ENPM 808A
Machine Learning

Final Project

Mechanisms of Action (MoA) Prediction

Govind Ajith Kumar - 116699488

Introduction

- The programme 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.

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.
- Python 3.x
- GPU Used: NVIDIA GTX950M
- 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

1. 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.
2. After performing the PCA, we fir the PCA transform.
3. We then split the data we have into training and test set. For readability, they are all converted to a pandas dataframe format.
4. This process is repeated for the Cell viability data.
5. 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.
6. This is followed by combining the training and testing features
7. We remove the unnecessary columns that do not contain numerical entries such as **'sig_id'**, **'cp_type'**.
8. Now we merge the training and training targets scored columns.
9. This is followed by removing the rows with **cp_type** as **ctl_vehicle** since control perturbations have no MoAs.

Model Used

- 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.
- These layers are
  - BatchNorm1d: This applies batch normalization over a 2D input.
  - Dropout: For regularization purposes
  - Weight_norm: For weight normalization (faster than batch normalization)
- 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.

Regularization

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.

Regularization contd...

- **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 overfitting, 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.

Hyperparameters

These parameters define the model and can be tuned in many different ways.

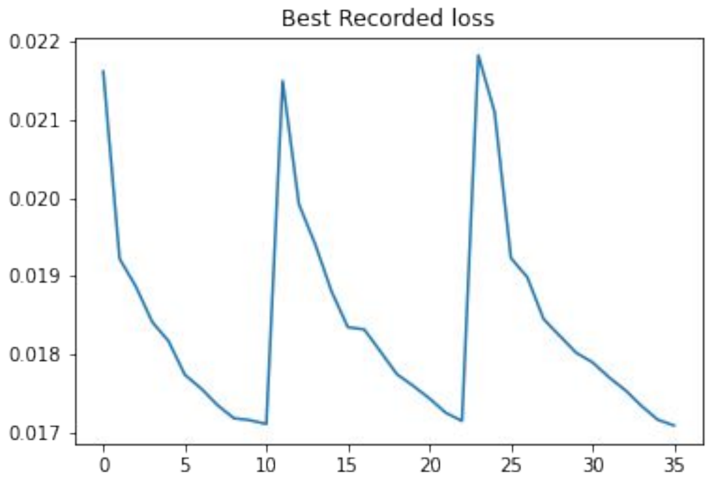
The detailed explanation of each of the parameter is provided in the project report.

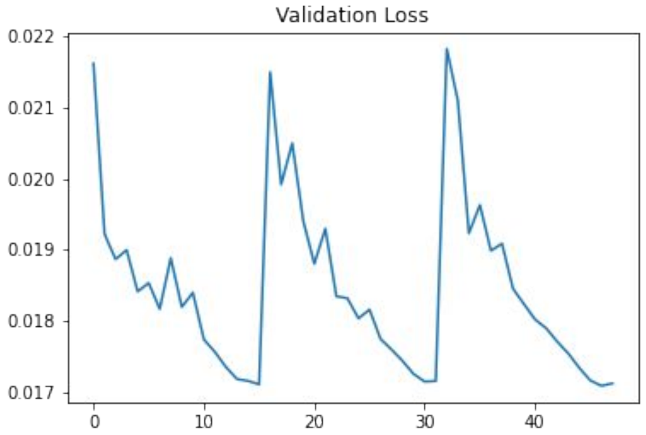
Hyperparameter	Value
Batch size	64
Learning rate	0.001
K folds	3
max_epochs	15
weight_decay	0.00005
num_features	880
num_targets	206
hidden_size	1024

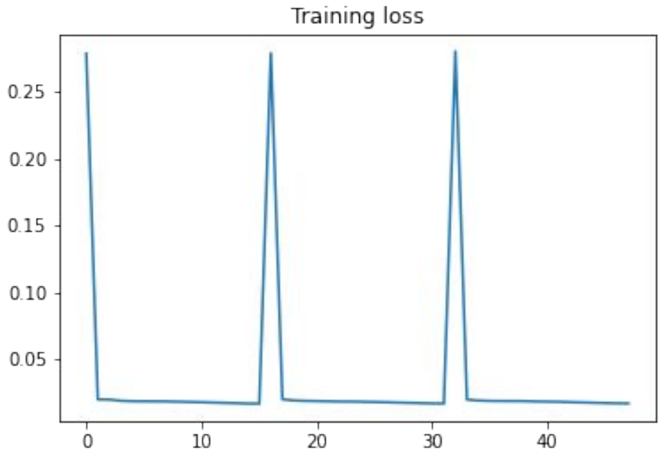
Results and Graphs

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**.

The following learning curves depict the reduction in loss over each fold.


Best Recorded loss


Validation Loss


Training loss