CS230

# A Differentiable Protein Loop Detector

**Alexander E. Chu**
Biophysics Program and Department of Bioengineering
Stanford University
alexechu@stanford.edu

## Abstract

De novo protein design is a new field in protein engineering and computational biophysics, in which the complete sequence and structure of protein molecules can be specified *in silico*. However, current methods are limited to fairly simple protein architectures, and better tools for protein design are needed to access new protein folds and allow for faster iteration in the design process. One common challenge is the ability to detect and improve poorly designed protein "loops." Here, I report two deep neural network models that are able to capture protein secondary structure and detect which residues participate in the loops to an accuracy of 0.88. These models are differentiable and promise to integrate well with other newly developed tools for generating and discriminating protein structure designs.

## 1 Introduction

The field of de novo protein design has experienced significant breakthroughs over the last ten years [1]. We have attained the ability to specify protein backbones that are realistic (referred to as backbone design) as well as design side-chain sequences onto these backbones that fold to the designed structure (sequence design). These steps are accomplished with the aid of Monte Carlo optimization algorithms, which depend on an accurate scorefunction to evaluate and rank designs [2]. While existing scorefunctions are fast and effective for calculating higher-level thermodynamic properties of protein structures, they fail at detecting small local deviations that can have a disproportionate effect on global protein folding. As a result, current efforts at backbone and sequence design are case-specific, highly empirical, and not automated, requiring laborious manual curation and ranking to discern valid backbones and sequences [3]. A method for evaluating protein structures for their validity or "realistic-ness," on metrics that are not captured by thermodynamic scorefunctions, would significantly increase the rate of development in de novo protein design.

Such a method would be most useful if focused on protein structural "loops." Loops are regions of protein structure that do not take on well-defined secondary structure such as $\alpha$-helices or $\beta$-strands, and are often the most destabilizing and hardest to design properly. Currently, the algorithm used to define secondary structure in a 3D protein structure (stored in a PDB file) is DSSP [4], which is not a differentiable function. Deep learning is well suited to solving this task more efficiently, due to the difficulty of representing complex 3D protein structural data in a standard regression model and the wide variety of tasks it has proven effective for. A fully differentiable pipeline that could replace DSSP by detecting loops in PDB files, evaluating loops for their "realistic-ness," and be integrated with other generative and discriminative models for protein design, would be an important tool for ranking and evaluating computationally generated protein designs. Here, I propose to develop a differentiable loop detector for protein structures from PDB files.

My central hypothesis is that the universe of naturally occurring protein structures defines a distribution for allowable protein folding geometries, and that the structural data uploaded to the Protein

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Data Bank (PDB) represents an adequate sample from this distribution. I propose to develop a model to detect loops in proteins in a differentiable function. This detector will enable downstream work to evaluate Monte Carlo-generated protein structures on the basis of their "realistic-ness," or similarity to native protein structures. Doing so will enable more rapid prototyping of protein designs and successful enzyme engineering outcomes.

The input to the detector model is a $64 \times 64$ symmetric matrix of distances (in $\mathbb{R}^+$). This is a representation of a protein structure, which is simply a polymer with a particular 3D conformation, or fold. The elements of the matrix are calculated as pairwise distances between the C$\alpha$ atoms of each residue of the protein polymer. The detector model itself has been implemented here as a fully-connected deep neural network and as a convolutional neural network. These models are used to predict, for each of the 64 residues represented, whether that residue is part of a loop.

## 2 Related work

This work has not been previously attempted. Successful deep learning methods for predicting protein secondary structure from sequence have been reported [5], but these models are involved in learning the mapping between protein sequence and protein structure. Instead, my model is intended to be used in a design framework, with a known sequence and structure, but with unknown structure/design quality. DSSP [4] is an effective secondary structure labeling algorithm that takes protein structures as input, but it is not differentiable.

## 3 Dataset and Features

Data has been obtained as a non-redundant subset of all experimentally determined protein structures downloaded from CATH [6], a protein structure classification database. I used the s40 download, which filters out any structures from proteins with greater than 40 percent sequence identity, preventing excessive redundancy in the dataset. This download contains 30,745 Protein Data Bank (PDB) files, each of which contains a 3D structure of a protein of variable length.

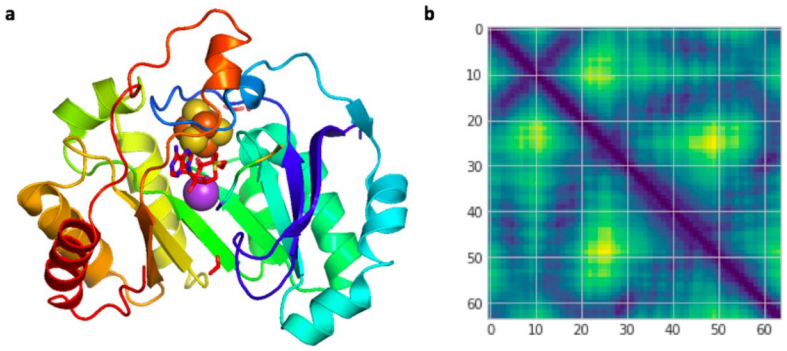


Figure 1. (a) A representative protein structure from the PDB. Loops are depicted as wires, whereas helices are depicted as ribbons and strands as arrows. (b) A representative 2D pairwise distance matrix for a 64-residue protein structure.

Data representation is an important consideration for the complex 3D structural data I will be working with. I will represent 3D structural data for these proteins as a 2D pairwise distance matrix between each residue of the sequence in the protein. In practice, this is a symmetric positive semidefinite matrix with one channel (depth 1). This representation lends itself well to standard convolutional approaches, and has already been shown to be sufficient for capturing global structural information [3, 7, 8]. For labels, I generated DSSP secondary structure definition labels for each residue in the protein, labeling a residue 1 if it belongs to a loop and 0 otherwise. Thus, for a protein of length $L$, I generate a pairwise distance matrix representation of the 3D structure in $\mathbb{R}^{L \times L}$, and a label vector in $\{0, 1\}^L$.

Proteins can be of variable length, yielding variable size matrices. In order to standardize input data size, I generated starting data by sliding windows of size $64 \times 64$ along the diagonal of the contact map for each matrix, at a stride of 10. In essence, this generates a distance matrix of a protein fragment, which should be sufficient to capture enough local environment for any given fragment. These are mapped to the corresponding DSSP label vector. Thus my input data are pairwise distance matrices in $\mathbb{R}^{64\times64}$, and my label vectors are binary vectors in $\{0,1\}^{64}$. The data was not normalized because we want to preserve true structural information and physical distances.

This yielded a total of 223,500 data examples, which I partitioned randomly 80-10-10 into train-dev-test sets to yield 178,800 training examples, 22,350 dev examples, and 22,350 test examples. Since local protein structure is generally independent of other local fragments, and the random partition preserves the structure of the PDB data, this is a valid partition.

## 4 Methods

My approach is to directly predict binary labels (loop vs. non-loop) for each residue in the input. Thus, for each $64 \times 64$ input "image," DSSP generates a binary vector of ground truth labels $y \in \{0,1\}^{64}$. I used two approaches to predict these $y$: a standard fully connected deep neural network, with four layers (FCNN), and a convolutional neural network (CNN). These models are implemented in the Keras framework [9]. The final layer of these networks is a dense layer with 64 output neurons with a sigmoid activation. Together these produce a prediction $\hat{y} \in [0,1]^{64}$.

To train these models, I minimized the standard binary cross-entropy loss between the predicted, encoded vector $\hat{y}$ and the ground truth label $y$. The objective function is shown here for $m$ examples and a label/prediction vectors of length 64. The optimal parameters $\theta^*$ are the parameters that minimize the loss.

$$\min L(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{64}\sum_{j=1}^{64} y_j^{(i)} \log \hat{y}_j^{(i)} + (1 - y_j^{(i)}) \log(1 - \hat{y}_j^{(i)})$$

$$\theta^* = \arg\min_{\theta} L(\hat{y}, y)$$

## 5 Experiments/Results/Discussion

For the FCNN model, training was carried out on AWS using a `p2.xlarge` instance. Hyperparameters considered during the training process included learning rate, minibatch size, the number of epochs to train for, and the overall model architecture (number of layers and number of neurons in each layer). Random weight initializations were used throughout. Initially, I used default hyperparameters of `learning_rate = 0.0001`, `minibatch_size = 32`, and `num_epochs = 20`. The number of epochs was lower initially because I wanted to iterate faster and find more global errors, with more refined debugging later with more deeply trained models. For this project, I also used the standard evaluation metric of accuracy, as it incorporates both false positives and false negatives, neither of which is especially detrimental over the other for this use case.

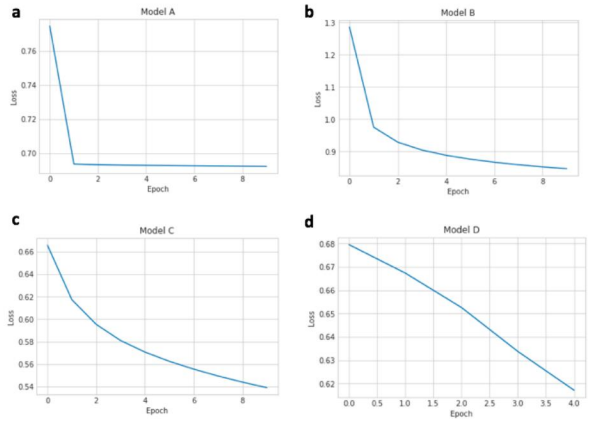


Figure 2. Loss curves for Models (a) A, (b) B, (c) C, and (d) D.

The initial model architecture was a three-layer neural network, with layer sizes of 256, 128, and 64 and ReLU activations. The initial input was unrolled from a $64 \times 64$ matrix to a vector of length 4096. This model (model A) achieved accuracy on the training set of 0.5850, and accuracy on the validation set of 0.5835, suggesting that the model suffers from high bias and is underfitting the training data (Table 1). The loss largely stopped decreasing after just a few epochs, suggesting that early stopping was not a cause of underfitting (Fig. 2A).

Next, I tried increasing the size of the model, using more neurons per hidden layer, with layer sizes of 4096, 1024, and 64. This model (model B) yielded training set prediction accuracy of 0.7100 and validation set prediction accuracy of 0.7095 (Table 1). Again this suggests high bias and low variance, and the loss stopped improving significantly after just a couple of epochs, suggesting that the underfitting is not due to early stopping (Fig. 2B). These results were a significant improvement, but I wondered if it was possible to achieve even better accuracy.

Next, I tried to deepen the network, training a four layer model with layer sizes 256, 1024, 256, and 64, again with ReLU activations (model C). This model achieved much greater accuracy on the training set, with an accuracy of 0.7314, and accuracy of 0.7292 on the validation set (Table 1). Encouragingly, the rate of improvement of the loss did not seem to drop off as quickly within 10 epochs (Fig. 2C), suggesting that longer training would produce better accuracy. Altering learning rate to be 0.001 also helped with improving accuracy. While the accuracy is significantly improved and I already expect it to be useful at this stage, I was unsure how to address the underfitting issue. I also trained much larger models, with thousands of hidden neurons and more layers, using other optimization algorithms such as Adam, but these models did not outperform this model. Using the same architecture, I trained again but for 1000 epochs (Final FCNN). This yielded the highest accuracy yet, 0.8412 on the training set and 0.8393 on the validation set. Observing that the validation loss and accuracy generally follow the trend of the training loss and accuracy, this suggests that the choice of loss function is good and that it allows generalizable training (Fig. 3A). I further observed that the validation accuracy had not yet begun to decrease when training was stopped after 1000 epochs, suggesting that the model could be trained for even longer, but there were limited resources. Variants of the model trained with different minibatch sizes did not yield better performance.

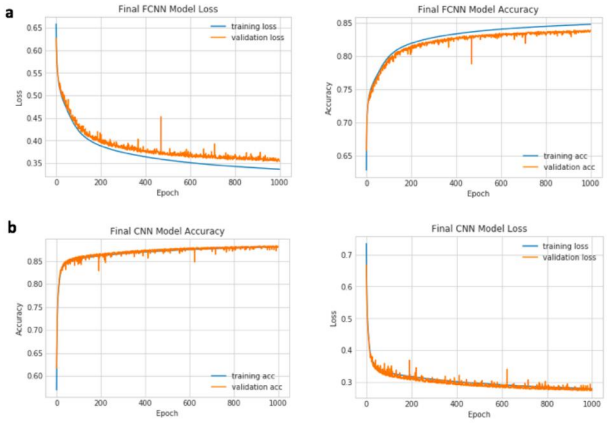


Figure 3. Loss and accuracy over training epochs for (a) the Final FCNN model and (b) the Final CNN model.

For the CNN model, training was carried out on AWS using a `p2.xlarge` instance. The same hyperparameters were varied as with FCNN training. Random weight initializations were used throughout. The model architecture that was chosen was simple and based on that used by Eguchi and Huang [7]. Initially I observed fairly poor accuracy for this model D (0.6679) which held for various alterations of model architectures, the hyperparameters, and optimization algorithms (Fig. 2D). Error analysis by viewing predictions for a few randomly selected training examples showed that the model was simply predicting '0' for all residues, or non-loop. This was surprising because the dataset does not suffer from class imbalance; the model was able to achieve accuracies around 0.6 by simply predicting a single class for all residues of all examples.

To investigate more closely, I examined the activation outputs of each layer of the model. Upon closer inspection of the activations of each layer, I noticed that the activations tended to have some extreme values and speculated that perhaps batch normalization could be used to better handle the

layer activations. I applied batch normalization both before and after each, as there is controversy as to which is more effective in practice, and found that batch normalization after the nonlinearity is more effective. This yielded initial accuracies of 0.7950 (training) and 0.8070 (validation) after just ten epochs of training, which was encouraging. Using the same hyperparameters that had been optimized for the FCNN, I then trained for 1000 epochs and achieved accuracies of 0.8805 (training) and 0.8811 (validation), the best accuracy yet observed for any model. Interestingly, the validation losses and accuracies are very close to those for the training set throughout, suggesting that the model (as did earlier versions) suffers from high bias and low variance. However, training larger and more complex convolutional networks did not improve performance; in fact, many of these experimental models performed worse than the simpler model, even on the training set.

Finally, after model iteration and training, I evaluated performance of the Final FCNN and Final CNN models on the test set, which had not been used at all during training, to preserve the validity of the inferential predictions made on the test set. I was able to achieve accuracies of 0.8405 for the Final FCNN model and 0.8822 for the Final CNN model (Table 1), showing the model generalizes well and has learned the distribution.

| Table 1. | Architecture | No. of parameters | Epochs | Batch Normalization? | Training accuracy | Validation accuracy | Test accuracy |
|---|---|---|---|---|---|---|---|
| Model A | Dense: 256, 128, 64 | 1,089,984 | 10 | No | 0.5850 | 0.5835 | - |
| Model B | Dense: 4096, 1024, 64 | 21,042,240 | 10 | No | 0.7100 | 0.7095 | - |
| Model C | Dense: 256, 1024, 256, 64 | 1,590,848 | 10 | No | 0.7314 | 0.7292 | - |
| Final FCNN | Dense: 256, 1024, 256, 64 | 1,590,848 | 1000 | No | 0.8412 | 0.8393 | 0.8405 |
| Model D | 7x7@32-Pool2,<br>4x4@64-Pool2,<br>4x4@128,<br>4x4@256-Pool2,<br>1x1@512-Pool2,<br>1x1@512-Pool2,<br>Dense: 256,512,64,<br>ReLU activations | 1,773,376 | 5 | No | 0.6626 | 0.6679 | - |
| Final CNN | 4x4@8-s2,<br>4x4@16-s2,<br>4x4@32-s2,<br>4x4@64-s2,<br>1x1@128-s1,<br>1x1@128-s1.<br>Dense: 256, 64.<br>LeakyReLU(0.2) activations | 610,344 | 1000 | Yes | 0.8805 | 0.8811 | 0.8822 |

## 6 Conclusion/Future Work

In conclusion, I was able to train two deep learning neural networks, a fully connected dense neural network and a convolutional neural network, to predict residue-wise secondary structure labels (loop vs. non-loop) to accuracies of 0.88 (Table 1). These networks are fully differentiable and could potentially be used in a computational protein design framework to detect and eventually evaluate the quality and robustness of computationally generated loops in de novo protein design. Both models performed well, although the CNN was able to perform slightly better with fewer parameters, reaffirming the efficiency of CNNs for these matrix-input problems.

In future work, I hope to further improve the accuracy of the models. With more computational resources, I would be able to do a more thorough random hyperparameter search, over the ones that were varied in this study, as well as various seeded random parameter initializations. It would also be interesting to see if very large models trained for much longer would be more accurate. Finally, I hope to be able to leverage this model to not only detect protein loops, but evaluate them on torsional angles and other biophysical properties that are common to real protein loops. In doing so, we would be able to distinguish poor computational designs that are currently hard to evaluate with the standard thermodynamic scorefunctions used in computational protein design.

The code for this work is available at `https://github.com/alexechu/cs230_loopdetector`. Data and model parameters available upon request.

## 7 Contributions

A.E.C. designed the research project and goals, carried out the research, analyzed the results, and wrote the paper. The author acknowledges helpful guidance from Raphael R. Eguchi (Lab of Possu Huang) in designing the research and implementing the project, and guidance from Ahmad Momeni (CS 230 TA) in implementing the project.

## References


[1] Po-Ssu Huang, Scott E. Boyken, and David Baker. "The coming of age of de novo protein design". en. In: *Nature* 537.7620 (Sept. 2016), pp. 320–327. ISSN: 1476-4687. DOI: 10.1038/nature19946. URL: https://www.nature.com/articles/nature19946 (visited on 06/04/2019).

[2] Carol A. Rohl et al. "Protein Structure Prediction Using Rosetta". In: *Methods in Enzymology*. Numerical Computer Methods, Part D 383 (Jan. 2004), pp. 66–93. DOI: 10.1016/S0076-6879(04)83004-0. URL: http://www.sciencedirect.com/science/article/pii/S0076687904830040 (visited on 06/04/2019).

[3] Namrata Anand and Possu Huang. "Generative modeling for protein structures". In: *Advances in Neural Information Processing Systems 31* (2018). Ed. by S. Bengio et al., pp. 7494–7505. URL: http://papers.nips.cc/paper/7978-generative-modeling-for-protein-structures.pdf (visited on 06/04/2019).

[4] Wolfgang Kabsch and Christian Sander. "Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features". en. In: *Biopolymers* 22.12 (1983), pp. 2577–2637. ISSN: 1097-0282. DOI: 10.1002/bip.360221211. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/bip.360221211 (visited on 06/04/2019).

[5] Fengbin Wang et al. "Structure of Microbial Nanowires Reveals Stacked Hemes that Transport Electrons over Micrometers". In: *Cell* 177.2 (Apr. 2019), 361–369.e10. ISSN: 0092-8674. DOI: 10.1016/j.cell.2019.03.029. URL: http://www.sciencedirect.com/science/article/pii/S0092867419302910 (visited on 04/24/2019).

[6] Natalie L. Dawson et al. "CATH: an expanded resource to predict protein function through structure and sequence". en. In: *Nucleic Acids Research* 45.D1 (Jan. 2017), pp. D289–D295. ISSN: 0305-1048. DOI: 10.1093/nar/gkw1098. URL: https://academic.oup.com/nar/article/45/D1/D289/2605733 (visited on 06/04/2019).

[7] Raphael R. Eguchi and Po-Ssu Huang. "Multi-Scale Structural Analysis of Proteins by Deep Semantic Segmentation". en. In: *bioRxiv* (Nov. 2018), p. 474627. DOI: 10.1101/474627. URL: https://www.biorxiv.org/content/10.1101/474627v3 (visited on 06/04/2019).

[8] Namrata Anand, Raphael Eguchi, and Po-Ssu Huang. "Fully differentiable full-atom protein backbone generation". In: *ICLR* (Mar. 2019). URL: https://openreview.net/forum?id=SJxnVL8YOV (visited on 06/04/2019).

[9] Francois Chollet. *Keras*. Dec. 2015.