedly updated to include sequences that are scored favorably by the predictor. Gupta and Zou[53c] call this approach feedback GAN. They train a GAN to generate protein-like sequences and score the generated sequences with two predictors: one predicting if the sequences encode for antimicrobial peptides (AMPs) and the other predicting secondary structures. They show that by replacing the (oldest) sequences in the training set with high-scoring generated sequences, the retrained generators produce sequences with the target properties enhanced. Since the predictor is used only to score the generated sequences, it need not be differentiable. Rather, the oracle can be treated as a black box. Listgarten and co-workers[42e,f] extended this idea to adaptive sampling. Adaptive sampling reweights the training samples based on the predictor's scores. They present a principled probabilistic argument to state that this will lead to generating sequences with an improved target property. In their case as well, the oracle is a (not necessarily differentiable) black box. Brookes and Listgarten[42f] demonstrate the utility of their method by showing that it excels in comparison to previous ones, including[53b,53c] a variant that adds feedback to a VAE generator. The protein engineering task they address is generating DNA sequences optimized for the high expression of their encoded proteins. Their competitive advantage is due to exploring more regions of the sequence space.[42f] Note, however, that diverging in sequence space away from the original set on which the predictor was trained carries the risk that the predictor will no longer be reliable. Brookes et al.[42e] show how to update the training set while avoiding the regions in sequence space where the predictor should not be trusted.

Tubiana et al.[23a] argue in support of using energy-based models because of their potential to explain the learned patterns. Namely, by restricting a generative model to a specific region of sequence space such as that defined by a family of proteins, one can better understand the nature of these generated sequences. Using RBMs to learn statistical interaction patterns within the sequences of 20 protein

https://doi.org/10.1021/acs.jpcb.1c02449
J. Phys. Chem. B 2021, 125, 6440−6450

families, Tubiana et al.[23a] show that they can interpret these patterns in a biologically meaningful way to describe structural, functional, and phylogenetic features.

Sampling from Conditional Models to Engineer Novel Sequences. The simplest way to generate sequences with a desired property is to train a generative model on a data set of sequences possessing these properties, if such a training set is available. For example, Repecka et al.[53d] trained a GAN model on a family of bacterial malate dehydrogenase (MDH) enzymes (EC 1.1.1.37) to generate sequences of proteins in the same family. The number of training samples was 16 706 (with an average sequence length of 319), which is relatively small in the context of training a GAN having approximately 60 million parameters. Their model samples sequences that have many desirable properties; namely, the generated sequences follow patterns that are characteristic of proteins in this family and at the same time expand the sequence space of the training data. Among the 60 generated sequences that they tested experimentally, 19 were soluble and 13 were catalytically active in vivo. While the generated sequences are not identical to the ones in the training set, they closely resemble them, perhaps suggesting that the limited training set leads to overfitting in this case.

One way to alleviate the problem of overfitting is to create larger data sets by pooling together multiple labeled sets and training a conditional generative model. In conditional generative models, the input to the model also includes the desired property as a conditional variable c, and the output consists of samples $x|c$ that have this property (Figure 1C). Training a single conditional model is not the same as training separate models for each property since the conditional model is capable of parameter sharing. Thus, samples with different values of the conditional variable contribute to training the shared parameters. In other words, training a conditional model is better.[39] There are different ways to implement conditional models. We review two implementations[23c,42a] of this for protein engineering.

Karimi et al.[23c] trained a GAN conditioned on a variable that encodes the structural fold; this variable was obtained from another model. In conditional GAN (cGAN), the conditional variable is given to both the generator and the discriminator. One of many ways to encode the conditional variable into the generator and discriminator models is using an embedding layer followed by a fully connected layer with a linear activation that scales the embedding to the size of the input before concatenating it as an additional channel. To calculate the value of the conditional variable describing any one of 1232 SCOP folds, Karimi et al.[23c] embed these on the basis of the structural distances among them and by using kernel principal component analysis.[59] They encode the conditional variable of a fold as its coordinates in the space spanned by the top 20 eigenvectors and use them in training a conditional Wasserstein GAN (cWGAN) with a gradient norm penalty in a two-step process. First, cWGAN is pretrained using protein sequences from UniRef50 with the conditional variable fixed in the center of all fold representations. Then, it is refined using a smaller set of sequences with their corresponding fold representations. This initial model is further improved by incorporating the gradients from a deep network oracle for fold prediction into the generator's loss, called guided cWGAN.

Greener et al.[42a] train a conditional VAE (CVAE)[60] for the unsupervised design of metal binding sites in pre-existing proteins and for producing proteins that fold into specified, including novel, topologies. For the metalloprotein design task, the model is conditioned on the binding ability of eight metals (Fe, Zn, Ca, Na, Cu, Mg, Cd, and Ni). The one-hot representation of a sequence is concatenated with a 1D tensor containing eight values of either zero or one to denote the absence or presence of one of the eight metal binding sites. Training is performed on a combined data set of ∼150 000 sequences, each with up to 140 residues. To modify a protein sequence that binds one metal into one that binds another, they first encode the sequence binding the original metal, with the original metal as the conditional variable using the encoder part of CVAE to get the mean and variance for the input sequence. Then the decoder of CVAE is applied to the samples from the obtained Gaussian but with the new metal as the conditional variable. Changing the conditional variable during decoding is not a standard use of CVAE. Since the decoder was trained to predict the original input, changing the conditioning challenges it with an unfamiliar task (namely, one that it was not trained for); therefore, the resulting generations may be invalid. Because filtering out bad generated samples or selecting the best ones is difficult, they train another oracle to predict the metal binding potential using the same labeled data set as in the CVAE training. Among 1000 generated sequences, they choose the one with the highest prediction for the desired metal. They show that the generated sequences include more residues that are known to bind the metal encoded in the conditional variable. Also, when presenting the trained CVAE with mutated sequences with the crucial residues (e.g., copper-binding histidine) removed, many of the generated modified sequences recover these critical residues.

Greener et al. implement another CVAE, which is conditioned on context-free grammar—a string—that describes the protein topology.[61] Their combined training data set includes ∼100 000 structures (with topology strings derived from SCOP). To design a sequence with a new topology, the decoder is sampled to produce 1000 different sequences from across the space. The sampled sequences are analyzed by an oracle based on Rosetta to find the best sequence among them, and this sequence is then fed into the encoder to obtain the mean and standard deviation of its latent code. Then, 1000 latent vectors are sampled from this Gaussian and decoded into a new set of candidate sequences. The authors suggest that such sampling is analogous to searching the space locally around the best sequence to find potentially better sequences. The process is repeated several times to traverse the search space before finding the final best sequence. In this task as well, the Rosetta oracle is used only for selection (i.e., not for training).

Models That Design One Amino Acid at a Time. In generative models that return probabilities for all possible amino acids (converted from scores using softmax, possibly after temperature adjustment), the probabilities can be used to pick the amino acid at random. Other alternatives include sampling only from the top k, possibly with a repetition penalty,[35b] or a beam search.[62] These generative models, denoted as autoregressive, can be used to generate an entire sequence by outputting the amino acids one by one along the polypeptide chain, each time using the model's previous predictions to condition the output of the next amino acid. For example, Müller et al.[12] trains an LSTM on a set of 1554 AMPs, and generates new sequences with this model. Notice that such generative models can also be used for the in-silico

https://doi.org/10.1021/acs.jpcb.1c02449
J. Phys. Chem. B 2021, 125, 6440−6450

mutagenesis of a given sequence, by sampling amino acids only in some of the positions in the sequence.

Autoregressive models are also very amenable to conditioning: One can condition on both discrete and continuous variables on multiple scales. One can even condition on latent embedding or the outputs of other neural networks.[25] In protein engineering, descriptors of the protein structure were used to condition autoregressive models. Once the model is trained, it can be used to design sequences for a given structure/fold. For these problems, the conditional variable c encodes the protein structure.[23c,60] Note that using a conditional model also implies that the data sets that can be used to train the model are very large because you can combine the data sets of different conditions. Stokach et al.[63] use this strategy and condition their network on protein structure. Their data set thus includes all protein sequence−structure pairs (70 million sequences and 80 thousand structures). When generating a sequence conditioned on c, in contrast to CVAE and cGAN described above (where the generating network is presented with two inputs: the latent code z and the conditioning variable c), in autoregressive models, such as RNNs and transformers, the initial input is only the conditioning variable. The intuition is that the generated sequence captures correlations that encode forces between interacting amino acids or amino acids that are compatible with the input structure.[39,63] The details vary: Ingraham et al.[39] use graph input for a transformer and encode structure as backbone dihedral angles and relative orientations among residue pairs. Strokach et al.[63] use graph input (encoded as the contacts in the structure) and a ResNet model. Madani et al.[35b] trained a conditional transformer model[36] with a larger and more complex set of conditional tags that encode a variety of annotations, including taxonomic, functional, and location information. Their model is trained on ∼280 M protein sequences conditioned on these tags, with the goal of providing a method for protein generation that can be tailored for the desired properties.

Increasing Diversity in Generated Sequences. The issue of diversity still lingers: if the sequence space has multiple regions that are optimal, then we would want the generator to output representatives from many (or even all) of them. The concern is that by optimizing a generated sequence set with a desired property, starting from an initial set, we stay in the same region rather than exploring diverse regions. Anishchenko et al.[54a] rely on a predictor for protein structure (they use the state-of-the-art trRosetta which predicts distance maps) and Monte Carlo sampling of sequence space to generate diverse protein-like sequences. Starting from a random sequence, they (probabilistically) accept the random walk in sequence space based on a score that measures if the internal distances predicted for this sequence are not merely the background (namely, they used the contrast or the KL-divergence between the distribution of distances predicted for the sequence and a background distribution). Diversity is achieved by using many different random starting points.

Linder et al.[53a] tackle the issue of diversity more directly: they develop deep exploration networks (DENs), which are deep generative models that explicitly maximize diversity. To do this, they train a generator network G with a modified loss function, which includes a term that penalizes cases where two sequences generated from randomly chosen latent variables are similar. They optimize the parameters of the generator network G using gradient descend, meaning that the predictor needs to be differentiable (they use a differentiable neural network). Generating diverse sequences may cause the network to diverge to faraway regions in sequence space. To avoid the regions where the predictor is no longer reliable, they use a VAE to keep the generation within the distribution of the samples they trained on.

Evaluating the Success of Generative Models. To evaluate the success of a generator for protein sequences, one would like to verify that the sequence is novel. (In Figure 1C, this means that the generated sequence marked by a "star" is not a point representing a known protein and is even not very similar to one.) One would also like to verify that the generator indeed outputs sequences with the desired properties. Because the properties are either a structure or a function that depends on this structure, the generative model must learn and recapitulate relationships among protein sequences, structures, and functions. We know, however, that sequence−structure−function relationships are complicated. For example, even when focusing only on the sequence−structure relationship, there are many different sequences that can fold to the same structure. At the same time, very similar sequences can fold to different structures.[64] This suggests that evaluating the success of generating a protein sequence for a given structure or property is also complicated.

As a first step, different computational methods were used to evaluate generated sequences. In the case of autoregressive models, one can measure the prediction of the true amino acid. Notice however that since many sequences can fold to the same structure, there is more than one correct answer. Indeed, autoregressive models achieve only 30−40%[39,63,65] accuracy, in agreement with our knowledge that there are many instances of protein sequences that are only 30% similar and still fold to the same structure. Considering that quality measures of the whole generated sequence or over-the-top scoring sequences are even more common (e.g., their secondary structure or hydrophobicity properties[12]), especially if these are properties for which reliable oracles exist. Generators can also be evaluated by the level of diversity of the set of sequences they designed.[53a,d,54a] For promising candidates, the structure of the sequence can be predicted by homology modeling or, in cases where the generated sequence is very different from the known proteins, by ab initio prediction. One can then compare the predicted structure to the desired one. Also, the energy of the conformation of this sequence can be calculated using Rosetta, and the stability of the conformation can be derived from MD simulations. On the basis of the predicted structure, other properties can be evaluated if we have an oracle that can predict them on the basis of structure. However, the final and most significant test is successfully expressing the generated sequences in the laboratory and measuring their properties.

Indeed, an experimental validation is needed, as the reliability of deep network oracles depends on several factors. The most challenging one is an out of distribution generalization. Specifically, deep network oracles learn to predict protein properties from examples of known proteins. Predictions are expected to be better when testing on instances from the same distribution as the training data, but current research is far from understanding if and when deep networks will be generalized to other inputs.[66] Because the goal in protein design is generating sequences that are different from the training set, generalization is a grave concern. Namely, the generative methods described in this review are optimized to find sequences in input space for which the oracle will assign

https://doi.org/10.1021/acs.jpcb.1c02449
J. Phys. Chem. B 2021, 125, 6440−6450

an optimal value, but these may be in regions of input space in which the oracle performs poorly. Listgarten and colleagues[42e,f] suggest addressing this issue by accounting for the uncertainty in the oracle's predictions while carrying out this optimization. Linder et al.[53a] suggest constraining the generated sequences to regions of sequence space where the oracle is more confident.

Several studies experimentally tested the sequences generated by deep networks. The exact experiment depends on the desired property of the protein. Repecka et al.[53d] designed sequences that have catalytic activity and show that in 24% of the sequences tested *in vitro* are both soluble and wild-type-level catalytic activity (even for generated sequences that differ from known ones by more than 100 positions). Strokach et al.[63] designed sequences that have predefined geometries that are similar to four CATH classes that were held out during the training of the network. For their most promising sequences, which passed several computational filters, they show that some have CD spectroscopy measurements that agree with the desired structural properties. Hawkins-Hooker et al.[42d] used VAE to generate sequences that are variants of luxA and experimentally validated their desired luminescence activity. Linder et al.[53a] generated novel GFP sequences and experimentally demonstrated state-of-the-art results. Thus, even though there are only a handful of studies that carried out an experimental validation, we believe that their results demonstrate the promise in using generative models for protein engineering.

SUMMARY

The exciting technology of deep learning is starting to make an impact on the design of novel proteins. Designing sequences that have a desired property requires addressing a multifaceted challenge: First, one needs to train an accurate oracle for this property. This typically requires that one defines the architecture and the loss function as well as a training data set. This oracle can rely on pretrained networks that were trained in a self-supervised manner to learn the properties of "protein-like" sequences. The oracle can then be used in innovative ways to efficiently sample or generate (possibly many different) sequences that have the desired property. Finally, evaluating whether these generated sequences have the desired property using experimental validation remains a challenge.

AUTHOR INFORMATION

Corresponding Author

Rachel Kolodny − *Department of Computer Science, Jacobs Building, University of Haifa, Haifa, Israel 3498838;* orcid.org/0000-0001-8523-1614; Email: trachel@cs.haifa.ac.il

Author

Margarita Osadchy − *Department of Computer Science, Jacobs Building, University of Haifa, Haifa, Israel 3498838*

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jpcb.1c02449

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

R.K.'s research was supported by the Israeli Science Foundation (ISF) (grant 450/16). M.O and R.K's research was supported by the Data Science Research Center at the University of Haifa.

REFERENCES

(1) (a) Yang, K. K.; Wu, Z.; Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. *Nat. Methods* **2019**, *16* (8), 687−694. (b) Huang, P.-S.; Boyken, S. E.; Baker, D. The coming of age of de novo protein design. *Nature* **2016**, *537* (7620), 320−327.

(2) (a) Gao, W.; Mahajan, S. P.; Sulam, J.; Gray, J. J. Deep Learning in Protein Structural Modeling and Design. *Patterns* **2020**, *1*, 100142. (b) Kell, D. B.; Samanta, S.; Swainston, N. Deep learning and generative methods in cheminformatics and chemical biology: navigating small molecule space intelligently. *Biochem. J.* **2020**, *477* (23), 4559−4580. (c) Lopez, R.; Gayoso, A.; Yosef, N. Enhancing scientific discoveries in molecular biology with deep generative models. *Mol. Syst. Biol.* **2020**, *16* (9), No. e9198. (d) Mazurenko, S.; Prokop, Z.; Damborsky, J. Machine Learning in Enzyme Engineering. *ACS Catal.* **2020**, *10* (2), 1210−1223. (e) Elton, D. C.; Boukouvalas, Z.; Fuge, M. D.; Chung, P. W. Deep learning for molecular design—a review of the state of the art. *Molecular Systems Design Engineering* **2019**, *4* (4), 828−849. (f) Wittmann, B. J.; Johnston, K. E.; Wu, Z.; Arnold, F. H. Advances in machine learning for directed evolution. *Curr. Opin. Struct. Biol.* **2021**, *69*, 11−18.

(3) (a) He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*; 2016; pp 770−778. (b) Masi, I.; Wu, Y.; Hassner, T.; Natarajan, P. Deep Face Recognition: A Survey. *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*; IEEE: 2018; pp 471−478. (c) Xiao, Y.; Tian, Z.; Yu, J.; Zhang, Y.; Liu, S.; Du, S.; Lan, X. A review of object detection based on deep learning. *Multimedia Tools Applications* **2020**, *79* (33), 23729−23791.

(4) (a) Graves, A.; Jaitly, N. Towards End-to-End Speech Recognition with Recurrent Neural Networks. *International Conference on Machine Learning*; PMLR: 2014; pp 1764−1772. (b) Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K., Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* **2018**.

(5) Elkahky, A. M.; Song, Y.; He, X. A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems. *Proceedings of the 24th International Conference on World Wide Web*; 2015; pp 278−288.

(6) (a) Yang, J.; Anishchenko, I.; Park, H.; Peng, Z.; Ovchinnikov, S.; Baker, D. Improved protein structure prediction using predicted interresidue orientations. *Proc. Natl. Acad. Sci. U. S. A.* **2020**, *117* (3), 1496−1503. (b) Senior, A. W.; Evans, R.; Jumper, J.; Kirkpatrick, J.; Sifre, L.; Green, T.; Qin, C.; ídek, A.; Nelson, A. W.; Bridgland, A. Improved protein structure prediction using potentials from deep learning. *Nature* **2020**, *577* (7792), 706−710. (c) Xu, J. Distance-based protein folding powered by deep learning. *Proc. Natl. Acad. Sci. U. S. A.* **2019**, *116* (34), 16856−16865.

(7) (a) Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529* (7587), 484−489. (b) Kisa'anin, B. Deep Learning for Autonomous Vehicles. *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*; IEEE: 2017; pp 142−142.

(8) Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* **2014**.

(9) Wu, Z.; Kan, S. J.; Lewis, R. D.; Wittmann, B. J.; Arnold, F. H. Machine learning-assisted directed protein evolution with combinatorial libraries. *Proc. Natl. Acad. Sci. U. S. A.* **2019**, *116* (18), 8852−8858.

(10) Hou, J.; Adhikari, B.; Cheng, J. DeepSF: deep convolutional neural network for mapping protein sequences to folds. *Bioinformatics* **2018**, *34* (8), 1295−1303.
(11) Riesselman, A. J.; Ingraham, J. B.; Marks, D. S. Deep generative models of genetic variation capture the effects of mutations. *Nat. Methods* **2018**, *15*, 816−822.
(12) Muller, A. T.; Hiss, J. A.; Schneider, G. Recurrent neural network model for constructive peptide design. *J. Chem. Inf. Model.* **2018**, *58* (2), 472−479.
(13) Soding, J. Protein homology detection by HMM-HMM comparison. *Bioinformatics* **2005**, *21* (7), 951−960.
(14) Finn, R. D.; Clements, J.; Eddy, S. R. HMMER web server: interactive sequence similarity searching. *Nucleic Acids Res.* **2011**, *39* (suppl_2), W29−W37.
(15) Leaver-Fay, A.; Tyka, M.; Lewis, S. M.; Lange, O. F.; Thompson, J.; Jacak, R.; Kaufman, K. W.; Renfrew, P. D.; Smith, C. A.; Sheffler, W. ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules. *Methods Enzymol.* **2011**, *487*, 545−574.
(16) Yang, K. K.; Wu, Z.; Bedbrook, C. N.; Arnold, F. H. Learned protein embeddings for machine learning. *Bioinformatics* **2018**, *34* (15), 2642−2648.
(17) (a) Alley, E. C.; Khimulya, G.; Biswas, S.; AlQuraishi, M.; Church, G. M. Unified rational protein engineering with sequence-based deep representation learning. *Nat. Methods* **2019**, *16* (12), 1315−1322. (b) Bileschi, M. L.; Belanger, D.; Bryant, D.; Sanderson, T.; Carter, B.; Sculley, D.; DePristo, M. A.; Colwell, L. J. Using deep learning to annotate the protein universe. *bioRxiv* **2019**, 626507.
(18) Service, R. F. The Game Has Changed. *AI Triumphs at Protein Folding*; American Association for the Advancement of Science: 2020.
(19) Liu, X. Deep recurrent neural network for protein function prediction from sequence. *arXiv preprint arXiv:.08318* **2017**.
(20) (a) Kulmanov, M.; Hoehndorf, R. DeepGOPlus: improved protein function prediction from sequence. *Bioinformatics* **2019**, *36* (2), 422−429. (b) Villegas-Morcillo, A.; Makrodimitris, S.; van Ham, R. C. H. J.; Gomez, A. M.; Sanchez, V.; Reinders, M. J. T. Unsupervised protein embeddings outperform hand-crafted sequence and structure features at predicting molecular function. *Bioinformatics* **2021**, *37* (2), 162−170. (c) Gligorijevic, V.; Renfrew, P. D.; Kosciolek, T.; Leman, J. K.; Berenberg, D.; Vatanen, T.; Chandler, C.; Taylor, B. C.; Fisk, I. M.; Vlamakis, H. Structure-based function prediction using graph convolutional networks. *Nat. Commun.* **2021**, *12*, 3168.
(21) Li, Y.; Wang, S.; Umarov, R.; Xie, B.; Fan, M.; Li, L.; Gao, X. DEEPre: sequence-based enzyme EC number prediction by deep learning. *Bioinformatics* **2018**, *34* (5), 760−769.
(22) Das, P.; Wadhawan, K.; Chang, O.; Sercu, T.; Santos, C. D.; Riemer, M.; Chenthamarakshan, V.; Padhi, I.; Mojsilovic, A. Pepcvae: Semi-supervised targeted design of antimicrobial peptide sequences. *arXiv preprint arXiv:.07743* **2018**.
(23) (a) Tubiana, J.; Cocco, S.; Monasson, R. Learning protein constitutive motifs from sequence data. *eLife* **2019**, *8*, No. e39397. (b) Rao, R.; Bhattacharya, N.; Thomas, N.; Duan, Y.; Chen, P.; Canny, J.; Abbeel, P.; Song, Y. Evaluating protein transfer learning with tape. *Advances in Neural Information Processing Systems*; **2019**; pp 9689−9701. (c) Karimi, M.; Zhu, S.; Cao, Y.; Shen, Y. De novo protein design for novel folds using guided conditional Wasserstein generative adversarial networks. *J. Chem. Inf. Model.* **2020**, *60*, 5667.
(24) Hochreiter, S.; Schmidhuber, J. J. N. c. Long short-term memory. *Neural Comp.* **1997**, *9* (8), 1735−1780.
(25) Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A.; Salakhudinov, R.; Zemel, R.; Bengio, Y. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *International Conference on Machine Learning*; PMLR: 2015; pp 2048−2057.
(26) Krause, B.; Lu, L.; Murray, I.; Renals, S. Multiplicative LSTM for sequence modelling. *arXiv preprint arXiv:.07959* **2016**.
(27) Oord, A. v. d.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:.03499* **2016**.
(28) Yu, F.; Koltun, V.; Funkhouser, T. Dilated residual networks. *Proceedings of the IEEE conference on computer vision and pattern recognition* **2017**, 472−480.
(29) (a) Heinzinger, M.; Elnaggar, A.; Wang, Y.; Dallago, C.; Nechaev, D.; Matthes, F.; Rost, B. Modeling the language of life−deep learning protein sequences. *BMC Bioinf.* **2019**, *20*, 723. (b) Rives, A.; Meier, J.; Sercu, T.; Goyal, S.; Lin, Z.; Liu, J.; Guo, D.; Ott, M.; Zitnick, C. L.; Ma, J. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proc. Natl. Acad. Sci.* **2021**, *118* (15), e2016239118.
(30) Asgari, E.; Mofrad, M. R. K. Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics. *PLoS One* **2015**, *10* (11), No. e0141287.
(31) Elnaggar, A.; Heinzinger, M.; Dallago, C.; Rihawi, G.; Wang, Y.; Jones, L.; Gibbs, T.; Feher, T.; Angerer, C.; Steinegger, M. ProtTrans: Towards Cracking the Language of Life's Code Through Self-Supervised Deep Learning and High Performance Computing. *arXiv preprint arXiv:2007.06225* **2020**.
(32) Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep contextualized word representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*; New Orleans, LA, 2018.
(33) Filipavicius, M.; Manica, M.; Cadow, J.; Martinez, M. R., Pre-training Protein Language Models with Label-Agnostic Binding Pairs Enhances Performance in Downstream Tasks. *arXiv preprint arXiv:.03084* **2020**.
(34) Bepler, T.; Berger, B., Learning protein sequence embeddings using information from structure. *arXiv preprint arXiv:1902.08661* **2019**.
(35) (a) Strodthoff, N.; Wagner, P.; Wenzel, M.; Samek, W. UDSMProt: universal deep sequence models for protein classification. *Bioinformatics* **2020**, *36* (8), 2401−2409. (b) Madani, A.; McCann, B.; Naik, N.; Keskar, N. S.; Anand, N.; Eguchi, R. R.; Huang, P.-S.; Socher, R. ProGen: Language Modeling for Protein Generation. *arXiv preprint arXiv:.03497* **2020**.
(36) Keskar, N. S.; McCann, B.; Varshney, L. R.; Xiong, C.; Socher, R., Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:.05858* **2019**.
(37) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I., Attention is all you need. *arXiv preprint arXiv:.03762* **2017**.
(38) Hinton, G. E.; Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science* **2006**, *313* (5786), 504−507.
(39) Ingraham, J.; Garg, V.; Barzilay, R.; Jaakkola, T. Generative models for graph-based protein design. *Advances in Neural Information Processing Systems*, **2019**; pp 15820−15831.
(40) Salakhutdinov, R.; Hinton, G. Deep Boltzmann Machines. *Artificial Intelligence and Statistics*; PMLR: 2009; pp 448−455.
(41) Kingma, D. P.; Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* **2013**.
(42) (a) Greener, J. G.; Moffat, L.; Jones, D. T. Design of metalloproteins and novel protein folds using variational autoencoders. *Sci. Rep.* **2018**, *8* (1), 16189. (b) Ding, X.; Zou, Z.; Brooks Iii, C. L. Deciphering protein evolution and fitness landscapes with latent space models. *Nat. Commun.* **2019**, *10* (1), 5644. (c) Sinai, S.; Kelsic, E.; Church, G. M.; Nowak, M. A. Variational auto-encoding of protein sequences. *arXiv preprint arXiv:.03346* **2017**. (d) Hawkins-Hooker, A.; Depardieu, F.; Baur, S.; Couairon, G.; Chen, A.; Bikard, D. Generating functional protein variants with variational autoencoders. *PLoS Comput. Biol.* **2021**, *17* (2), No. e1008736. (e) Brookes, D.; Park, H.; Listgarten, J. *Conditioning by Adaptive Sampling for Robust Design*; 2019; PMLR: pp 773−782. (f) Brookes, D. H.; Listgarten, J. Design by adaptive sampling. *arXiv preprint arXiv:1810.03714* **2018**.

https://doi.org/10.1021/acs.jpcb.1c02449
J. Phys. Chem. B 2021, 125, 6440−6450

(43) Rezende, D.; Mohamed, S. Variational Inference with Normalizing Flows. *International Conference on Machine Learning*; PMLR: 2015; pp 1530−1538.
(44) Dinh, L.; Krueger, D.; Bengio, Y. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516* **2014**.
(45) Dinh, L.; Sohl-Dickstein, J.; Bengio, S. Density estimation using real nvp. *arXiv preprint arXiv:.08803* **2016**.
(46) Kingma, D.; Salimans, T.; Josefowicz, R.; Chen, X.; Sutskever, I.; Welling, M. Improving variational autoencoders with inverse autoregressive flow. *arXiv preprint arXiv:1606.04934***2017**.
(47) Papamakarios, G.; Pavlakou, T.; Murray, I. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:.07057* **2017**.
(48) Kingma, D. P.; Dhariwal, P. Glow: Generative flow with invertible 1 × 1 convolutions. *arXiv preprint arXiv:.03039* **2018**.
(49) Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein Generative Adversarial Networks. *International Conference on Machine Learning*; PMLR: 2017; pp 214−223.
(50) Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A. Improved training of wasserstein gans. *arXiv preprint arXiv:.00028* **2007**.
(51) (a) Mertikopoulos, P.; Lecouat, B.; Zenati, H.; Foo, C.-S.; Chandrasekhar, V.; Piliouras, G. Optimistic mirror descent in saddle-point problems: Going the extra (gradient) mile. *arXiv preprint arXiv:.02629* **2018**. (b) Adolphs, L.; Daneshmand, H.; Lucchi, A.; Hofmann, T. Local Saddle Point Optimization: A Curvature Exploitation Approach. *The 22nd International Conference on Artificial Intelligence and Statistics*; PMLR: 2019; pp 486−495. (c) Daskalakis, C.; Ilyas, A.; Syrgkanis, V.; Zeng, H. Training gans with optimizm. *arXiv preprint arXiv:.00141* **2017**.
(52) Karras, T.; Aila, T.; Laine, S.; Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:.10196* **2017**.
(53) (a) Linder, J.; Bogard, N.; Rosenberg, A. B.; Seelig, G. A Generative Neural Network for Maximizing Fitness and Diversity of Synthetic DNA and Protein Sequences. *Cell Systems* **2020**, *11* (1), 49−62. (b) Killoran, N.; Lee, L. J.; Delong, A.; Duvenaud, D.; Frey, B. J. Generating and designing DNA with deep generative models. *arXiv preprint arXiv:.06148* **2017**. (c) Gupta, A.; Zou, J. Feedback GAN for DNA optimizes protein functions. *Nature Machine Intelligence* **2019**, *1* (2), 105−111. (d) Repecka, D.; Jauniskis, V.; Karpus, L.; Rembeza, E.; Rokaitis, I.; Zrimec, J.; Poviloniene, S.; Laurynenas, A.; Viknander, S.; Abuajwa, W. Expanding functional protein sequence spaces using generative adversarial networks. *Nature Machine Intelligence* **2021**, *3*, 324.
(54) (a) Anishchenko, I.; Chidyausiku, T. M.; Ovchinnikov, S.; Pellock, S. J.; Baker, D. De novo protein design by deep network hallucination. *bioRxiv* **2020**, DOI: 10.1101/2020.07.22.211482 (b) Biswas, S.; Khimulya, G.; Alley, E. C.; Esvelt, K. M.; Church, G. M. Low-N protein engineering with data-efficient deep learning. *Nat. Methods* **2021**, *18* (4), 389−396.
(55) Keefe, A. D.; Szostak, J. W. Functional proteins from a random-sequence library. *Nature* **2001**, *410* (6829), 715.
(56) Simonyan, K.; Vedaldi, A.; Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:.00028* **2013**.
(57) Costello, Z.; Martin, H. G. How to hallucinate functional proteins. *arXiv preprint arXiv:.00458* **2019**.
(58) Chen, X.; Kingma, D. P.; Salimans, T.; Duan, Y.; Dhariwal, P.; Schulman, J.; Sutskever, I.; Abbeel, P. Variational lossy autoencoder. *arXiv preprint arXiv:.02731* **2016**.
(59) Schölkopf, B.; Smola, A.; Müller, K.-R. Kernel Principal Component Analysis. *International Conference on Artificial Neural Networks*; Springer: 1997; pp 583−588.
(60) Sohn, K.; Lee, H.; Yan, X. Learning structured output representation using deep conditional generative models. *Advances in Neural Information Processing Systems* **2015**, *28*, 3483−3491.
(61) Taylor, W. R. A 'periodic table'for protein structures. *Nature* **2002**, *416* (6881), 657−660.
(62) Fan, A.; Lewis, M.; Dauphin, Y. Hierarchical neural story generation. *arXiv preprint arXiv:.04833* **2018**.
(63) Strokach, A.; Becerra, D.; Corbi-Verge, C.; Perez-Riba, A.; Kim, P. M. Fast and flexible protein design using deep graph neural networks. *Cell Systems* **2020**, *11* (4), 402−411.
(64) Kosloff, M.; Kolodny, R. Sequence-similar, structure-dissimilar protein pairs in the PDB. *Proteins: Struct., Funct., Genet.* **2008**, *71* (2), 891−902.
(65) O'Connell, J.; Li, Z.; Hanson, J.; Heffernan, R.; Lyons, J.; Paliwal, K.; Dehzangi, A.; Yang, Y.; Zhou, Y. SPIN2: Predicting sequence profiles from protein structures using deep neural networks. *Proteins: Struct., Funct., Genet.* **2018**, *86* (6), 629−633.
(66) Xu, K.; Zhang, M.; Li, J.; Du, S. S.; Kawarabayashi, K.-i.; Jegelka, S. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848* **2020**.