ProBind
EC552 Final Project

Alyazyah Almarzooqi, Panagiota Kiourti, Thuy Pham, Leen Arnaout

Short Project Description: ProBind has the ability to model the behavior of the binding of proteins. It can produce several models, one per protein. Each model is a Convolutional Neural Network that represents the binding behavior of one protein. The model accepts one or more DNA sequences of 300 b.p. each and their reverse complements as input. The output of the model is one binding value per DNA sequence. This output scalar value is between 0 and 1 and represents how well the protein that is modeled binds to the given DNA sequences.

The tool accepts different data for training and evaluation in the following formats:

1. Training:
    a. .csv: comma-separated values

        1 file that contains alternating lines of all sample DNA sequences and their corresponding binding values. The first line in the file should start with a DNA sequence, and the next line must contain that sequence's binding value. Each base in the DNA sequence is separated by a comma. Each DNA sequence must contain exactly 300 b.p.

        Example:

        ```
        A,C,T,G,A,T,C,G,T,A, …
        0.9,
        C,T,A,G,T,A,G,C,A,C, …
        0.5
        ```

    b. .txt:

        1 file that contains alternating lines of all sample DNA sequences and their corresponding binding values. The first line in the file should start with a DNA sequence, and the next line must contain that sequence's binding value. Each base in the DNA sequence is delimited by a single white space. Each DNA sequence must contain exactly 300 b.p.

        Example:

        ```
        A C T G A T C G T A …
        ```

```
0.9
C T A G T A G C A C …
0.5
```

c. .npy files:
   i. 1 separate file of a numpy array of shape (num_sequences, 4, 300)
   ii. 1 separate file of a numpy array of shape (num_sequences, 1)

2. Testing/Evaluating cross-talk
   a. .csv: comma-separated values

      1 file that contains 2 DNA sequences with lengths ≥ 300 b.p. The second DNA sequence begins on a new line. Each base in the sequences are separated by a comma.

      Example:

      ```
      A,C,T,G,A,T,C,G,T,A, …
      C,T,A,G,T,A,G,C,A,T, …
      ```

   b. .txt:

      1 file that contains 2 DNA sequences with lengths > 300 b.p. The second DNA sequence begins on a new line. Each base in the sequences are delimited by a single white space.

      Example:

      ```
      A C T G A T C G T A …
      C T A G T A G C A C …
      ```

   c. .npy files:
      i. 1 separate file of a numpy array of shape (1, 4, num_base_pairs) for the 1st DNA seq
      ii. 1 separate file of a numpy array of shape (1, 4, num_base_pairs) for the 2nd DNA seq
   d. DNA strings through the two text boxes

The user can choose to use random data for training, already inside the "data" folder or generate more random data. When a prediction model is trained, the user can use it to find potential cross-talk between input sequences and the chosen protein. A plot is provided showing the binding values produced by the model and a threshold at which to consider cross-talk occurrence. We provide some models already trained using random data inside the "models"

folder. Lastly, the user can delete or rename a model. All functionalities described are available through buttons in the home screen of the GUI.

Major Software Components:

1. Train: It accepts .txt, .csv, .npy as described above and defines a Convolutional Neural Network as in [1] with weights of each layer randomly initialized. We split the data to 80% training and 20% testing. Using the number of epochs that the user specified, the training parses the dataset as many times as specified in the number of epochs in order to fit the weights of the neural network according to the given dataset. During training, we test the model after each one parse of the data to compute a test loss based on the Mean Squared Error between the predicted binding value and the actual one. We log the training loss of each batch in a UI window so that the user can monitor the training process. When the training is finished we plot the training and the test losses per each epoch.

2. Data Generation: Generates 3 .npy files for training that represent the forward DNA sequences, the reverse complements of those sequences, and the binding values associated with each sequence. Each base is chosen uniformly at random from the set {A, C, T, G} represented numerically by integers {0,1,2,3}, where 0 and 1 are complements and 2 and 3 are complements. These integer values are then used to represent the DNA sequence as an array of one-hot vectors. Binding values are similarly chosen uniformly at random and must be in the range [0, 1].

   Generated numpy arrays for the forward sequences and their reverse-complements are of the shape (2500, 4, 300). Generated numpy array for the binding values are of the shape (2500,). All 3 generated files are automatically saved to the "data" folder to be made available for use in training a new model.

3. Produce Bindings & Cross-Talk Evaluation: Displays a form that prompts the user for relevant inputs. User selects a trained model from a list of saved models and must choose an option for providing 2 DNA sequences to evaluate. DNA sequences may be entered manually as 2 uninterrupted strings in separate text boxes or uploaded in one of the accepted file formats (.csv, .txt, .npy). Specific formatting requirements for the different file extensions are described above in the "Short Project Description" section.

   The backend loads the selected model. Input DNA sequences are converted to .npy arrays (if not already) in order to be passed to the trained model for evaluation. Each sequence is separated into blocks of 300 b.p. which are then passed to the selected model in order to obtain a predicted binding value. A series of binding values are predicted for each input sequence.

The binding values are then plotted on two adjacent figures for user comparison. A slider controls the plotting of a constant line that represents a threshold for consideration of cross-talk existence.

4. Delete & Rename Models: Prompts user via dialog to select file for modification. If "rename model" is selected, a form field is displayed that requests the new file name. File (and new name) choice(s) is then passed to the backend in the form of an absolute file path (and input string). Backend code handles the desired file modification.

Special Instructions for Code Compilation, bit file creation:

1. Install anaconda: on windows, on mac, on linux
2. Installation and Run instructions below.

Linux	1. conda create -n probind python=3.6 pytorch torchvision matplotlib numpy seaborn 2. conda activate probind 3. python3 -m pip install fbs PyQt5==5.9.2 --user 4. git clone git@github.com:pkiourti/probind.git 5. cd probind/Final_Project_ProBind/Code/ 6. export PYTHONPATH=$(pwd) 7. cd gui 8. fbs run
MacOS	1. conda create -n probind python=3.6 2. conda activate probind 3. python3 -m pip install pytorch 4. python3 -m pip install torchvision 5. python3 -m pip install matplotlib numpy seaborn 6. python3 -m pip install fbs PyQt5==5.9.2 7. git clone git@github.com:pkiourti/probind.git 8. cd probind/Final_Project_ProBind/Code/ 9. export PYTHONPATH=$(pwd) 10. cd gui 11. fbs run
Windows	1. conda create -n probind python=3.6 2. conda activate probind 3. conda install -c pytorch pytorch 4. conda install -c pytorch torchvision

	5. conda install -c conda-forge matplotlib numpy seaborn PyQt==5.9.2 6. python3 -m pip install fbs 7. Set the environmental variable PYTHONPATH to where probind/Code is. 8. git clone git@github.com:pkiourti/probind.git 9. cd probind/Final_Project_ProBind/Code/gui 10. fbs run

References:

[1] Wang, Meng, et al. "DeFine: deep convolutional neural networks accurately quantify intensities of transcription factor-DNA binding and facilitate evaluation of functional non-coding variants." *Nucleic acids research* 46.11 (2018): e69-e69.