# Learning Reduced Latent Representations of Protein Structure Data

Fardina Fathmiul Alam
Dept of Computer Science
George Mason University
falam5@gmu.edu

Taseef Rahman
Dept of Computer Science
George Mason University
trahman2@gmu.edu

Amarda Shehu*
Dept of Computer Science
George Mason University
amarda@gmu.edu

## ABSTRACT

The protein modeling community has long been interested in dimensionality reduction of structure data. Motivated by rapid progress in neural network research, we investigate autoencoders of various architectures on reducing the dimensionality of protein structure data generated by template-free protein structure prediction methods. We show that autoencoders that model nonlinear relationships among variables outperform linear dimensionality reduction. We evaluate various architectures and propose a better-performing one. We further show that the learned, low-dimensional latent representations capture inherent information useful for structure prediction. Given the ease with which open-source neural network libraries, such as Keras, which we employ here, allow constructing, training, and evaluating neural networks, we believe that autoencoders will gain in popularity in the structure biology community and open up further avenues of research.

## CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Applied computing** → **Molecular structural biology**; **Bioinformatics**.

## KEYWORDS

protein structure; latent representation; dimensionality reduction; autoencoders; deep neural networks.

**ACM Reference Format:**
Fardina Fathmiul Alam, Taseef Rahman, and Amarda Shehu. 2019. Learning Reduced Latent Representations of Protein Structure Data. In *10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (ACM-BCB '19), September 7–10, 2019, Niagara Falls, NY, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3307339.3343866

## 1 INTRODUCTION

There is now a great demand for machine learning (ML) methods to handle, summarize, and make observations from growing molecular structure data [6, 21–23, 25, 33]. In particular, the protein modeling community has long been interested in reducing the dimensionality of protein structure data [1]. One driving objective has been to highlight functionally-related collective motions of groups of atoms [12, 14, 24, 25]. Another objective has been to visualize structure spaces and make structure-function discoveries [21, 23, 29], as well as expedite the search for novel structures and structural transitions [9, 10, 13, 18, 19, 27, 28, 30]. Predominantly, linear methods, such as Principal Component Analysis (PCA) [31], have been favored due to ease of implementation and evaluation. Some work has considered non-linear methods, such as Isomap [32], Locally Linear Embedding [26], Diffusion Maps [11], and others [34], but these have not been as broadly adopted, mainly due to difficulty of evaluation and/or implementation.

Motivated by rapid progress in neural network research, we investigate autoencoders (AEs) of various architectures on reducing the dimensionality of protein structure data. To the best of our knowledge, the first AE occurrence can be found in Ref. [5], where an AE is applied to a substituted cyclo-octane of 24 atoms. In more recent work [17], an AE with several hidden layers in the encoder and a novel cost function is applied to molecular dynamics simulation data obtained for Asp7, a small molecule of 12 backbone dihedral angles, and Trp-cage, a small protein of 20 amino acids. Work in [7] investigates a similar AE architecture to summarize the Trp-Cage folding landscape.

In this paper, we conduct a systematic, quantitative evaluation of shallow and deep AEs. We focus on structure data generated by template-free protein structure prediction (PSP) platforms due to the ease with which tertiary structures can be generated. These structures, are also referred to as decoys, as they hide among them the biologically-active/native structure. Moreover, we focus on small- to medium-size proteins that are beyond the reach of molecular dynamics simulation. These molecules consist of thousands of atoms. On such data, we demonstrate that deep AEs allows modeling complex, nonlinear relationships among variables and outperform linear dimensionality reduction models. However, we also demonstrate that higher depth does not necessarily translate to better performance. A proof-of-concept evaluation additionally demonstrates how learned latent representations capture inherent structure-function information that can be useful for important learning tasks in template-free PSP.

The rest of this paper is organized as follows. Section 2 describes AE architectures and relates details regarding training and evaluation. Section 3 presents a detailed comparative evaluation against PCA on decoy data over a benchmark set of protein targets often used by decoy generation algorithms. This section also shows the ability of the AE-learned representations to predict proximity of decoys to the native structure. Section 4 concludes the paper with a summary and discussion of future work.

*Corresponding Author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ACM-BCB '19, September 7–10, 2019, Niagara Falls, NY, USA*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6666-3/19/09…$15.00
https://doi.org/10.1145/3307339.3343866

## 2 METHODS

We first summarize the architecture of a traditional/vanilla AE before demonstrating how more complex AEs can be obtained and relating details on training and evaluation.

### 2.1 Shallow versus Deep AE Architectures

An AE contains an *encoder* and a *decoder*. Fig. 1(a) relates a vanilla AE (vAE), where the encoder *maps* the input layer $x$ to the layer $y$, and the decoder maps the same layer $y$ to the output layer $z$. The sought latent representation is the layer $y$. The *encoder* is a deterministic mapping $f_\theta$ parameterized by a vector of parameters $\theta = \{W, b\}$ that transforms an input vector $x$ into a hidden representation/code vector $y$. Learning $y$ is the objective. Typically, $f_\theta$ is an affine mapping that can also be followed by a nonlinearity: $f_\theta(x) = \sigma(W \cdot x + b)$. Here, $\sigma$ refers to the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, and $W$ and $b$ are the weights and biases, respectively, that connect the input units (elements of $x$) to the hidden units (elements of $y$). Note that $\sigma$ is a specific activation function. Others can be used, such as tanh, rectified linear unit (RELU), or leaky RELU (LR). RELU returns $max(0, z)$ (zeroing out negative units). To remedy premature convergence of units to zeroes, LR replaces the zero region with a slope whose coefficient can be specified by the user. The *decoder* maps $y$ to a vector $z = g_{\theta'}(y)$, where $\theta^{'} = \{W^{'}, b^{'}\}$, where $W^{'}$ and $b^{'}$ are the weights and biases, and $g_{\theta'}$ is an affine mapping followed (or not) by nonlinearity. As $z$ is the reconstructed version of $x$, $|z| = |x|$. Typically, one seeks a *reduced* representation $y$ such that $|y| < |x|$; this setting is known as under-complete. Nothing disallows $|y| > |x|$ (over-complete setting).

Many theoretical arguments point in favor of composing several levels of nonlinearity as key to efficiently modeling complex relationships between variables and achieving better generalization performance on challenging recognition tasks [3]. One can do so in two different ways, by stacking separate encoder-decoder pairs, or by adding various hidden layers to the encoder and decoder in one encoder-decoder pair. Specifically, stacked AEs (sAEs), shown in Fig. 1(b), are one type of deep AEs that stack vAEs [35]. Typically, one abuses notation and refers to the vAEs as layers. The learned hidden $z$ of one layer becomes the input of the next layer.

Fig. 1(b) illustrates this as follows. Let us assume that the dimensionality of an input instance $x$ is 5. The first vAE/layer can be summarized as $L^{(1)} = \{x \rightarrow y_1 \rightarrow z_1\}$, with the subscript 1 indicating that this is the first layer. In this layer, Fig. 1(b) illustrates that $|y_1| = 4$, and $|z_1| = 5$. The second layer can be summarized as $L^{(2)} = \{y^{(1)} \rightarrow y^{(2)} \rightarrow z^{(2)}\}$. In this layer, the input is not $x$ but the learned vector $y^{(1)}$ obtained from the previous layer. Fig. 1(b) illustrates that $|y^{(1)}| = 4$, $|y^{(2)}| = 3$, and $|z^{(2)}| = 4$. A third layer has been shown in Fig. 1(b) that can be summarized as $L^{(3)} = \{y^{(2)} \rightarrow y^{(3)} \rightarrow z^{(3)}\}$, where $|y^{(2)}| = 3$, $|y^{(3)}| = 2$, and $|z^{(3)}| = 3$. The need to know $y^{(i)}$ learned from layer $L^{(i)}$ to serve as input for layer $L^{(i+1)}$ necessitates that each layer be trained separately and in order.

Alternatively, an AE capable of learning complex, non-linear relationships can be constructed not via stacking, but by adding more hidden layers in the encoder and decoder. Fig. 1(c) provides an illustration. In this architecture, the encoder makes use of several hidden layers. The first hidden layer can be over-complete, whereas the others gradually reduce the number of units until the target dimensionality is reached. The layer where the number of units is the same as the target dimensionality is the input to the decoder, which mirrors the encoder (in reverse) in the composition of hidden layers. To distinguish this architecture from the above, we will refer to this architecture as dAE.

AEs provide a general dimensionality reduction framework. When using affine encoder and decoder without any nonlinearity (and a squared error loss, which we detail below), the resulting AE essentially performs PCA [2]. We demonstrate this in Section 3. The employment of nonlinearity in the encoder allows carrying out nonlinear dimensionality reduction.

### 2.2 AE Training

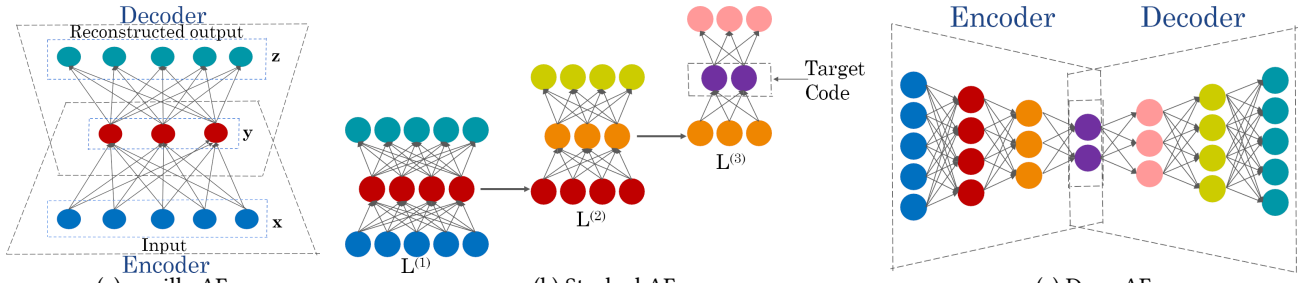
The training of an AE is guided by a loss function that measures how different the input $x$ and its reconstructed version $z$ are from each-other. The loss function is also referred to as the reconstruction error. For real-valued $x \in [0, 1]^d$ (of dimensionality $d$), it is more natural for the loss to be measured as the squared error $||x - z||^2$ [35]. Note that in an sAE, each layer $L^{(i)}$ is guided by its own reconstruction error $||y^{(i-1)} - z^i||^2$ (with $x$ serving as $y^0$ for the first layer). For real-valued $x$, due to the Gaussian interpretation of $z$, it is also more natural not to use nonlinearity in the decoder, though it has become common practice to do so. Nonlinearity in the decoder is justified when dealing with binary data $x$, as the decoder needs to produce $z \in [0, 1]^d$. Doing so yields a loss function that is the cross-entropy between two independent multivariate Bernoullis.

The parameters $\theta$ and $\theta^{'}$ are learned in a gradient descent process whose objective is the minimization of the loss function. We make use of the Adam optimizer due to its demonstrated superiority [16]. A learning rate controls how much the weights are adjusted with respect the negative gradient of the loss function. A lower learning rate may stall premature convergence into local minima. Typically, one starts with a low learning rate that increases gradually to expedite convergence. Since the loss function may be complex, its optimization proceeds in epochs. In each epoch, the training data are passed forward to compute the current value of the loss function. The negative gradient of the loss function is evaluated at this value and then passed backwards to update the weights and biases. In each epoch, the training data is divided into batches, where the batch size is the number of input instances passed forward before the parameters are updated.

One of the choices during training is the employment of dropout. Dropout refers to ignoring a randomly-selected subset of units/elements in the input during training. Ignoring means that selected units are not considered (zeroed out) during the forward and backward passes. The reason for dropout is to prevent overfitting. Another consequence of dropout is a reduced training time per epoch but an increase in the time required to converge.

### 2.3 Investigated AE Architectures

Since the protein structure data we utilize are specified as Cartesian coordinates, the inputs $x$ are real-valued. We have experimented with various AE architectures, shallow, stacked, and deep ones, where encoders and decoders employ nonlinearity or not. Since




(a) vanilla AE (b) Stacked AE (c) Deep AE

**Figure 1: (a) A traditional/vanilla AE, where the he encoder maps an input vector $x$ to a hidden vector $y$, and the decoder maps $y$ to an output vector $z$, a reconstructed version of $x$. The training process that determines the parameters of the encoder and decoder (weights and biases) aims to minimize the loss function. For real-valued $x$, this is typically the squared error $||x - z||^2$. (b) vAEs can be stacked to obtain an sAE. Each vAE can be considered a layer. The learned, hidden $y$ of one layer (one vAE) becomes the input of the next layer (the next vAE). (c) In this deep architecture, the encoder and decoder make use of several hidden layers. The decoder mirrors the encoder, performing the reverse mapping.**

the space of possible architectures can be vast, we restrict our investigation to the following. First, we make use of a vAE with only affine mappings in the encoder and decoder. For instance, for a dataset of 222 dimensions (corresponding to Cartesian coordinates), the encoder maps an input layer of 222 units to the code layer of 2 units, and the decoder then maps the code layer of 2 units to an output layer of 222 units. Abusing notation in the interest of convenience, we refer to this vAE as $222 \rightarrow 2 \rightarrow 222$.

We also experiment with various sAE architectures. At one extreme, we consider an 8-layer sAE: $L^{(1)} = \{|x| \rightarrow m \rightarrow |x|\}$, $L^{(2)} = \{m \rightarrow 125 \rightarrow m\}$, $L^{(3)} = \{125 \rightarrow 64 \rightarrow 125\}$, $L^{(4)} = \{64 \rightarrow 32 \rightarrow 64\}$, $L^{(5)} = \{32 \rightarrow 16 \rightarrow 32\}$, $L^{(6)} = \{16 \rightarrow 8 \rightarrow 16\}$, $L^{(7)} = \{8 \rightarrow 4 \rightarrow 8\}$, and $L^{(8)} = \{4 \rightarrow 2 \rightarrow 4\}$, where $m > |x|$ depends on input dimensionality. When $x \leq 222$, $m = 250$; when $|x| > 250$, $m = |x| + 28$. The overall principle is to increase the dimensionality over $|x|$. The model can experience a loss of information when going from real-valued input data to values in $[0, 1]$ (due to nonlinearity in the encoder), which can be counterbalanced by an initial increase in dimensionality [15]. On the other extreme, we consider an 3-layer sAE, $L^{(1)} = \{|x| \rightarrow m \rightarrow |x|\}$, $L^{(2)} = \{m \rightarrow 125 \rightarrow m\}$, and $L^{(3)} = \{125 \rightarrow 2 \rightarrow 125.\}$

Finally, we consider various dAEs. At one extreme, we consider a dAE, where the encoder is the mapping $|x| \rightarrow m \rightarrow 125 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2$ (and the decoder does the reverse). At the other end, we consider a dAE, where the encoder maps $|x| \rightarrow m \rightarrow 125 \rightarrow 2$ (and the decoder does the reverse). We also consider various choices in activation functions (affine mappings, sigmoid, or LR) and dropout (or not).

It is worth highlighting that model depth comes at a great cost (number of weights and biases) that have to be learned. While data size in the structure biology community has increased, it cannot approach the million regime for image data, where deep learning is shown to be superior. Back-of-the-envelope calculations make this point. For an sAE with 8 layers as described above, the number of weights that have to be learned when $|x| = 222$ is $2 \times (222 \times 250 + 250 \times 125 + 125 \times 64 + 64 \times 32 + 32 \times 16 + 16 \times 8 + 8 \times 4 + 4 \times 2) = 194,956$; the factor of 2 at the beginning considers the encoder and decoder in each layer. Tying the weights of the decoder to those of the encoder (effectively learning only the weights of the encoder and transposing them to obtain the weights of the decoder) brings this number down to close to 100K. In a dAE, the additions are all turned into multiplications, and the number of weights is $2 \times (222 \times 250 \times 125 \times 64 \times 32 \times 16 \times 4 \times 2) = 2.9097984e+13$. Such large numbers of weights ensure that training converges prematurely to a local minimum in a very high-dimensional loss space.

## 2.4 Evaluation and Implementation Details

The trained models are compared to one another and PCA on MSE on the testing dataset in terms of mean squared error (MSE); the squared error is measured over every instance in the testing dataset, and the mean of these values is reported. PCA is used as a baseline due to its popularity. For fair comparison, the PCA model is trained over the same training data as the AEs and is evaluated over the same testing data in terms of MSE. We make use of Python's sklearn library to do so. After creating a PCA object, the object/model is trained over the training data $x$ using `pca.fit(x)`. Any data instance $D$ can be transformed/projected over the number of dimensions of interest using $D^{'}$ `= pca.transform(`$D$`)` afterwards; $D^{'}$ `refers to the transformed data.` The transformed data $D^{'}$ can be mapped back to obtain the *reconstructed* data $D^{''}$ at some dimensionality of interest $d$ using $D^{''}$ `= pca.inverse_transform(`$D^{'}$`,` $d$`)`. The reconstruction error is then measured as $||D - D^{''}||^2$, and the MSE over the data instances in the testing dataset is reported to evaluate PCA in comparison to the sAE models. Each AE model can converge to a different local minimum of the loss function, as the optimization process depends on initial values of the parameters, which are set at random, as is common practice. Therefore, each AE model is trained 3 times (starting with random initial parameters), resulting in 3 trained models. When comparing the MSE of a particular architecture to the MSE of PCA, we report the mean and variance of the MSEs obtained over the 3 runs.

The comparative evaluation highlights a model that achieves the lowest overall MSE on a 2D latent representation. The learned representation is evaluated visually and quantitatively. First, datasets are projected onto this representation and visualized in comparison to projections over the PCA-learned representations. Second, the learned representation is evaluated for its ability to encode "nativeness" in a regression setting, where the goal is to predict the least

root-mean-squared deviation (lRMSD) [20] of each decoy from the native structure from the learned representation. Two settings are considered, linear versus non-linear regression. In the latter, the sigmoid function is used to return values in $[0, 1]$; all lRMSD values are scaled in the same range. Each model is trained over the same training dataset and evaluated over the same testing dataset as the AEs. Model quality is evaluated on the MSE between predicted and given lRMSDs over the testing dataset.

The implementation, training, and evaluation of the various AEs is carried out in Keras [8], an open-source neural-network library written in Python. In the models with dropout during training, dropout rates of $0.1 - 0.4$, are evaluated; The increase in dropout pressures against overfitting. Each of the investigated AEs is trained for a total of 100 epochs with a batch size of 256. Learning rates vary from 0.009 to 0.0005 (varied per layer on sAEs). When LR is employed, the negative slope coefficient $\alpha$ is set to 0.3. Training times for all considered AE models vary from 326.479 seconds to 1076.575 seconds depending on the size of the training dataset.

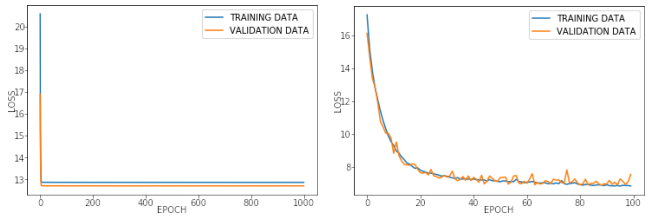
We employ 19 proteins of varying lengths (53 to 146 amino acids long) and folds ($\alpha$, $\beta$, $\alpha + \beta$, and *coil*). These proteins are typically employed to evaluate debuting decoy generation algorithms for template-free PSP [36, 37]. On each protein, we run the Rosetta AbInitio protocol in an embarrassingly parallel manner to obtain an ensemble of $50,000 - 60,000$ decoys per protein. Each decoy is a tertiary structure specified via the Cartesian coordinates of its atoms. We simplify each decoy by only maintaining the main carbon (CA) atom of each amino acid. Since the proteins we employ vary in lengths from 53 to 146 amino acids, input instances vary in dimensionality from $159 = 53 \times 3$ to $438 = 146 \times 3$. The decoy ensemble of each protein is split to obtain a training, validation, and testing dataset. A 0.5:0.1:0.4 split yields the training, validation, and testing datasets, respectively. The known native structure of each protein is obtained from the Protein Data Bank (PDB) [4].

## 3 RESULTS

In the interest of space, we only only report the performance of a subset of the (better-performing) AE models constructed and trained, omitting the very deep, underperforming architectures. We report on the performance of a vAE with affine mappings in the encoder and decoder, and a vAE with $\sigma$ in the encoder ($\text{vAE}_{\sigma_E}$). We evaluate an sAE with three layers $L^{(1)} = \{|x| \rightarrow m \rightarrow |x|\}$, $L^{(2)} = \{m \rightarrow 125 \rightarrow m\}$, and $L^{(3)} = \{125 \rightarrow 2 \rightarrow 125\}$, and $\sigma$ in the encoder layers, referring to it as $\text{sAE}_{\sigma_E}$. These models are compared to a dAE architecture, where the encoder maps $|x| \rightarrow m \rightarrow 125 \rightarrow 2$, and the decoder does the reverse. We report on two variations, $\text{dAE}_{\sigma_E, \sigma_D}$, where $\sigma$ is used in the encoder and decoder, and $\text{dAE}_{LR_E, \sigma_D}$, where the encoder uses LR instead. Each model is trained with or without dropout. The better performing models are those trained with dropout.

### 3.1 Comparative Evaluation

To substantiate the lack of overfitting, we relate the per-epoch training versus validation loss (on one selected protein) for vAE and $\text{dAE}_{LR_E, \sigma_D}$ in Fig. 2. Convergence is reached later in the dAE than the vAE model due to the complexity of the loss surface (which has more dimensions due to the higher number of parameters). The other AE models also indicate no overfitting (data not shown). We note that on sAEs, this analysis is done for every layer, as each sAE layer is trained separately.




vAE $\qquad$ $\text{dAE}_{LR_E, \sigma_D}$

**Figure 2: Training versus validation loss is shown over epochs. The shallow model converges faster.**

Fig. 3 compares mean MSEs (over 3 runs) of various AE models (related above) in comparison to PCA over the testing dataset. Fig. 3 shows that vAE obtains similar MSEs as PCA, and that $\text{vAE}_{\sigma_E}$ does not yield superior performance. The performance of $\text{sAE}_{\sigma_E}$ is the poorest, which is due to the fact that each subsequent layer considers a problem of lower dimensionality; the representations learned at each layer and associated reconstruction errors do not necessarily lead to an optimal overall representation and overall reconstruction error. Effectively, each layer makes a greedy decision that leads to an overall poor representation and high reconstruction error. In contrast, $\text{dAE}_{LR_E, \sigma_D}$ and $\text{dAE}_{\sigma_E, \sigma_D}$ are the best performing ones, with $\text{dAE}_{\sigma_E, \sigma_D}$ achieving the lowest mean MSEs. It is worth noting that the variances obtained for each model are $\leq 1e-3$; the only exceptions are $\text{sAE}_{\sigma_E}$ on 1sap (variance of 0.018), $\text{dAE}_{LR_E, \sigma_D}$ on 2h5n(D) (variance of 0.02), and $\text{dAE}_{\sigma_E, \sigma_D}$ on 2ci2 and 2h5n(D) (variances of 0.026 and 0.059, respectively).

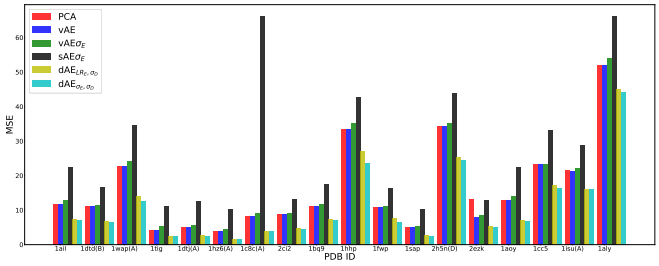



**Figure 3: The AE architectures are trained 3 times, starting with initial random weights and biases, and the mean MSEs obtained over the runs are reported. Each protein is identified with the PDB id of its known native structure. The chain is shown in parenthesis.**

### 3.2 Visualization of Latent Space

We visualize the 2D latent space in which PCA, vAE, and $\text{dAE}_{LR_E, \sigma_D}$ project the decoy datasets. We do so for the protein with native structure under PDB id 1aoy, plotting all embedded Rosetta-generated decoys as disks color-coded by their least RMSD from the native structure. A blue-to-red color scheme indicates lower-to-higher RMSDs. Fig. 4(a) shows the PCA-learned 2D embedding. Fig. 4(b)-(c) do so for the vAE-learned embedding from two different runs,

each one initialized with random parameter values. Fig. 4(d)-(f) relates the embedding learned from three different runs, each one initialized with random parameter values, of the best-performing dAE (in terms of MSEs), $\text{dAE}_{LR_E,\sigma_E}$ model.

Fig. 4(b)-(c) makes it clear that the models can be different and depend on the initialization of parameters. The embeddings are similar to the PCA-obtained one in Fig. 4(a) (within a rotation around $z$ axis), visually indicating that the vAE model indeed reproduces PCA. Fig. 4(d)-(f) show similar embeddings obtained by three different runs of $\text{dAE}_{LR_E,\sigma_D}$. The embeddings related in Fig. 4(d)-(f) are qualitatively more useful, as they either co-localize low lRMSD decoys better or separate them better in the latent space, holding more promise for further learning from the latent space.

### 3.3 Nativeness Encoding in the Latent Space

The above visualizations are a qualitative evaluation of the information present in the learned latent representations. We carry out a quantitative, proof-of-concept evaluation of whether the learned representations encode in them proximity to the native structure. We do so in a comparative setting for the 2D representations learned by PCA and the top-performing dAE (on mean MSE), $\text{dAE}_{\sigma_E,\sigma_D}$. In each case, we train a linear regression model and a perceptron (which adds $\sigma$) over the same training dataset over which we have trained PCA and $\text{dAE}_{\sigma_E,\sigma_D}$ to predict decoy lRMSDs from the native structure. Fig. 5 evaluates the regression and perceptron models on their mean MSE and variance over the testing datasets. In the case of the perceptron, since $\sigma$ maps the input to an output in $[0, 1]$, we scale lRMSDs to the same range. We note that regression variance is the coefficient of determination, so higher values indicate better performance. Low values indicate that the dependent variable cannot be predicted from the independent variable.

The results in Fig. 5 show that the PCA- and dAE-learned representations encode in them sufficient information about nativeness. There are no significant differences between the PCA- versus dAE-learned representations regarding MSE, but dAE-learned representations yield higher regression variance. Nonlinearity in the prediction model (perceptron) also improves performance over linear regression.

## 4 CONCLUSION

Motivated by rapid progress in neural network research, this paper has investigated and evaluated several shallow and deep AE architectures on reducing the dimensionality of protein structure data generated by template-free protein structure prediction methods. Nonlinear models are reported to perform best. A proof-of-concept experiment suggests that the obtained latent representations and so hold useful information for discriminating decoys in important tasks in template-free PSP, such as decoy selection.

The proposed work opens up many lines of future investigation. One can broaden the model search by considering hyperparameters, such as the learning rate, the negative slope coefficient in parametric RELU, etc. One can also evaluate the utility of the learned representations in unsupervised and supervised learning tasks related to decoy selection in PSP or other settings, where an understanding of the relationship between structure and function may be aided by reduced representations of structure.

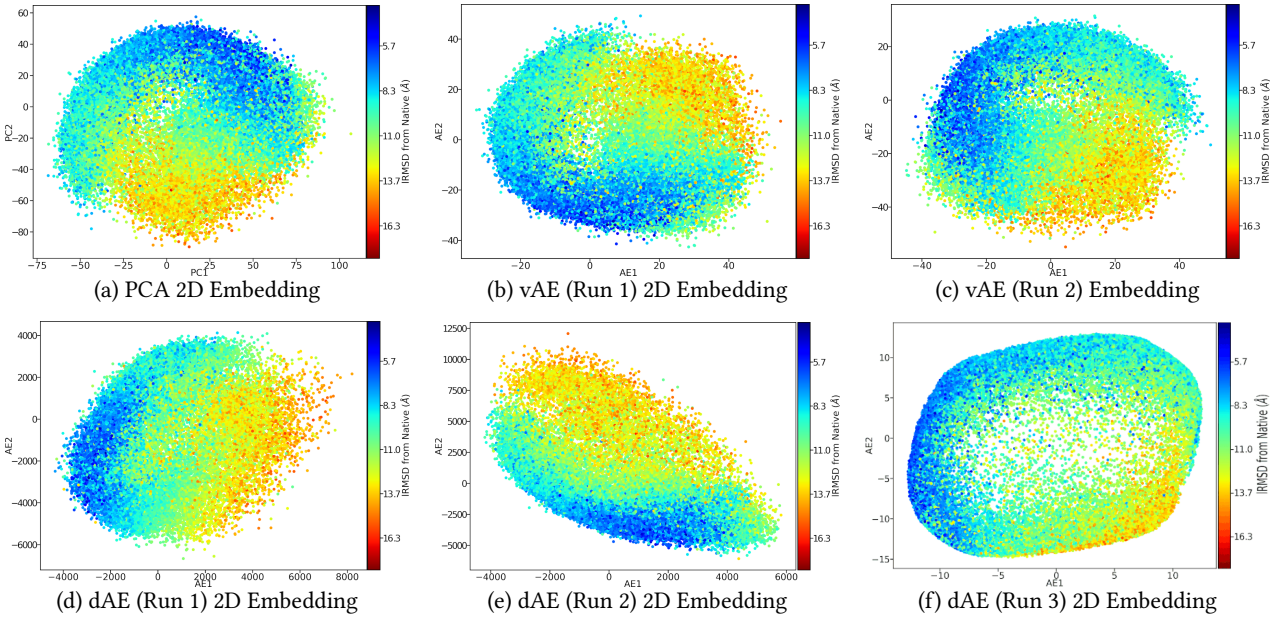
The presented work shows that training AEs is fraught with challenges. Great care has to be taken to evaluate whether a learned model converges to a local minimum of the loss function. Deep architectures result in high-dimensional loss surfaces, where gradient-based search is likely to converge prematurely to a local minimum near the starting point (the initial parameter values). As we have demonstrated, models have to be trained several times. Altogether, we believe that AEs hold much promise for structure data reduction and summarization. The ease of implementation and evaluation in platforms, such as Keras and others, will help wide adoption of AEs in the structure biology community.

## 5 ACKNOWLEDGMENTS

This work is supported in part by NSF IIS Grant No. 1763233, NSF DMS Grant No. 1821154, and a Jeffress Trust Award. Computations were run on ARGO, a research computing cluster provided by the Office of Research Computing at George Mason University, VA (URL: http://orc.gmu.edu).

## REFERENCES

[1] A. Amadei, A. B. Linssen, and H. J. Berendsen. 1993. Essential dynamics of proteins. *Proteins* 17, 4 (1993), 412–425.
[2] P. Baldi and K. Hornik. 1989. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks* 2 (1989), 53–58.
[3] Y. Bengio. 2009. Learning Dee Architectures for AI. *Foundations and Trends in Machine Learning* 2, 1 (2009), 1–127.
[4] H. M. Berman, K. Henrick, and H. Nakamura. 2003. Announcing the worldwide Protein Data Bank. *Nat. Struct. Biol.* 10, 12 (2003), 980–980.
[5] W. M. Brown, S. Martin, S. N. Pollock, E. A. Coutsias, and J. P. Watson. 2008. Algorithmic dimensionality reduction for molecular structure analysis. *J Chem Phys* 129, 6 (2008), 064118.
[6] I. Budowksi-Tal, Y. Nov, and R. Kolodny. 2010. FragBag, an accurate representation of protein structure, retrieves structural neighbors from the entire PDB quickly and accurately. *Proc Natl Acad Sci USA* 107, 8 (2010), 3481–3486.
[7] W. Chen, A. R. Tan, and A. L. Ferguson. 2018. Collective variable discovery and enhanced sampling using autoencoders: Innovations in network architecture and error function design. *J Chem Phys* 149 (2018), 072312.
[8] François Chollet et al. 2015. Keras. https://keras.io.
[9] R. Clausen, B. Ma, R. Nussinov, and A. Shehu. 2015. Mapping the Conformation Space of Wildtype and Mutant H-Ras with a Memetic, Cellular, and Multiscale Evolutionary Algorithm. *PLoS Comput Biol* 11, 9 (2015), e1004470.
[10] R. Clausen and A. Shehu. 2015. A Data-driven Evolutionary Algorithm for Mapping Multi-basin Protein Energy Landscapes. *J Comp Biol* 22, 9 (2015), 844–860.
[11] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker. 2005. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proc Natl Acad Sci USA* 102, 21 (2005), 7426–7431.
[12] P. Das, M. Moll, H. Stamati, L. E. Kavraki, and C. Clementi. 2006. Low-dimensional free energy landscapes of protein folding reactions by nonlinear dimensionality reduction. *Proc. Natl. Acad. Sci. USA* 103, 26 (2006), 9885–9890.
[13] G. Fiorin, M. L. Klein, and J. Hénin. 2013. Using collective variables to drive molecular dynamics simulations. *Intl J Interface Chem Phys* 111, 22-23 (2013), 3345–3362.
[14] B. J. Grant, A. A. Gorfe, and J. A. McCammon. 2010. Large conformational changes in proteins: signaling and other functions. *Curr. Opinion Struct. Biol.* 20, 2 (2010), 142–147. https://doi.org/10.1016/j.sbi.2009.12.004
[15] G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313, 5786 (2006), 504–507.
[16] D. Kingma and J. Ba. 2015. Adam: A Method for Stochastic Optimization. In *Intl Conf Learning Representations (ICLR)*. IEEE Press, 1–15.
[17] T. Lemke and C. Peter. 2019. EncoderMap: Dimensionality Reduction and Generation of Molecule Conformations. *J Chem Theory Comput* 15, 2 (2019), 1209–1215.
[18] T. Maximova, E. Plaku, and A. Shehu. 2018. Structure-guided Protein Transition Modeling with a Probabilistic Roadmap Algorithm. *IEEE/ACM Trans Comput Biol and Bioinf* 15, 6 (2018), 1783–1796.
[19] T. Maximova, ZQi. Zhao, D. B. Carr, E. Plaku, and A. Shehu. 2017. Sample-based Models of Protein Energy Landscapes and Slow Structural Rearrangements. *J Comput Biol* 25, 1 (2017), 33–50.




(a) PCA 2D Embedding (b) vAE (Run 1) 2D Embedding (c) vAE (Run 2) Embedding

(d) dAE (Run 1) 2D Embedding (e) dAE (Run 2) 2D Embedding (f) dAE (Run 3) 2D Embedding

**Figure 4: (a) plots the decoy dataset of the protein with native structure under PDB id 1aoy in the 2D latent PCA space. The embedded decoys are color-coded based on their lRMSD from the native structure in a blue-to-red color scheme indicating lower-to-higher lRMSDs. (b)-(c) show the 2D latent space learned by two different learned vAE models, each starting from random initial parameters. (d)-(f) do so for three different learned models of dAE$_{LR_E,\sigma_D}$.**

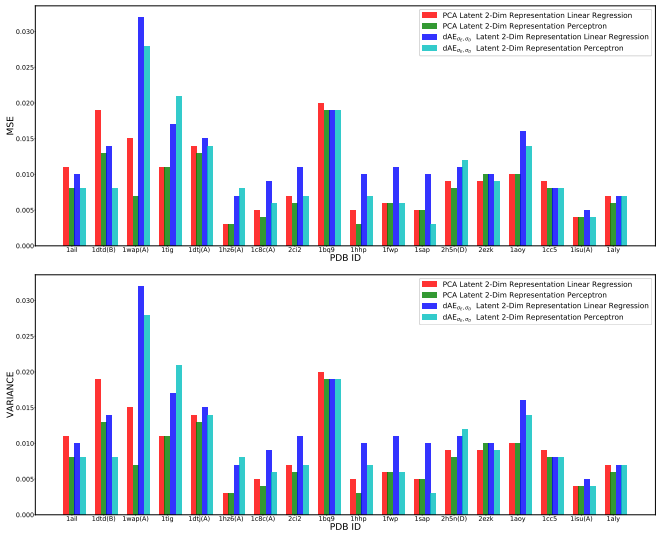



**Figure 5: MSEs and variance over testing datasets of linear and nonlinear regression models over 2D representations learned from PCA and dAE$_{\sigma_E,\sigma_D}$.**


[20] A. D. McLachlan. 1972. A mathematical procedure for superimposing atomic coordinates of proteins. *Acta Crystallogr. A.* 26, 6 (1972), 656–657.

[21] K. Molloy, J. V. Min, D. Barbará, and A. Shehu. 2014. Exploring Representations of Protein Structure for Automated Remote Homology Detection and Mapping of Protein Structure Space. *BMC Bioinf* 15, Suppl 8 (2014), S4.

[22] F. Noé and C. Clementi. 2017. Collective variables for the study of long-time kinetics from molecular trajectories: theory and methods. *Curr Opin Struct Biol* 43 (2017), 141–147.

[23] M. Osadchy and R. Kolodny. 2011. Maps of protein structure space reveal a fundamental relationship between protein structure and function. *Proc Natl Acad Sci USA* 108, 30 (2011), 12301–12306.

[24] E. Plaku, H. Stamati, C. Clementi, and L. E. Kavraki. 2007. Fast and Reliable Analysis of Molecular Motions Using Proximity Relations and Dimensionality Reduction. *Proteins: Struct. Funct. Bioinf.* 67, 4 (2007), 897–907.

[25] Mary A Rohrdanz, Wenwei Zheng, Mauro Maggioni, and Cecilia Clementi. 2011. Determination of reaction coordinates via locally scaled diffusion map. *J Chem Phys* 134, 12 (2011), 124116.

[26] S. T. Roweis and L. K. Saul. 2000. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290, 5500 (2000), 2323–2326.

[27] E. Sapin, D. B. Carr, K. A. De Jong, and A. Shehu. 2016. Computing energy landscape maps and structural excursions of proteins. *BMC Genomics* 17, Suppl 4 (2016), 456.

[28] E. Sapin, K. A. De Jong, and A. Shehu. 2018. From Optimization to Mapping: An Evolutionary Algorithm for Protein Energy Landscapes. *IEEE/ACM Trans Comput Biol and Bioinf* 15, 3 (2018), 719–731.

[29] A. Shehu, D. Barbará, and K. Molloy. 2016. A Survey of Computational Methods for Protein Function Prediction. In *Big Data Analytics in Genomics*, K. C. Wong (Ed.). Springer.

[30] A. Shkurti et al. 2019. CoCo-MD: A Simple and Effective Method for the Enhanced Sampling of Conformational Space. *J Chem Theory Comput* 15, 4 (2019), 2587–2596.

[31] J. Shlens. 2003. A tutorial on Principal Component Analysis. , 16 pages. https://www.cs.princeton.edu/picasso/mats/PCA-Tutorial-Intuition_jp.pdf

[32] J. B. Tenenbaum, V. de Silva, and J. C. Langford. 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* 290, 5500 (2000), 2319–2323.

[33] G. A. Tribello and P. Gasparotto. 2019. Using dimensionality reduction to analyze protein trajectories. *Frontiers Mol Biosci* 6 (2019), 46.

[34] L. J. P. van der Maaten, E. O. Postma, and H. J. van den Herik. 2009. Dimensionality reduction: A comparative review. *J Mach Learn Res* 10, 1-41 (2009), 66–71.

[35] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. 2010. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J Mach Learn Res* 11 (2010), 3371–3408.

[36] A. Zaman and A. Shehu. 2019. Balancing multiple objectives in conformation sampling to control decoy diversity in template-free protein structure prediction. *BMC Bioinformatics* 20, 1 (2019), 211. https://doi.org/10.1186/s12859-019-2794-5

[37] G. Zhang, L. Ma, X. Wang, and X. Zhou. 2018. Secondary Structure and Contact Guided Differential Evolution for Protein Structure Prediction. *IEEE/ACM Trans Comput Biol and Bioinf* (2018). https://doi.org/10.1109/TCBB.2018.2873691