ENPM808A Final Project Report

Govind Ajith Kumar
UID: 116699488

Introduction

The program is an exercise in developing a machine learning model that can Predict the multiple targets of the Mechanism of Action (MoA) response(s) of different samples, each identified by a different sig_id, given various inputs such as gene expression data and cell viability data.

The model is tested by finally evaluating the logarithmic loss function and applying this to each drug-MoA annotation pair. In this report, we go over the model used, the reasons for selections, and how the hyperparameters were chosen, followed by the results, plots, and conclusions.

Python Libraries and other requirements

- While the program can run on the CPU, it is much faster on GPU. Hence, please have your GPU/CUDA enabled.
- GPU Used: NVIDIA GTX950M
- Python 3.x
- Some of the libraries that are required are:
    a. PyTorch
    b. seaborn
    c. NumPy
    d. sklearn
    e. matplotlib
    f. pandas

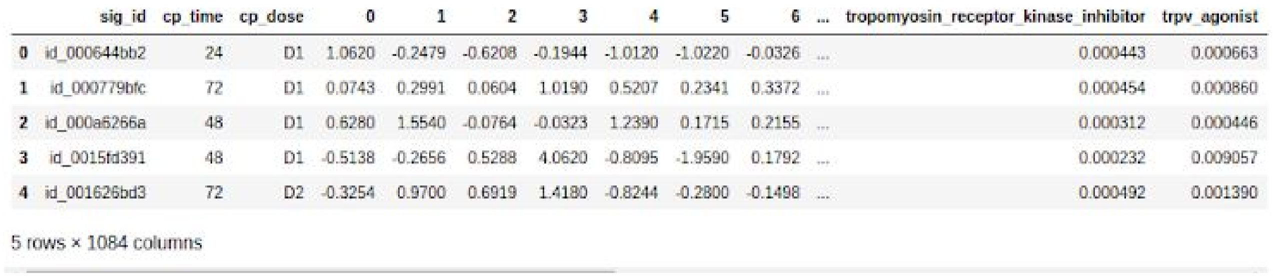
Approach used

I decided to use a neural network for the assignment. We are going to leverage open libraries such as PyTorch to develop the neural network and configure them to predict the MoAs. PyTorch was used to do this because of the presence of online support and the wide array of tool sets at its disposal. The steps used to solve is as follows:

1. Prepare the dataset
2. Select the model
3. Decide the Hyperparameters
4. Train the model
5. Test for the out of sample error
6. Plot the validation loss, training loss, and the best-recorded loss

Dataset preparation

The dataset is prepared by first separating the gene expression and cell viability columns. Now, on the Gene expression data, since our dimensions are really high, we can resort to using PCA for dimensionality reduction but is still able to capture the characteristics of the data. Now, this can be done by choosing a random dimension and having the same random state as before. After performing the PCA, we fir the PCA transform. We then split the data we have into training and test set. For readability, they are all converted to a pandas data frame format.
This process is repeated for the Cell viability data.
Now, we set the desired threshold to calculate the VarianceThreshold. As per the math, all the Features with a training-set variance lower than this threshold will be removed. This is followed by combining the training and testing features
We remove the unnecessary columns that do not contain numerical entries such as **'sig_id'**, **'cp_type'**.
Now we merge the training and training targets scored columns.
This is followed by removing the rows with **cp_type** as **ctl_vehicle** since control perturbations have no MoAs. The final train dataset after all the preparation looks like this:



	sig_id	cp_time	cp_dose	0	1	2	3	4	5	6	...	tropomyosin_receptor_kinase_inhibitor	trpv_agonist
0	id_000644bb2	24	D1	1.0620	-0.2479	-0.6208	-0.1944	-1.0120	-1.0220	-0.0326	...	0.000443	0.000663
1	id_000779bfc	72	D1	0.0743	0.2991	0.0604	1.0190	0.5207	0.2341	0.3372	...	0.000454	0.000860
2	id_000a6266a	48	D1	0.6280	1.5540	-0.0764	-0.0323	1.2390	0.1715	0.2155	...	0.000312	0.000446
3	id_0015fd391	48	D1	-0.5138	-0.2656	0.5288	4.0620	-0.8095	-1.9590	0.1792	...	0.000232	0.009057
4	id_001626bd3	72	D2	-0.3254	0.9700	0.6919	1.4180	-0.8244	-0.2800	-0.1498	...	0.000492	0.001390

5 rows × 1084 columns

Model

We are going to solve the problem by developing a 15 layer neural network that can help us with a wide variety of features we have in this problem. We are going to be using a simple feed-forward network. Using PyTorch we can easily model this.

We are going to pass this neural network model to our training function along with an optimizer and scheduler.

It is important to note that we have a lot of features and our dimensions are really large. In order to tackle this problem, we are going to deploy regularization techniques. The techniques of *regularization* that we are going to start with are:

Batch Normalization:

Applying batch normalization is done to standardize the input for each mini-batches and will help reduce the number of epochs for which the training is done. This limits the covariate shift (*this is the value by which the hidden layer values shift around*) and allows to learn from a more stable set of data. Sometimes, it also allows for a higher learning rate. This is also used for regularization and helps reduce overfitting. Generally, if batch normalization is used, you can use a smaller dropout, which in turn means that lesser layers can be lost in every step.

Using dropout layers:

For regularization purposes, the dropout is set by setting a probability. Random neural networks are picked at a probability, say ***p***, or dropped at a probability of ***1-p***. This is essential to prevent overfitting of the model and also reduces the computation time. A fully connected neural network, if run without dropout will start forming dependencies between each other and this can lead to over-fitting, which is not favorable.

PCA:

Besides this PCA is also applied during the dataset preparation, which can help with dimensionality reduction and can run the program faster. Doing PCA reduces the dimensions, but it can still capture the characteristics of the data.

We are going to be running this on an NVIDIA GTX950M GPU because of the exponentially fast running times as compared to running on a CPU.

Some of the other most important features linked to the training function are

In this training function, we are using Adam optimizer. This is a standard optimizer and is widely used because it combines the advantages of the Adaptive gradient algorithm and the root-mean-square propagation. Basically, it does not stick to one learning rate and adapts it to the problem. It is widely known to offer good results really fast. The Adam optimizer takes in a learning rate (here, 0.001) and a factor for weight decay. When training neural networks, it is common to use weight decay where after each update, the weights are multiplied by a factor. Here the factor is set to 0.00001.

Here we are also using a learning rate scheduler. We use a learning rate scheduler to converge to the lowest loss faster. This is also seen to provide higher accuracy. I tried using a StepLR scheduler. This works by decaying the learning rate of each parameter by a fixed constant called gamma and repeats the process every epoch. However, the trained model was not as good as the one that used OneCycleLR. OneCycleLR bounces the learning rate between a fixed maximum value and a fixed minimum value, which was set to 0.001, which is much lower than the maximum value, which was set to around 0.01.

Multiple loss functions are available through PyTorch, but we are using the Binary Cross Entropy loss with a sigmoid layer. After a trial and error with other losses, this was seen to offer better results.

Hyperparameters

Some of the most important hyperparameters that are used are shown in the table below:

Hyperparameter	Value	Notes
Batch size	64	Number of training samples used in one training iteration
Learning rate	0.001	It controls how quickly or slowly a neural network model can learn a model or a problem.
K folds	3	Creating a 3 fold multilabel stratified k fold
max_epochs	15	It is the number of times a training vector is used to update the weights. The value can be set through trial and error.
weight_decay	0.00005	When training neural networks, it is common to use weight decay where after each update, the weights are multiplied by a factor slightly less than 1
num_features	880	This is the number of features corresponding to the columns in the feature_columns
num_targets	206	This is the number of targets corresponding to the columns in the target_columns
hidden_size	1024	Can set by trial and error. This is the number of neuron layers in the middle.

Validation

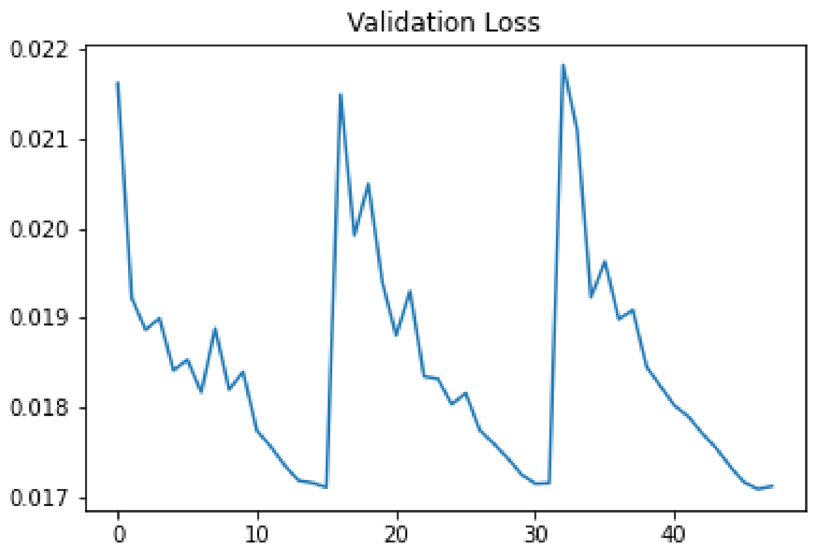
In each fold the trained model is validated by the batch that was not used to train, this process is called validation and the dataset that is used for this purpose is called the validation set.

Results

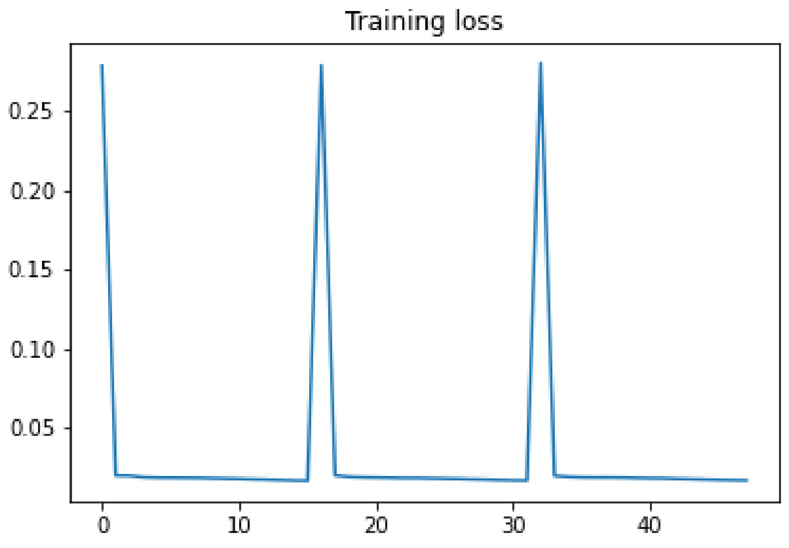
Various combinations were run and we finally obtained a cross-validation error over the test set was obtained. It was observed to be **0.015734850054263057**.

Graphs

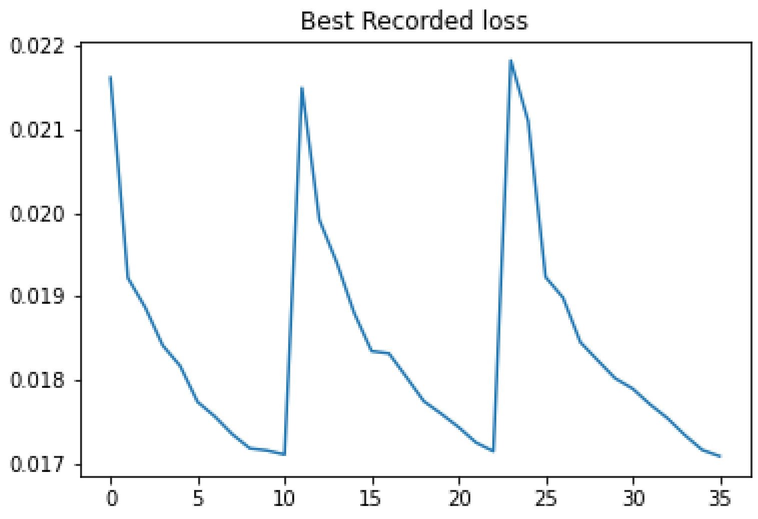
The plot for the validation loss over every Kth fold can be seen in the graph below:



The plot for the training loss over every Kth fold can be seen in the graph below:



The plot for the best-recorded loss over every Kth fold can be seen in the graph below:


Best Recorded loss