Final Project Report for CS 184A

Project Title: Neural Network Models on Surface Protein Levels Predictions based on Gene Expression

Student Names

Junping Luo, 91884660, junpingl@uci.edu

Sophia Luo, 24589230, fangninl@uci.edu

1. Introduction and Problem Statement

The goal of the project is to predict the modality of surface protein levels(continuous output) based on the modality of gene expression levels(continuous input). The central dogma specifies the flow of genetic information: from DNA to RNA to protein. This theory relates three variables: chromosome accessibility(of DNA), gene expression(measured by counts of RNA), and surface protein level(protein).

In our project, the portion of RNA to protein in central dogma is targeted specifically. Two different neural networks models, multi-layer perceptron(MLP) and convolutional neural network(CNN), are applied respectively to predict the surface protein level from gene expression level data. Our CNN model trained by raw data achieves a higher accuracy than MLP with a score of 0.906 out of 1.

2. Related Work

One past technical approach to predict the surface protein levels is using a transfer learning framework cTP-net (single cell Transcriptome to Protein prediction with deep neural network) to impute surface protein abundances from single-cell RNA sequencing (Zhou et al., 2020). Zhou et al. emphasized the importance of denoising data by indicating the stronger correlation they found between the abundance of protein and RNA after using denoised data. We take into account the effect of the noise generated in raw data acquisition and inherent stochasticity of translation regulation and prepossed the data through standardization and normalization.

Another approach is from the Kaggle competition. That approach used Centered Log Ratio Transformation to preprocess the data and random k-fold cross validation to validate the model. In our approach, we fit its work by trying different data preprocessing methods and applying the random k-fold cross validation for our model validation. We also chose the same loss function (MSE) and optimizer (AdamW) as it did.

3. Data Sets

The dataset we use is from a Kaggle competition “Open Problems - Multimodal Single-Cell Integration: Predict how DNA, RNA & protein measurements co-vary in single cells.” This competition provides two datasets, each of which is from cells collected with different single-cell assay technology: the first is the 10x Chromium Single Cell Multiome ATAC + Gene Expression technology (Multiome) and the second is the 10x Genomics Single Cell Gene Expression with Feature Barcoding technology technology using the TotalSeq™-B Human Universal Cocktail, V1.0 (CITEseq).

Each of the two assay technologies measures two modalities. For Multiome, it measures chromatin accessibility (DNA) and gene expression (RNA), and the CITEseq measures gene expression (RNA) and surface protein levels. Therefore, these two datasets are for predictions in two parts in the central dogma of molecular biology: DNA to RNA, and RNA to Protein.

In our project, we chose only one of these two datasets to train and do prediction: we used CITEseq dataset to predict the Surface Protein Levels given the RNA gene expression.

In this dataset, there are 6 documents included: metadata & metadata_cite_day_2_donor, train_cite inputs and targets, and test_cite & test_cite_inputs_day_2 inputs.

Name	# of Data Points	Files Size
Metadata	288544	10MB
Training Data	70988	2.5GB
Testing Data	48663	2GB

Table 1 Data Info Before Processing

The input data contains RNA gene expression levels data that have been library-size normalized and log1p transformed. The normalization is to normalize each cell by total counts over all genes, so every cell has the same total count after normalization (cite). The transformation is to logarithmize the data.

In the target data, all data points are surface protein levels for the same cells that have been dsb normalized. dsb is a method for normalizing and denoising data from CITEseq.

For convenience, the dataset we used is processed by Kaggle user senkin13 (single_cell_features). His group turned the original dataset composed of .h5 file (Hierarchical Data Formats File) into .npy and .feather file, which are more easy to use for training purposes.

Name	# of Data Points	Files Size
Metadata	119651	2.44 MB
Training Data	70988	584 MB
Testing Data	48663	374 MB

Table 2 Data Info After Processing

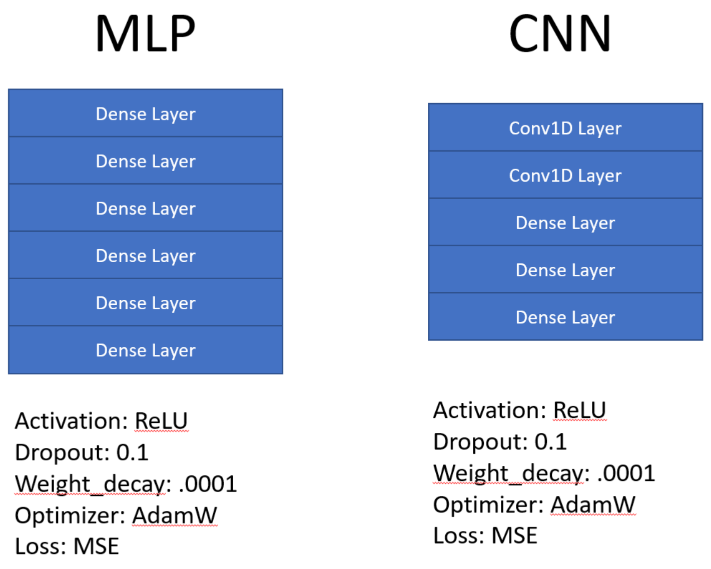
Throughout the data extraction and rearrangement, the file size significantly decreased, which would shorten the time for further training and then improve the efficiency.

4. Description of Technical Approach

In this project, we compared two different neural network models: MLP and CNN.

MLP stands for multi-layer perceptron, which is a type of artificial neural network that uses multiple layers of interconnected nodes to learn and make predictions based on input data. The MLP model uses a series of algorithms to iteratively adjust the weights and biases of the nodes in each layer, allowing it to learn and adapt to new data over time.

CNN stands for convolutional neural network, which is a type of artificial neural network designed specifically for image and video analysis. The CNN model uses a series of convolutional layers, which apply a series of filters to the input data to detect specific features and patterns.



Activation: ReLU
Dropout: 0.1
Weight_decay: .0001
Optimizer: AdamW
Loss: MSE

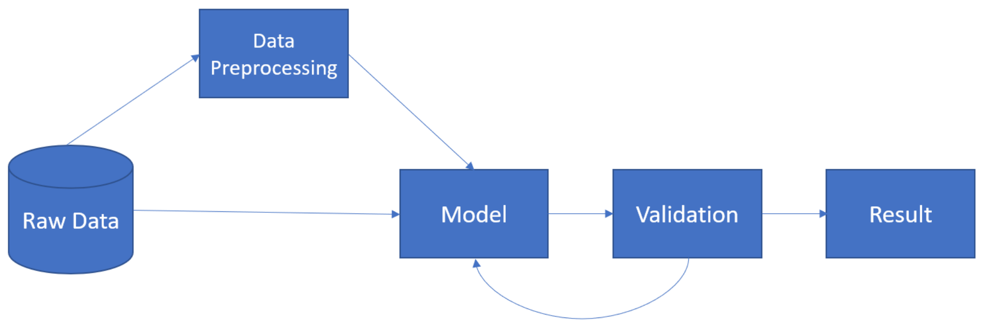
Activation: ReLU
Dropout: 0.1
Weight_decay: .0001
Optimizer: AdamW
Loss: MSE

We also applied data preprocessing by using scikit-learn, a free library that provides many tools for machine learning and statistical modelings.

For data preprocessing, we tried two methods: standardization and normalization. Detailed application of these two methods on our project will be in part 6: Experiments and Evaluation.

Data standardization is the data preprocessing step that involves transforming the values of a given dataset to have a mean of zero and a standard deviation of one. This process is often used to ensure that all the features in a dataset are on the same scale and have the same units of measurement, which can improve the performance of many machine learning algorithms. Data standardization can be performed using a variety of techniques, such as min-max scaling, mean normalization, and z-score scaling.

Data normalization is the data preprocessing step that involves transforming the values of a given dataset to have a common range or distribution. This process is often used to ensure that all the features in a dataset are on the same scale and have the same units of measurement, which can improve the performance of many machine learning algorithms. Unlike data standardization, which transforms the values to have a mean of zero and a standard deviation of one, data normalization typically transforms the values to have a range between 0 and 1 or -1 and 1.



The flow chart figure above shows the process of our approach. Notice that two arrows start from the raw data, which means that both preprocessed data and raw data will be used to train the models for comparison.

5. Software

Code we wrote

Name	Description	Input	Output	Comment
Data Loading	This part of the code script loads the data from the local data path.	Hardcoded file path string.	File objects that can provide data when needed.	Pandas and numpy were used to load data from two different types of files (.feather and .npy).
Data Analysis and Preprocessing	This part of the code displays the value of the mean, median, and standard deviation of all of the features in raw data by using a box plot.	Raw data.	Box plot that shows mean, median, and standard deviation of all the features.	Library matplotlib.pyplot is used.
Data Preprocessing	This part of the code preprocesses the raw data by applying standardization and normalization.	Raw data.	Standardized data and normalized data.	Library sklearn.preprocessing is used.
Model Construction	This part of the code constructs two models: MLP and CNN.	N/A	N/A	Library Tensorflow is used.

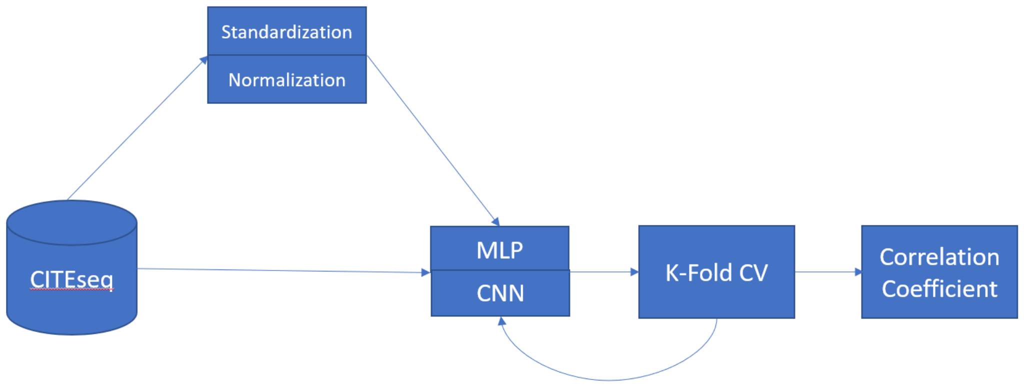
Data Preprocessing Method Selection	This part of the code uses the MLP model to evaluate the data (raw data, standardized data, normalized data) for the training.	Raw data, standardized data, normalized data.	Selected data from the preprocessed and un-preprocessed.	
Model Training, Validation, and Evaluation	This part of the code trains the models and applies the k-fold CV method for validation, and correlation score as evaluation score.	Training data.	Heatmap plot that displays score of models trained with different parameters.	Using k-fold CV, the training data also serves as testing data.

Code from others (w/ references)

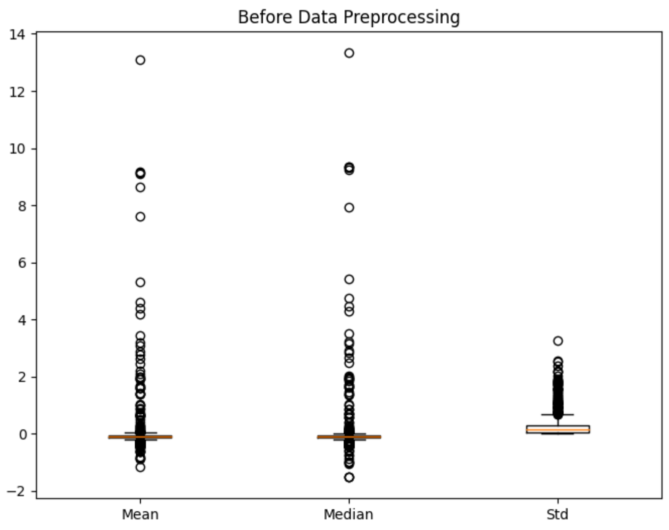
Name	Description	Input	Output	Ref.
Model Evaluation Method: correlation score	This method calculates Pearson's Correlation Coefficients as the score.	y_true, y_pred	Correlation Score	https://www.kaggle.com/competitions/open-problems-multimodal/discussion/366453
Data Generator	This class provides a method to easily get data in a given batch size.	N/A	N/A	
K-Fold Validation	This method implements k-fold cross validation method by given neural network and dataset.	Training data, model, other hyperparameters	Y_pred predicted by models trained with validation data.	

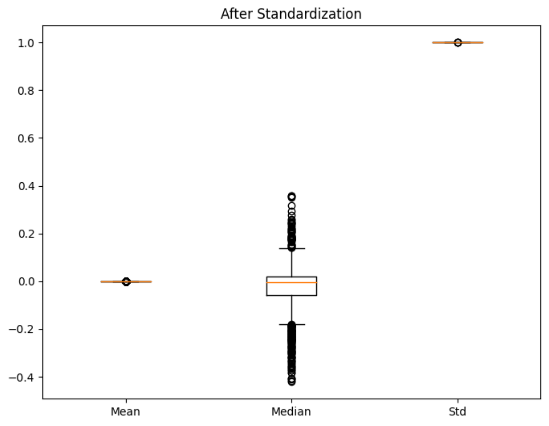
6. Experiments and Evaluation

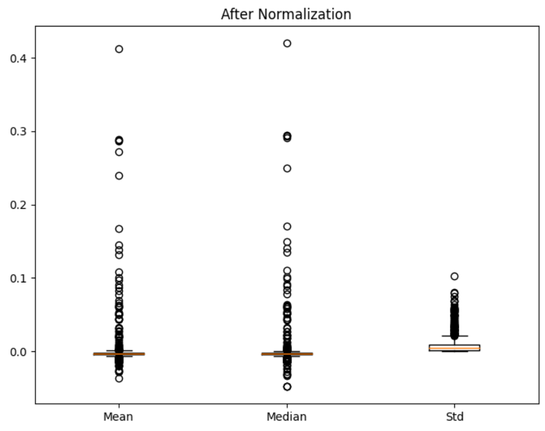
In this project, our group did experiments (data standardization and normalization) in data preprocessing, and used K-fold cross-validation as our evaluation method. For the final result evaluation, we use Pearson's Correlation Coefficient as the score of the model's performance, which is corresponding to the evaluation score used by the Kaggle competition using this dataset.



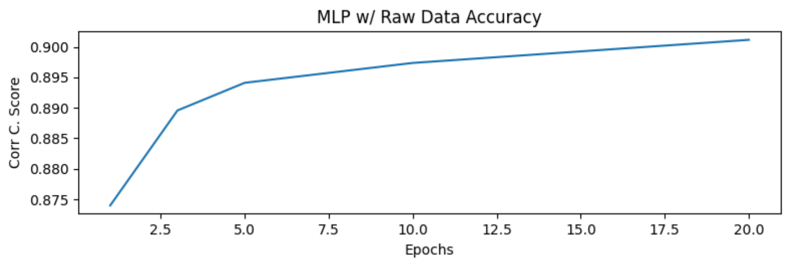
We used sklearn.preprocessing to transform and standardize the data to improve its quality. Below is the distribution of the mean, median, and standard deviation of each of the features in data points.

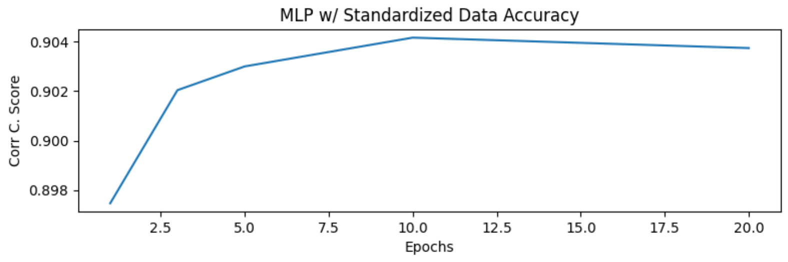


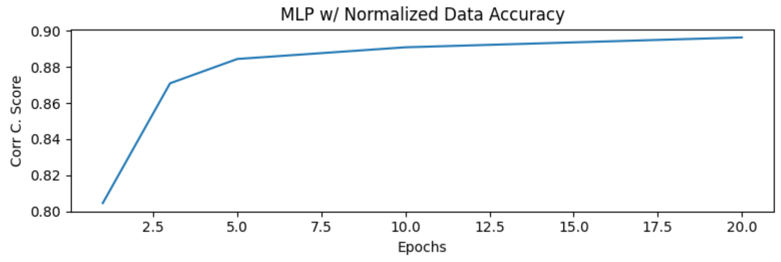




Using the standardized data, we found that the accuracy of the model is increasing when epoch <= 10, and decreasing after then. Using the normalized data, we found that the accuracy of the model is increasing, but its performance is worse than the model trained with raw data as a whole.

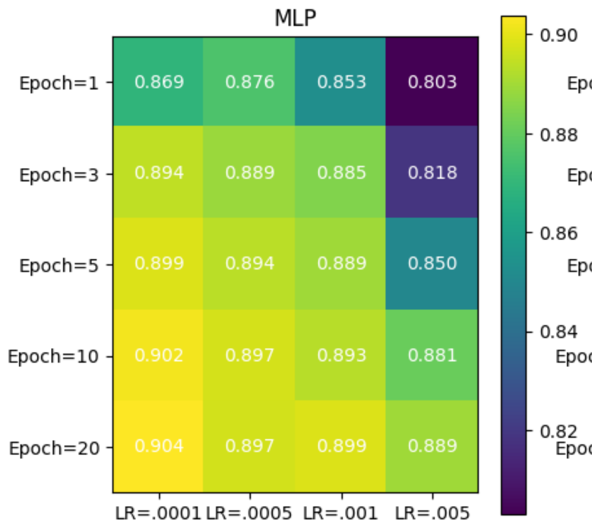


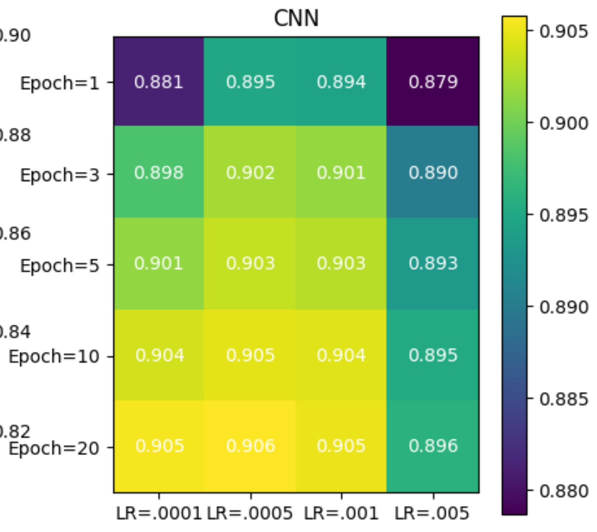




We observed a possible tendency that the accuracy of the model trained with raw data will increase as the number of epochs gets bigger but not for the model with standardized data. Therefore, we decided to choose raw data as input data for training as it benefits the future training with larger numbers of epochs.

Then, we trained both of the models with two different hyperparameters: number of epochs and value of learning rate. Epoch: [1, 3, 5, 10, 20]; Learning Rate: [.0001, .0005, .001, .005]





As the figures above shown, in the MLP model, the best result was 0.904, with epoch of 20 and learning rate of 0.0001. For the CNN model, the best result was 0.906, with epoch of 20 and learning rate of 0.0005.

Also, the minimum score of CNN model (0.879) is higher than that of the MLP model (0.803), and the score range of CNN model ([0.879, 0.906]) is narrower than that of MLP model ([0.803, 0.904]).

Therefore, with the selected hyperparameters, the CNN model is slightly better than the MLP model in this experiment. Plus, the CNN model is less sensitive to the setting of hyperparameters than the MLP model is.

7. Discussion and Conclusion [at least ½ a page]

When choosing epoch number and learning rate, it's more time efficient to test and record the accuracy of every recombination of different epoch number and learning rate for both MLP and CNN model and visualized in heatmap. The heatmap is straightforward to provide the trend of accuracy as different parameters change. Also, though not applicable to our project, trying every recombination can avoid missing the best recombination when the relationship between certain parameters and accuracy is not simply linear(e.g. accuracy may increase and then decrease as the number of epochs gets larger).

It's expected that as the epoch number increases to be very large, the accuracy of the model will increase followed by decrease. However, for both MLP models trained by raw and normalized data, accuracy doesn't seem to fall as epoch number increases from 1 to 20. It is speculated that the epoch number is not increased to be large enough to reach the inflection point of accuracy.

It's expected that pre-processed data can yield higher accuracy, but the MLP model trained by raw data actually yields higher accuracy. The possible reason for this result is that the data provided is already library-size normalized and logarithmized. Our data preprocessing methods, standardization and normalization, don't work well for our specific dataset or models.

One limitation of the current approach is the low computing power of the GPU, which prevents us from trying larger training epochs and implementing more complex models to find better hyperparameters and models to address the problem. Another limitation is that the data we use for the project is not the original data in .h5 format but data processed by a team of Kaggle in .npy and .feather format.

In the future, the training efficiency can be improved through selecting the most representative features and compressing the dimension. More information regarding single-cell data and translation should be analyzed to assist the feature selection process. Also, additional models can be explored and tested to find the most appropriate one fitting single-cell data analysis problem.

8. Individual Contributions

Junping Luo: During this project, I oversaw the overall progress. I set up the environment locally and wrote the major fundamental part of the code, like data preprocessing, models constructions, validations, and hyperparameters selections. For the final report, I wrote part 4, 5, 6 independently, and wrote part 3 with Sophia Luo.

Sophia Luo: For this project, I searched for relevant background biology principles and theory to demonstrate the target problem. I searched for previous approaches targeting the same problem, from which I summarized their main approaches and insightful points. I tested the code script and made improvements with Junping Luo. For the final report, I was responsible for part 1, 2, 7, as well as 3 with Junping Luo.