Chapter 5

Machine Learning-driven Protein Library Design: A Path Toward Smarter Libraries

Mehrsa Mardikoraem and Daniel Woldring

Abstract

Proteins are small yet valuable biomolecules that play a versatile role in therapeutics and diagnostics. The intricate sequence–structure–function paradigm in the realm of proteins opens the possibility for directly mapping amino acid sequence to function. However, the rugged nature of the protein fitness landscape and an astronomical number of possible mutations even for small proteins make navigating this system a daunting task. Moreover, the scarcity of functional proteins and the ease with which deleterious mutations are introduced, due to complex epistatic relationships, compound the existing challenges. This highlights the need for auxiliary tools in current techniques such as rational design and directed evolution. To that end, the state-of-the-art machine learning can offer time and cost efficiency in finding high fitness proteins, circumventing unnecessary wet-lab experiments. In the context of improving library design, machine learning provides valuable insights via its unique features such as high adaptation to complex systems, multi-tasking, and parallelism, and the ability to capture hidden trends in input data. Finally, both the advancements in computational resources and the rapidly increasing number of sequences in protein databases will allow more promising and detailed insights delivered from machine learning to protein library design. In this chapter, fundamental concepts and a method for machine learning-driven library design leveraging deep sequencing datasets will be discussed. We elaborate on (1) basic knowledge about machine learning algorithms, (2) the benefit of machine learning in library design, and (3) methodology for implementing machine learning in library design.

Key words Library design, Directed evolution, Machine learning, Deep learning

1 Introduction

Proteins are molecules with a wide variety of applications in biological processes. They have fundamental functions in living organisms such as being catalysts, receptors, structural elements, transporters, and regulators [1–5]. Accordingly, increased potential for mapping the protein sequence to its function will result in comprehension and regulation of biological processes associated with functional disorders. As a result, techniques to efficiently navigate the protein fitness landscape are at the forefront of protein

Michael W. Traxlmayr (ed.), *Yeast Surface Display*, Methods in Molecular Biology,
vol. 2491, https://doi.org/10.1007/978-1-0716-2285-8_5,
© The Author(s), under exclusive license to Springer Science+Business Media, LLC, part of Springer Nature 2022

engineering. Nevertheless, the complexity and ruggedness of the protein fitness landscape, and the high potential of failure in searching through the fitness landscape (mutations resulting in unstable and non-functional variants), make this task more demanding. A growing number of computational and experimental approaches (e.g. high-resolution stability calculations [Chapter 3 of this volume], virtual screening [6], deep sequencing [7], cytometry-based selections [8], ancestral sequence reconstruction [Chapter 4 of this volume]) seek to address these gaps in knowledge.

Machine learning algorithms offer a platform for harnessing large, diverse datasets for the purpose of understanding natural protein features and guiding protein engineering efforts. This provides the opportunity to map protein sequence to function without requiring explicit biophysical knowledge of individual sequences. An additional advantage of such algorithms relates to expanding the utility of experimentally derived sequences beyond the often small subsets of lead variants, i.e. while directed evolution discards low fitness protein variants, machine learning can learn the characteristics of these sub-optimal variants and, in turn, increase the model's predictive performance [9]. Machine learning can be particularly advantageous for protein engineering campaigns that involve low-throughput or laborious selections. Therefore, its usefulness depends on factors such as library size, screening difficulty, fitness landscape ruggedness, and the accuracy of the predictive model. As a result, the ability to capture complicated trends among protein datasets, aided by non-linear functionality to reveal important features, makes machine learning a powerful tool for guiding protein library design.

Machine learning has played an important role in protein engineering for more than 30 years, yielding improved prediction of tertiary structure [10] and protein–protein interactions [11] using amino acid sequence information alone [10, 11]. Machine learning algorithms such as support vector machines (SVMs), random forest (RF), and gradient boosting machines (GBMs) have provided platforms for protein function and property prediction including stability, catalytic activity, and secondary structure [12–15]. Machine learning models can also be used in predicting the developability and evolvability of protein sequences [16]. Deep learning (DL) [17], as a sub-field of machine learning, imitates human brain functionality in decision making and learning experiences. Utilizing non-linear functions, the algorithm can learn and extract desired features from the provided input data, well suited for dealing with rich datasets with high dimensionality. This makes deep learning methods particularly promising for evaluating sequence–function trends among a rapidly growing number of protein sequences. Although practical and promising, developing a fine-tuned strategy to employ machine learning in the field demands an awareness of the existing challenges and capabilities. Here, we present a procedure for establishing deep learning models that

guide protein library design. Common challenges and best practices in this burgeoning field will be highlighted.

We consider multiple practical applications of machine learning within the context of protein library design: (1) combinatorial library design based on deep sequencing data following high-throughput directed evolution, (2) process parallelization and parameterization of features within far-reaching parts of the fitness landscape, and (3) the ability to sample the diversity applied by specific degenerate codon techniques and oligonucleotide combinations prior to experimental implementation.

1.1 Providing a Better Starting Point for Directed Evolution

Directed evolution campaigns are initialized using a parent sequence to implement mutations onto. Therefore, the path-dependency of directed evolution benefits from a high-fitness or highly stable sequence as a starting point to increase the probability of finding optimal regions within a fitness landscape [18]. Machine learning can guide directed evolution by identifying a large collection of promising sequences based on a curated input data. In a recent example, a machine learning model was trained based on the initial library of fluorescent proteins to build a second small yet enriched protein library [19].

1.2 Investigating Unexplored Parts of the Fitness Landscape

Machine learning algorithms provide some unique advantages that enable finding unexplored variants which may possess high fitness. Large datasets which are processed with high-complexity algorithms are not only able to predict functionality but can learn hidden characteristics and rules that exist in provided data (*see* **Note 1**). As an example, UniRep [20, 21] has been trained on 24 million protein sequences from Uniref50 [22] to obtain general trends in protein sequences and statistical representations of amino acids. The authors claim strong generalizations and high performance in downstream tasks taking advantage of their embedding methods. Another factor which is effective in finding the unexplored high fitness sequences is the application of parallelism and multi-tasking within machine learning. Multi-task learning is a subset of machine learning algorithms that trains multiple tasks simultaneously in one unique model [23]. This feature enables capturing the epistatic relationships in protein sequences by providing a path-independent search in the fitness landscape [24]. In this way, applying machine learning to protein engineering enables an increased likelihood of accessing undetected regions of the fitness landscape.

1.3 Estimating Degenerate Codon Performance via Fitness Distribution Analysis

Implementing a well-trained machine learning algorithm enables the evaluation of multiple design strategies and reduces experimental effort. Various experimental techniques have been developed to improve the efficiency of directed evolution such as gene shuffling [25] and neutral drift library screening [26] to manage the library size and increase the likelihood of finding the desired property.

Another highly used method is to generate libraires based on degenerate codons in order to introduce tailored diversity at individual positions. As an example, while NNK is used to generate a library coding for all amino acids (with 3% stop codons), other combinations such as NYC (coding for hydrophobic residues), KST (coding for small residues), and NDT (coding for a balanced set of all amino acid properties) are available. In addition, several impressive computational attempts have been used to even go beyond these techniques and optimize the oligonucleotide combinations [27, 28]. Among these potential degenerate codon techniques for the library design, the user can pre-analyze their performance for their desired protein.

Figure 1 proposes a comparison of candidate degenerate codons. As a base platform, an initial deep learning model is trained on the objective protein. Subsequently, the algorithm can generate sequences that incorporate candidate degenerate codons and assign a predicted fitness score to each based on the previously trained model. Finally, the distribution of fitness scores is calculated for all variants of each degenerate codon strategy. Statistical tools such as Jensen–Shannon Divergence [29] and survival function [30] provide a direct comparison between individual distributions in terms of diversity and fitness (*see* **Note 2**). The criterion for choosing the best performance depends on the particular application the platform is used for.

2 Materials

There are a plethora of software libraries and packages provided for implementing model optimization, machine learning, and deep learning algorithms. Choosing one depends on the specific application that the user has in mind. TensorFlow [31], Keras [32], PyTorch [33], and Scikit-learn [34] are among the most popular libraries with each having some trade-offs (*see* **Note 3** for analysis). The discussion found here focuses on deep learning in Keras. Please refer to our GitHub (https://github.com/WoldringLabMSU/DeepLearning.git) for a collection of relevant Python scripts to guide model implementation as well.

1. The latest version of python (https://www.python.org) installed (Installing Anaconda (https://www.anaconda.com) is highly suggested for beginners as it is straightforward and easy to use).
2. Spyder or Jupyter Notebook environments in Anaconda are both popular and practical in doing machine learning and deep learning projects.
3. pandas [35] (For dealing with data frames).
4. numpy [36] (For efficient mathematical operations).

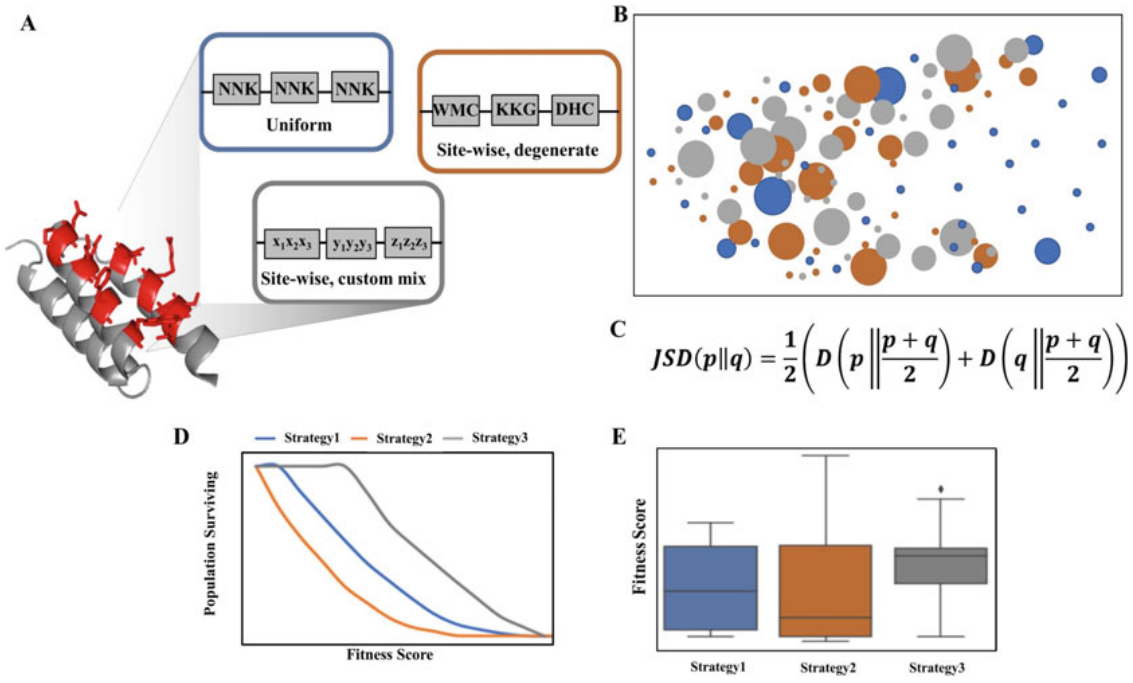

Fig. 1 Evaluation of degenerate codon performance within multiple design strategies using the trained model based on experimental data on the protein of interest. (**a**) Elaboration of different hypothetical degenerate codon strategies in three different sites of the protein (the previously trained model needs to be trained on high-quality experimental data in order to predict the fitness of generated sequences in each technique.) The first utilizes the NNK technique in all sites, while the second uses different degenerate codons at each site. The third strategy uses custom mix codons (versatile ratios of base pairs). Sequences compatible with the rubrics of each strategy can be generated and a fitness score for the generated sequences will be assigned to each sequence based on the previously trained model in the protein of interest. (**b**) The score function (transformed and standardized predicted scores) distribution will be obtained for each candidate library. One method to comprehend the predicted data is by clustering the generated data. The exemplary plot shown in here is based on both the sequences and scores of the three strategies when each dot represents one sequence produced by one of the strategies. The radius of the dots will represent their individual fitness scores, and the distance between any two dots will represent how distant those amino acid sequences are from each other in sequence space. (**c**) The Jenson–Shannon Divergence is useful for quantifying differences that exist between individual distributions and a reference distribution or the extent of similarity between candidates' distributions. (**d**) The survival function provides the probability of improved score functions compared to the current score function, providing more insight for analysis of distributions. (**e**) Box plots can be produced based on the generated fitness scores and are informative for recognizing the quantiles and making comparison between them in the three different strategies

5. scikit_learn [29] (This is mostly used in machine learning but its preprocessing section has a myriad of useful functions for analysis and fine tuning the data).
6. tensorflow [37] (For using keras, its backend should be installed (CNTK and Theano are other options, as well).
7. keras [32].

In order to use these libraries, use pip command (`pip install X`) in the command prompt.

3 Methods

The following is a general workflow representing the required actions in building a deep learning algorithm from deep sequencing data (Fig. 2). Here, the goal is to produce a supervised learning algorithm for predicting protein function based on amino acid sequence.

3.1 Data Processing as an Initial Yet Pivotal Step in Any DL Algorithm

The main purpose of this step is to prepare the data to be fed into the deep learning algorithm. Importantly, what can be learned by the model strongly depends on what is provided to the algorithm as an input. If the aim is to map the sequence to function, protein sequence should be as an input labeled with the desired functionality (output). Three important steps in data processing include input data refinement, input data representation, and output data representation (if dealing with supervised learning). *See* **Note 3**.

3.1.1 Input Data Refinement

Scaling and refining the data aims to remove the "importance" of the raw magnitude of one factor relative to another and as a result reduces estimation errors and calculation times. One package which

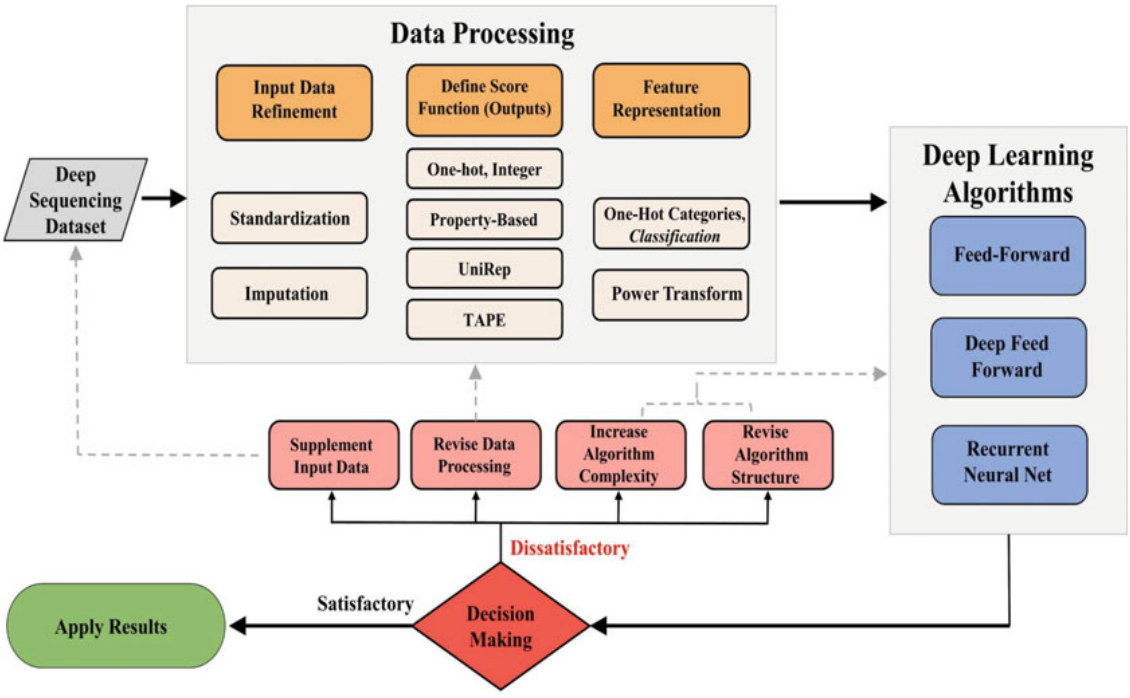

Fig. 2 This figure provides the overall workflow of what the user may encounter when using deep sequencing data and wants to apply some state-of-the-art machine learning algorithm. The main steps for using this path are data processing, choosing an appropriate learning algorithm and deciding based on the algorithms' findings. The detailed explanation of steps is mentioned in the content

can be used for standardization is StandardScaler or RobustScaler (advantageous when dealing with outliers) from sklearn preprocessing package:

```
from sklearn.preprocessing import StandardScaler
          scaler1 = StandardScaler()
          scaler1.fit(Feature)
          Feture_standardized1 = scaler1.transform(Feature)
          ###
          from sklearn.preprocessing import RobustScaler
          scaler2 = RobustScaler()
          scaler2.fit(Feature)
          Feture_standardized2 = scaler2.transform(Feature)
```

3.1.2 Input Data Representation

One-hot encoding (*see* **Note 4**), integer encoding (*see* **Note 5**), physiochemical property-based encoding (*see* **Note 6**), and sequence embedding (e.g. UniRep [20] (https://github.com/churchlab/UniRep), and TAPE [38] (https://github.com/songlab-cal/tape)) (*see* **Note 7**) are notable options for representing amino acid sequence data. The user can manually one-hot encode the input sequences via defining dictionaries or using packages in python (Refer to https://github.com/WoldringLabMSU/DeepLearning.git for more information on encoding).

```
from sklearn.preprocessing import LabelEncoder
          from sklearn.preprocessing import OneHotEncoder
         Amino_Acids = ["A","C", "D", "E","F", "G", "H", "I",
"K", "L", "M", "N", "P","Q","R", "S", "T", "V","W", "Y"]
          label_encoder = LabelEncoder()
          onehot_encoder = OneHotEncoder(sparse = False)
               integer_encode = label_encoder.fit_transform
(Amino_Acids)
              integer_encoded = integer_encoded.reshape(len
(integer_encode), 1)
           Amino_Acids_onehot = onehot_encoder.fit_transform
(integer_encoded)
```

3.1.3 Output Data Representation

Following high-throughput selection and deep sequencing of an initial combinatorial library, amino acid sequences can be labeled based on the observed enrichment ratios as a metric for relative fitness [39]. Depending on the experimental conditions for selection and depth of sequencing, the distribution of enrichment ratio data may take on various forms and require further refinement (*see* **Note 9**).

3.2 Deep Learning Algorithm Selection Requires an Understanding of Each Algorithm Structure

3.2.1 Overview

Artificial neural network (ANN) architecture is inspired by human brain function which can learn from various input data. The building blocks for this network (neurons) receive information from adjacent neurons, then process this information with the aid of an activation function (*see* **Note 9**) before being sent to other downstream neurons. The effect and significance of each connection is proportional to its assigned weight.

There are many deep learning methods utilized in the field from feed-forward neural network (FNN) to the convolutional neural network (CNN) [40] and recurrent neural network (RNN) [41]. FNNs are primary neural network algorithms which are rigorous and powerful in capturing high dimensional features. It consists of three distinct layers (each composed of neurons): input layer, hidden layer, and output layer. The input layer receives the data features, the hidden layer transforms the input layer to the output by updating the weights iteratively to minimize the loss function. For each of these neural network algorithms, backpropagation is used to update model weights to minimize the loss function via gradient decent. Finally, the output layer captures the results. Figure 3 illustrates one example structure of a feed-forward neural network.

Convolutional neural networks are an additional deep learning algorithm inspired by the visual cortex in animals. CNNs capture hidden relationships and spatial dependencies in provided input via various filters, pooling, and convolutional kernels. Therefore, local properties will be obtained by using sliding filters and non-linear functions [42]. In recurrent neural network, there is a cyclic connection in the algorithm structure, whereby the current state of the algorithm will be updated based upon the past state and the current input data [43]. This characteristic makes RNN useful in time series data and capturing temporal relationships in sequences. In addition, variations of RNN such as LSTM [44] and GRU [45] have been developed to resolve the potential problems in its architecture such as capturing long distance relationships and resolving vanishing gradient problems. The best performance in the mentioned algorithms depends on the type of data and purpose for using deep learning. However, more advanced architectures like RNN/CNN are able to learn properties more efficiently than a traditional FNN. Transformers are another promising deep learning model utilizing attention-based mechanisms [46]. They enable learning the context and extract meaning from unlabeled data. This characteristic makes them very attractive to be used in protein engineering platform as the number of unlabeled sequences in protein databases outweighs the labeled data [20, 38, 47].

Descriptions of commonly used terms and techniques used in deep learning and optimization are shown in Table 1.

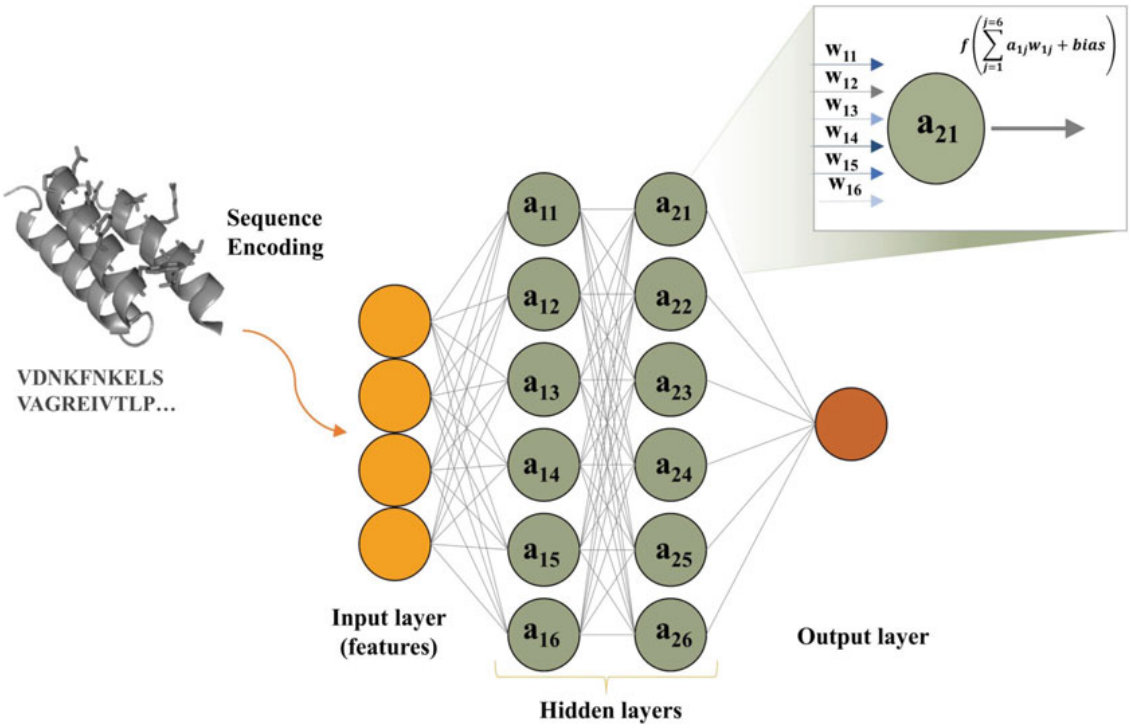

Fig. 3 This figure represents one simple example for a feed forward neural network. It consists of three main layers: the input layer, hidden layer(s), and output layer. It is a symbolic representation of the path from sequence to function. The magnifying glass focuses on one neuron (a21), illustrating how information passes through each neuron. Moreover, the information from six neurons in the previous hidden layer are multiplied by their weights (different colors are representative of activation level in each corresponding neuron). Afterwards, the result will be summed with bias and passes through the activation function to determine the output of that neuron

3.2.2 Guidance for Building a Deep Learning Structure

Here we build a simple feed-forward neural network in Keras to show one possible and simple format for building a neural network structure (Refer to https://github.com/WoldringLabMSU/DeepLearning.git for more advanced structures).

1. Importing the required libraries from Keras.

```
from tensorflow import keras
          import keras
          from keras.models import Sequential
          from keras.layers.core import Dense
          from keras.optimizers import Adam
          from keras.activations import relu,
 from keras.activations import sigmoid
 from sklearn.model_selection import train_test_split
```

2. Splitting the data set into train and test,

```
X_train, X_test, Y_train, Y_test = train_test_split(features,
labels, test_size=0.2, random_state=42)
```

Table 1
Deep learning and optimization terminology and usage

Concepts and terms	Definition	Application and suggestion
Train set	Part of data used for fitting the parameters	
Test set	Part of data used for evaluation of the trained model	
Evaluation set	Part of data (it can also be considered as segment of training data) used for tunning the hyperparameters	Avoid overfitting
Classification	Prediction when output values are discrete classes	
Regression	Prediction when output values are continues (mostly called as quantities than discrete labels)	
Overfitting	Lack of model ability to generalize from trained data to unseen data	
Loss function	A function for evaluating the algorithm calculating the difference between predicted and actual value	Common for classification: accuracy, cross-entropy Common for regression: mse,mape
Backpropagation	Algorithm used for updating model weights based upon the gradient of the loss function	
Epoch	Each epoch represents the entire training dataset has been passed to the system entirely	Needs optimization
Batch size	Number of samples in each break of training set (called as batch) where backpropagation will be carried out	Needs optimization
Optimizer	Strategy for minimizing the loss function	Needs optimization ADAM and SGD (two potential candidates)
Learning rate	The magnitude of steps in each iteration during backpropagation (implemented via gradient*learning rate)	Needs optimization (Adaptive learning rate is suggested)
Shuffling	Randomly change the order of existed data	
Dropout	Randomly dropping some connections between neurons	Avoid overfitting
Dense layer	Linear operation which maps every input to every output by weights	
Convolutional layer	Consists of filters convolving through the provided matrix	
Pooling layer	Used after convolutional layer for reducing the spatial size	Max pooling Average pooling
Data augmentation	Increasing the number of samples by adding implementing modifications to the original samples	Avoid overfitting
Regularization	Modification in the original cost function to reduce bias and increase penalty pertinent to magnitude of the weights	Avoid overfitting

3. Defining the model.

```
My_Model = Sequential()
```

4. Defining the input layer, number of neurons in that layer, its shape and its activation function.

```
My_Model.add(Dense(20, activation='relu', input_shape=
(100,)))
```

5. Defining the hidden layers (the # of hidden layers is another hyperparameter), number of neurons (hyperparameter) in each hidden layer and corresponding activation function.

```
My_Model.add(Dense(10, activation='relu'))
          My_Model.add(Dense(5, activation='relu'))
```

6. Defining the output layer [the last activation function will depend on the task (*see* **Note 8**)]:

```
My_Model.add(Dense(1, activation='linear'))
```

7. Compiling the model and determining the optimizer, learning rate, loss function, and evaluation metrics (numbers should be optimized based on the problem):

```
My_Model.compile(Adam(lr=0.01, decay=0.003),loss='mean_squar-
ed_error',metrics=['mse'])
```

8. Fitting the model is accomplished by defining the batch size, epochs, and verbose. Validation split specifies the fraction of data to be used for validation:

```
My_Model.fit(X_train,Y_train,batch_size=100,epochs=40,ver-
bose=2, validation_split=0.2,  shuffle= True)
```

9. Evaluating the fitting performance:

```
Metric = My_Model.evaluate(X_test, Y_test, verbose =2)
```

10. Predicting the labels for the test data set:

```
y_test_predicted= My_Model.predict(X_test, verbose = 2)
```

3.3 Visualization Guidance

Evaluation metrics provide useful insights to algorithm performance. Visualizations of these results help to further interpret and communicate the findings. For example, the seaborn library allows for showing the correlation between predicted output values versus actual values in regression:

```
import seaborn as sns sns.jointplot(Prediciton,Actual_value,
kind='scatter')
```

For classification, one simple method is using a confusion matrix to show model performance on each class prediction consisting of: true positives, true negatives, false positives, and false negatives values.

```
from sklearn.metrics import confusion_matrix
      import matplotlib.pyplot as plt
      import seaborn as sns
      cm = confusion_matrix(Y_test, Prediction)
     tn, fp, fn, tp= confusion_matrix(Y_test,Prediction).ravel
()
      ax= plt.subplot()
      sns.heatmap(cm, annot=True, fmt='g', ax=ax)
      ax.set_xlabel('Predicted labels')
      ax.set_ylabel('True labels')
      ax.set_title('Confusion Matrix')
```

Understanding and building such a structure is an important first step to take for utilizing deep learning in different projects. However, even with a powerful algorithm and appropriate data processing step, the prediction might be drastically off which leads us to decision making and further fine-tuning steps.

3.4 Decision Making and Evaluating Parameters

After processing the data and choosing the appropriate algorithm, one should be able to accurately evaluate the performance of that algorithm. Evaluation metrics depend on whether the problem is regression or classification. In classification, metrics such as accuracy, confusion matrix, AUC, and Recall are useful. While in regression, metrics such as RMSE, MSE, MAPE are better suited (*see* **Note 10**). Obtaining poor metrics for a model may arise from various stages in the algorithm such as inadequate input data, deficient preprocessing step, overfitting, and untuned hyperparameters. Choosing the right hyperparameters and preventing the system from overfitting are two necessary tasks in training any deep learning algorithm. Two generally used methods for resolving some of these issues are elaborated in the following.

3.4.1 Hyperparameter Optimization [48]

Hyperparameters are set of variables that dictate various characteristics of the algorithm's structure and influence the process used for training models. It can be considered as a meta-optimization technique whereby parameter value fitness is monitored via the loss function during the training process [49]. Multiple methods can be employed for searching through hyperparameter space (e.g., manual search, grid search, random search, and Bayesian optimization). The manual search involves tuning the hyperparameters by a user based on guess and check. In grid search, for each parameter of interest, the user defines a list of values to implement. The algorithm then calculates the loss for all possible combinations of parameter values. In random search, some but not all combinations of parameters will be selected randomly, and the best loss will be chosen based on the selected parameters. Bayesian optimization offers high efficiency by using information from the past as an experience to choose the next set of hyperparameters. HYPEROPT (http://hyperopt.github.io/hyperopt/) and OPTUNA (https://optuna.org/) are among the python libraries designed for hyperparameter optimization based on the objective function.

(Refer to https://github.com/WoldringLabMSU/DeepLearning.git for more information).

3.4.2 K-Fold Cross-Validation

This resampling approach allows for more accurate prediction in model performance using even a limited data set. It splits the data into k complementary groups and uses $k - 1$ groups for training and one group for evaluating the performance. The performance of the cross-validation is calculated by taking the mean and variance over all k performances. This enables a less biased estimate of model performance [49]. (Refer to https://github.com/WoldringLabMSU/DeepLearning.git, for example, K-fold cross-validation).

3.5 Protein Library Construction

Machine learning techniques enable the design of smart libraries by identifying the protein positions that are highly amenable to mutation and determining the most suitable degenerate codon candidate(s) for the protein of interest. Based on these designs, full length genes can then be constructed using overlap extension PCR of degenerate oligos to incorporate the intended site-wise amino acid diversity. Finally, the newly constructed full length gene library, combined with a linearized yeast surface display vector (e.g. pCT-CON2), can be electroporated into yeast (EBY100) [50] and evaluated by high-throughput techniques [16, 51–56].

4 Notes

1. *Informed models*: It should be clarified that it not always large datasets and high-complexity algorithms that improve the prediction performance (a simple feed-forward neural network including the epistasis relationship in feature representation may perform better than a convolutional neural network with one-hot encoding as its feature representation method).
2. *Learning paradigms in ANN*: The learning paradigms in ANN are supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. In supervised learning, the data are provided with assigned labels that then guide the algorithm to learn the input–output mapping through a backpropagation process. Unsupervised learning algorithms (e.g. K-nearest neighbors and K-means) are used when there are no values assigned to the input features. This allows for the detection of patterns even when additional information is not provided with the input data. Semi-supervised learning is a method well suited where only partially labeled data exists, but the algorithm aims to fully benefit from provided information either labeled or unlabeled [57]. Reinforcement learning is agent-based learning interacting with the environment. Therefore, the algorithm learns based on rewards from correct prediction and penalties from incorrect guesses (i.e., trial and error methodology) [58].
3. *Deep learning packages analysis*: TensorFlow is one of the most popular and fastest evolving open source deep learning tools [37]. It is compatible with both GPU/CPU computation and is well suited for working with multi-dimensional arrays. One downside could be the low-level API which makes it to be not the ideal choice for the direct creation of deep learning algorithms. Keras (open source) can support backends such as CNTK, TensorFlow, and Theano and its simple API makes it straightforward to implement. PyTorch is among the more flexible programming packages in python and supports tensor computation and GPU-acceleration. Its dynamic graph and easy debugging make it a strong option to choose. At its core, it uses CPU and GPU tensor and NN backends. Therefore, PyTorch brings speed and flexibility to deep learning models despite needing third-party visualization [59]. Regardless of the choice in packages, the magnitude and quality of the input data will have a strong impact on the predictive power of the resulting model.
4. *One-hot encoding*: The simplest method is to use one-hot encoding by constructing a matrix of one and zeros where one represents the existence of the element at a specific position

of the sequence. One-hot encoding is easy to implement and has been proven to be effective in many cases, yet it is highly memory intensive and struggles to capture the relationship between amino acids in protein sequences. This large and sparse encoding often leads to complications in training as a result of the inherently high dimensionality.

5. *Integer encoding*: Integer encoding is implemented by representing each amino acid by an integer. For integer encoding one observed drawback is the tendency of the system to assume linear relationships among the provided labels. For example, if the labels are 1, 2, and 3, the system assumes a relationship between the amino acids (such that 1 is closer to 2 than 3) that are assigned to these labels. As a result, orthogonality between the labels matters. However, integer encoding is often used with a linear embedding layer (*see* tf.keras.layers.embedding) whereby integer encoding calls an embedding column like a "lookup" table.
6. *Property-based encoding*: Some practical encodings are obtained based on the physiochemical properties (e.g. charge, hydrophobicity and size) of the sequences [60]. One example is principal components score Vectors of Hydrophobic, Steric, and Electronic properties (VHSE8) [61].
7. *Sequence embedding via self-supervised learning*: Utilizing language-based models and taking advantage of techniques such as next token prediction and masked token prediction, fixed vector representations of protein sequences will be reached. Two benchmark studies in this area are UniRep [20] and TAPE [38] embeddings.
8. *Desired data distribution for DL algorithms*: Gaussian-like distributions tend to have better performance in deep learning. Therefore, dealing with data having a skewed distribution may benefit from applying power transform functions to make the data more gaussian. For example, simple power transform functions may take the *n*th root or the *n*th order logarithm of the variable. More advanced power-transformers include the Box-Cox and Yeo-Johnson methods. In order to obtain new distributions with the mentioned methods, Scipy [62] library or scikit-learn preprocessing package power transformer can be used:

```
import scipy
       import scipy.stats
       Yeo-Johnson = scipy.stats.yeojohnson(label_List)
       Box_Cox = scipy.stats.boxcox(label_List)
```

9. *Activation function*: One suggestive method is to use ReLU in all hidden layers and choose an appropriate activation function for the output layer to match the distribution of data with the nature of the task**.** Generally, for the output layer, sigmoid (for binary-class), softmax, and tanh (for multi-class) are used for classification tasks, and linear activation function is used for regression.
10. *Evaluation metrics*: Evaluation metrics are representative of algorithm performance and should be considered within the context of the nature of the problem and origin of the input data. As an example, in biased data when 90% of the population are in category 1 and the remainder are in category 2, it is highly probable that the algorithm predicts all the data to be in the first category. In this case, if one wants to rely on the metrics, the accuracy will be 90% which is not addressing the performance of the algorithm (i.e., not representing the poor performance in prediction of other class). In this case, the confusion matrix provides useful information about the number of false positives, false negatives, true positives, and true negatives.

References

1. Hogan BL (1996) Bone morphogenetic proteins: multifunctional regulators of vertebrate development. Genes Dev 10:1580–1594
2. Schlessinger J (2000) Cell signaling by receptor tyrosine kinases. Cell 103:211–225
3. Syrovatkina V, Alegre KO, Dey R et al (2016) Regulation, signaling, and physiological functions of G-proteins. J Mol Biol 428: 3850–3868
4. Hellinga HW, Marvin JS (1998) Protein engineering and the development of generic biosensors. Trends Biotechnol 16:183–189
5. Mishra NK, Chang J, Zhao PX (2014) Prediction of membrane transport proteins and their substrate specificities using primary sequence information. PLoS One 9:e100278
6. Yang T, Wu JC, Yan C et al (2011) Virtual screening using molecular simulations. Proteins 79:1940–1951
7. Wrenbeck EE, Faber MS, Whitehead TA (2017) Deep sequencing methods for protein engineering and design. Curr Opin Struct Biol 45:36–44
8. Kronqvist N, Löfblom J, Jonsson A et al (2008) A novel affinity protein selection system based on staphylococcal cell surface display and flow cytometry. Protein Eng Des Sel 21: 247–255
9. Yang KK, Wu Z, Arnold FH (2019) Machine-learning-guided directed evolution for protein engineering. Nat Methods 16:687–694
10. Bohr H, Bohr J, Brunak S et al (1990) A novel approach to prediction of the 3-dimensional structures of protein backbones by neural networks. FEBS Lett 261:43–46
11. Ofran Y, Rost B (2003) Predicted protein-protein interaction sites from local sequence information. FEBS Lett 544:236–239
12. Ward JJ, McGuffin LJ, Buxton BF et al (2003) Secondary structure prediction with support vector machines. Bioinformatics 19: 1650–1655
13. Petrova NV, Wu CH (2006) Prediction of catalytic residues using Support Vector Machine with selected protein sequence and structural properties. BMC Bioinformatics 7:1–12
14. Li BQ, Feng KY, Chen L et al (2012) Prediction of protein-protein interaction sites by random forest algorithm with mRMR and IFS. PLoS One 7:1–10
15. Quan L, Lv Q, Zhang Y (2016) STRUM: structure-based prediction of protein stability changes upon single-point mutation. Bioinformatics 32:2936–2946
16. Golinski AW, Mischler KM, Laxminarayan S et al (2021) High-throughput developability

assays enable library-scale identification of producible protein scaffold variants. Proc Natl Acad Sci U S A 118:1–11
17. Tahir M, Tayara H, Chong KT (2019) iRNA-PseKNC(2methyl): identify RNA 2'-O-methylation sites by convolution neural network and Chou's pseudo components. J Theor Biol 465: 1–6
18. Bloom JD, Labthavikul ST, Otey CR et al (2006) Protein stability promotes evolvability. Proc Natl Acad Sci U S A 103:5869–5874
19. Saito Y, Oikawa M, Nakazawa H et al (2018) Machine-learning-guided mutagenesis for directed evolution of fluorescent proteins. ACS Synth Biol 7:2014–2022
20. Alley EC, Khimulya G, Biswas S et al (2019) Unified rational protein engineering with sequence-based deep representation learning. Nat Methods 16:1315–1322
21. Biswas S, Khimulya G, Alley EC, Esvelt, KM, Church GM (2021) Low-N protein engineering with dataefficient deep learning. Nat Methods 18(4):389–396 https://doi.org/10.1038/s41592-021-01100-y
22. Suzek BE, Wang Y, Huang H et al (2015) UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. Bioinformatics 31:926–932
23. Crawshaw M (2020) Multi-Task Learning with Deep Neural Networks: A Survey. arXiv:2009.09796
24. Im J, Park B, Han K (2019) A generative model for constructing nucleic acid sequences binding to a protein. BMC Genomics 20:1–13
25. Ness JE, Kim S, Gottman A et al (2002) Synthetic shuffling expands functional protein diversity by allowing amino acids to recombine independently. Nat Biotechnol 20:1251–1255
26. Gupta RD, Tawfik DS (2008) Directed enzyme evolution via small and effective neutral drift libraries. Nat Methods 5:939–942
27. Engqvist MKM, Nielsen J (2015) ANT: software for generating and evaluating degenerate codons for natural and expanded genetic codes. ACS Synth Biol 4:935–938
28. Jacobs TM, Yumerefendi H, Kuhlman B et al (2015) SwiftLib: rapid degenerate-codon-library optimization through dynamic programming. Nucleic Acids Res 43:e34
29. Menéndez ML, Pardo JA, Pardo L et al (1997) The Jensen-Shannon divergence. J Frankl Inst 334:307–318
30. Bewick V, Cheek L, Ball J (2004) Statistics review 12: survival analysis. Crit Care 8: 389–394
31. Tensorflow (2017) Index @ Www.Tensorflow.Org
32. Chollet F, & others (2015) Keras. GitHub. Retrieved from https://github.com/fchollet/keras
33. Mazza D, Pagani M (2021) Automatic differentiation in PCF. Proc ACM Program Lang 5: 1–4
34. Pedregosa F, Varoquaux G, Gramfort A et al (2012) Scikit-learn: machine learning in Python. J Mach Learn Res 12
35. McKinney W (2010) Data structures for statistical computing in Python. In: Proc 9th Python Sci Conf 1, pp 56–61
36. Harris CR, Millman KJ, van der Walt SJ et al (2020) Array programming with NumPy. Nature 585:357–362
37. Abadi M, Barham P, Chen J et al (2016) TensorFlow: a system for large-scale machine learning. In: Proc 12th USENIX Symp Oper Syst Des implementation, vol 2016. OSDI, pp 265–283
38. Rao R, Bhattacharya N, Thomas N et al (2019) Evaluating protein transfer learning with tape. Adv Neural Inf Process Syst 32:9689
39. Whitehead TA, Chevalier A, Song Y et al (2012) Optimization of affinity, specificity and function of designed influenza inhibitors using deep sequencing. Nat Biotechnol 30:543–548
40. Shroff R, Cole AW, Diaz DJ et al (2020) Discovery of novel gain-of-function mutations guided by structure-based deep learning. ACS Synth Biol 9:2927–2935
41. Zhao Z, Gong X (2019) Protein-protein interaction interface residue pair prediction based on deep learning architecture. IEEE/ACM Trans Comput Biol Bioinformatics 16: 1753–1759
42. Zhang Q, Zhang M, Chen T et al (2019) Recent advances in convolutional neural network acceleration. Neurocomputing 323: 37–51
43. Yu Y, Si X, Hu C et al (2019) A review of recurrent neural networks: LSTM cells and network architectures. Neural Comput 31: 1235–1270
44. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9: 1735–1780
45. Cho K, Merriënboer B Van, Gulcehre C, et al (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. EMNLP 2014 - 2014 Conf Empir Methods Nat Lang Process Proc Conf 1724–1734

46. Vaswani A, Shazeer N, Parmar N, et al (2017) Attention is all you need, In: Advances in neural information processing systems, pp. 5998–6008
47. Brandes N, Ofer D, Peleg Y et al (2021) ProteinBERT: a universal deep-learning model of protein sequence and function. Comput Biol Chem 95:107596
48. Bergstra J, Yamins D, Cox DD (2013) Making a science of model search: Hyperparameter optimizationin hundreds of dimensions for vision architectures. Presented at the 30th International Conference on Machine Learning (ICML 2013), Atlanta, Gerorgia, June 16–21, 2013. In JMLR Workshop and Conference Proceedings 28(1):115–123
49. Raschka S (2018) Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. arXiv:1811.12808
50. Chao G, Lau WL, Hackel BJ et al (2006) Isolating and engineering human antibodies using yeast surface display. Nat Protoc 1:755–768
51. Woldring DR, Holec PV, Zhou H et al (2015) High-throughput ligand discovery reveals a sitewise gradient of diversity in broadly evolved hydrophilic fibronectin domains. PLoS One 10:e0138956
52. Woldring DR, Holec PV, Stern LA et al (2017) A gradient of sitewise diversity promotes evolutionary fitness for binder discovery in a three-helix bundle protein scaffold. Biochemistry 56: 1656–1671
53. Kruziki MA, Bhatnagar S, Woldring DR et al (2015) A 45-amino-acid scaffold mined from the pdb for high-affinity ligand engineering. Chem Biol 22:946–956
54. Kruziki MA, Sarma V, Hackel BJ (2018) Constrained combinatorial libraries of Gp2 proteins enhance discovery of PD-L1 binders. ACS Comb Sci 20:423–435
55. Stern LALA, Csizmar CMCM, Woldring DRDR et al (2017) Titratable avidity reduction enhances affinity discrimination in mammalian cellular selections of yeast-displayed ligands. ACS Comb Sci 19:315–323
56. Hasenhindl C, Traxlmayr MW, Wozniak-Knopp G et al (2013) Stability assessment on a library scale: a rapid method for the evaluation of the commutability and insertion of residues in C-terminal loops of the CH3 domains of IgG1-Fc. Protein Eng Des Sel 26:675–682
57. Zhu X, Goldberg AB (2009) Introduction to semi-supervised learning. Synth Lect Artif Intell Mach Learn 3:1–130
58. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. In: Sutton RS, Barto AG (eds) Bradford book. The MIT Press
59. Nguyen G, Dlugolinsky S, Bobák M et al (2020) Machine learning and deep learning frameworks and libraries for large-scale data mining : a survey. Artif Intell Rev 52:77–124
60. Yang KK, Wu Z, Bedbrook CN et al (2018) Learned protein embeddings for machine learning. Bioinformatics 34:2642–2648
61. Mei HU, Liao ZH, Zhou Y et al (2005) A new set of amino acid descriptors and its application in peptide QSARs. Pept Sci Orig Res Biomol 80:775–786
62. Virtanen P, Gommers R, Oliphant TE et al (2020) SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat Methods 17:261–272