MATH569 Project: An Application of Deep Neural Networks to Mechanisms of Action of Drugs

Olugbenga Abdulai
Master of Data Science Program
Illinois Institute of Technology

December 4, 2020

Background

Mechanism of action (MOA) of a drug refers to how the drug works on a molecular level in the body. It is the biochemical process through which a drug produces its effect. With the advent of technological advancements in Pharmacology, we are able to better understand the cause and effect relationships between drugs and cell responses to them. Thus, we can treat a sample of human cells, obtain cellular responses, and compare them with gene expression data and cell viability data of drugs with known mechanisms of action. Where data is involved, statistical techniques can be leveraged upon and this is the goal of this study.

Problem Statement

Given information on cell viability and gene expression of a drug, we aim to predict its mechanisms of action. If highly successful, this may help advance drug discovery and understanding.

Methodology

Dataset

The nature of the data [1] used for this study makes it suitable to apply deep neural networks to this task. The dataset features roughly 5000 unique drugs profiled multiple times at various dosage times and levels such that there are at least six entries for each drug. The data provided is split already into training and test sets with some supplementary target (or response) values termed “non-scored targets”. The data is wide with 875 features, 206 targets, and 403 additional non-scored targets. There are 772 gene expression features, each denoted by "g-". Each gene feature represents the expression of one particular gene. Hence, there are 772 individual genes being monitored. There are 100 cell viability features, each denoted by "c-". Each cell feature represents the viability of one particular cell line. Therefore, there are 100 individual cell lines being monitored. The task is a multilabel classification machine learning problem because a drug may have multiple MOA annotations. Each of the 206 targets are labeled as binary holding a value of 1 if that cell is activated by the drug and 0 otherwise.

Exploratory Data Analysis

The results of the exploratory data analysis performed on the dataset are discussed here. The training data consists of 23814 examples and the test set has 3982 examples to be predicted. The features in the training and test data can be summarized as follows:

- Cp_type - indicates samples treated with a compound or with a control perturbation
- Cp_dose - treatment dose (high or low)
- Cp_time - treatment duration (24, 48, or 72 hours)
- Gene information - 776 features
- Cell viability information - 100 features

The actual names of the genes and cell lines have been hidden by the providers of the dataset to prevent a hand-crafted solution.

For the cp_type feature, control perturbations have no MOA values (targets) hence we remove those examples from the dataset. Figure 1 shows the count distribution of the cp_type feature. Figure 2 shows the count distribution of cell activations for the drug samples. The insight here is that there are 9367 examples in the training data with all zeros as targets. It also indicates that for the majority of the training examples, only one (roughly 50% cases) or two cells are activated (targets holding a value of 1).

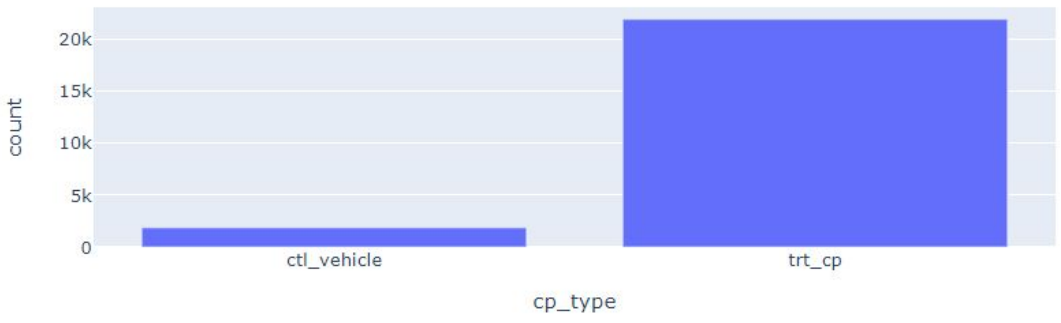


Figure 1: Distribution of cp_type feature

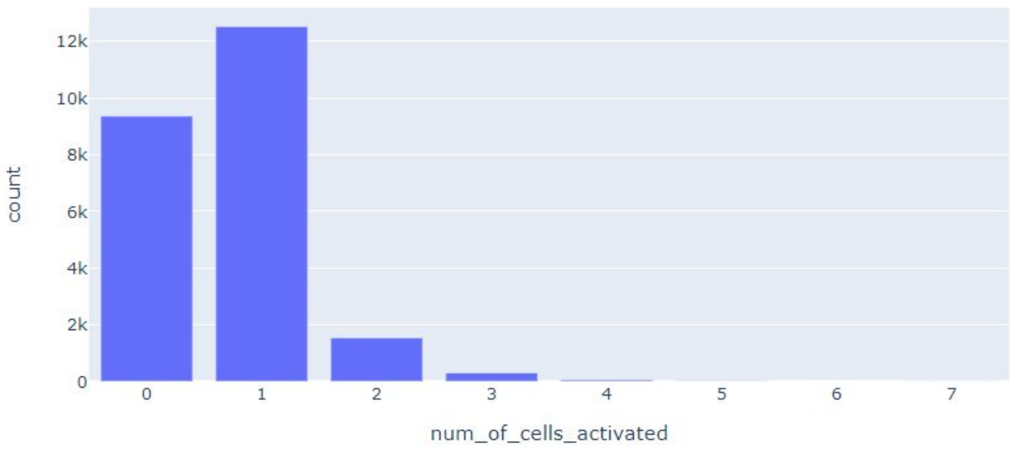


Figure 2: distribution of number of cells activated for drugs in the training set

It is important to note that there are groups of categories of the target variable. From exploring the target names we see that there are inhibitors, antagonists, agonists, activators, and so on. Thus the targets consist of subsets (or classes) of each of these categories. These are pharmacological terms so we won't go into the details of what they mean. Although it is important to recognize the frequency of these various categories. Figure 3 shows how the categories are distributed in the training data.

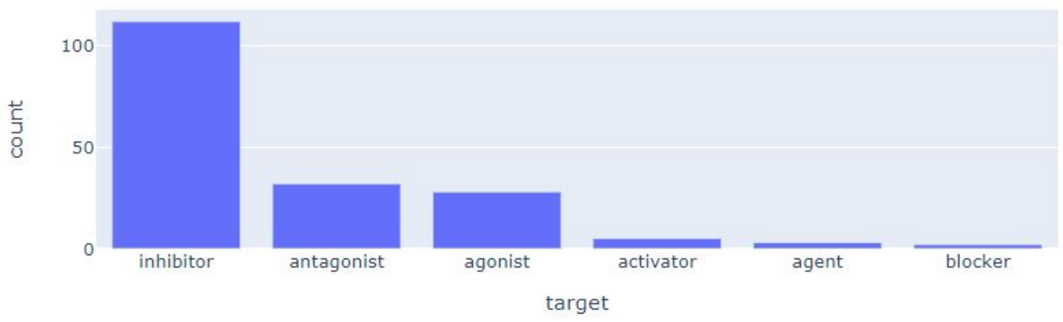


Figure 3: Distribution of target categories

We can see that inhibitors are the dominant category followed by antagonists and agonists. Figure 4 is a plot of the distribution of the target classes and shows that nfkb_inhibitor (832 times) and proteasome_inhibitor (726 times) are activated frequently and dominant. Also there are target classes (not shown) that have very few frequency of activations (as low as 1). The median of frequency of activations is approximately 39 which when compared to the maximum (832) shows that the data is severely imbalanced.

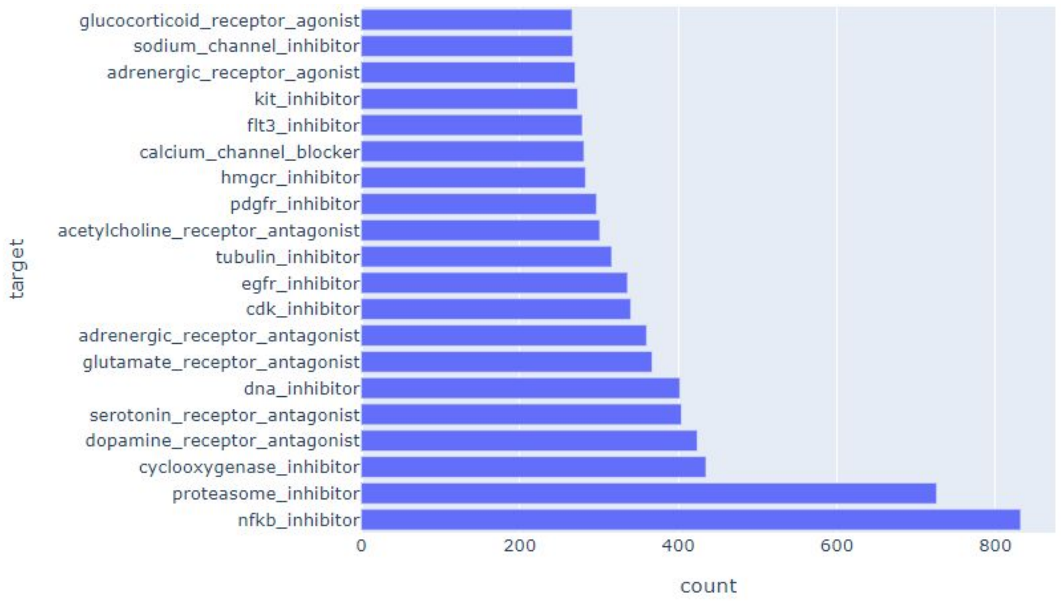


Figure 4: Count distribution of target classes

We perform some correlation analysis on the data, starting with the targets. The target classes with very high correlation with another target class (>0.9) are proteasome_inhibtor, nfkb_inhibitor, kit_inhibitor, pdgfr_inhibitor, and flt3_inhibitor. We then proceed to look at correlations in the features. For the cell viability features, out of 4950 possible pairs, we find that 4187 unique pairs are highly positively correlated. There are also no negative correlations. For the gene expression features, out of a possible 297,606 pairs of features, there are 562 pairs of highly positively correlated (>0.7) features and 342 pairs of highly negatively correlated (<-0.7) features. Figures 5 and 6 show histogram plots of a few examples of cell viability and gene expression features respectively.

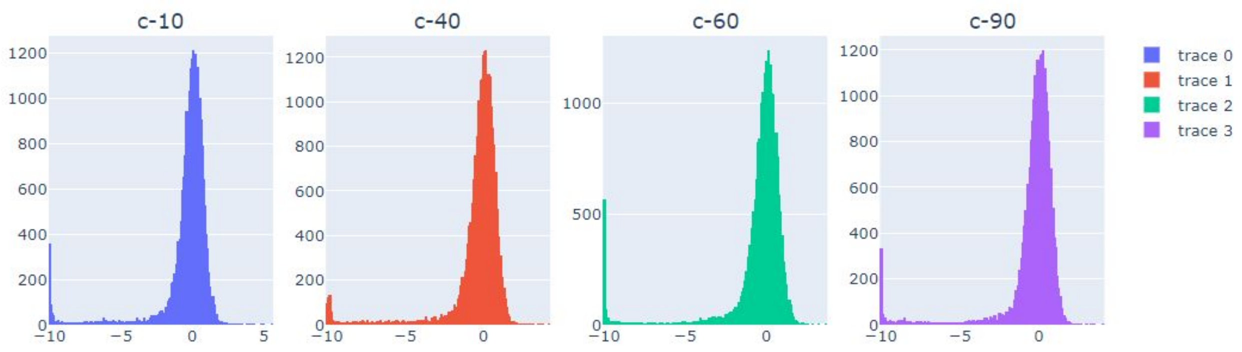


Figure 5: Histograms of a few cell viability features

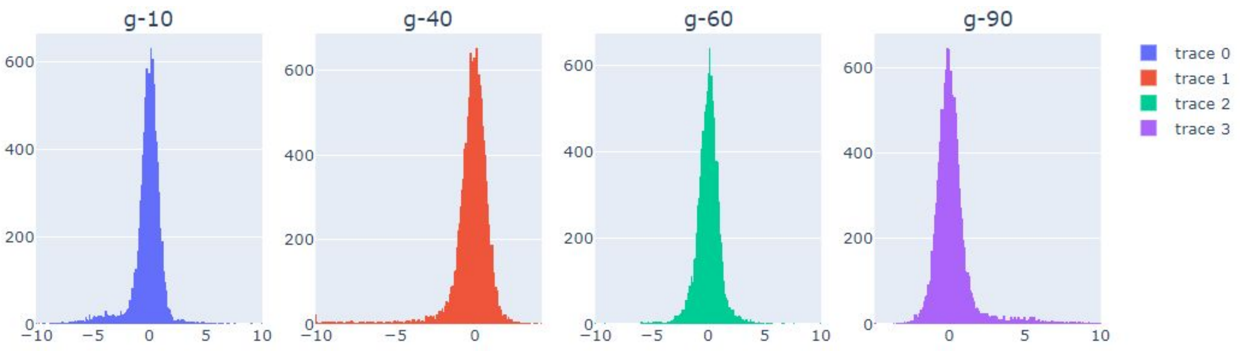


Figure 6: Histograms of a few gene expression features

Both groups of histograms look fairly normally distributed. The distribution of the cell viability features have a spike around -10 to -9 but we leave this to the model to pick up the pattern.

Feature Engineering

As a pre-processing step, the categorical features, cp_type and cp_dose are one-hot encoded while the numeric features are standardized and scaled. Because of the large number of features and correlations between them we apply dimensionality reduction with principal component analysis (PCA). We will show later that in fact, the neural network can handle this situation of wide data very well without necessarily needing to reduce the dimensions.

PCA is a statistical technique that involves reducing the dimensions of a dataset, hence improving the interpretability and at the same time maximizing the explained variance. The new features, called principal components, are obtained by solving an eigenvalue and eigenvector problem and they are the features that maximize the explained variance (hold the most statistical information). We apply PCA to the gene features and cell features separately so we can track the feature type (gene or cell) after the reduction. For the gene features, we choose 100 principal components and for the cell features we use 20 components. These “strong” PCA features obtained are appended to the existing features.

For feature selection, we use variance thresholding. Variance thresholding involves removing features whose variance is below a certain specified threshold. By doing this, we keep only features that have predictive power. After these feature engineering steps, the final training data has 1546 features to be fed into the model.

Modeling

We employ deep neural networks for this study. The central idea of neural networks [2] is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features. The result is a powerful learning method, with widespread applications in many fields. Another angle to look at neural networks from is the idea of the perceptron. A perceptron can model any simple binary boolean gate and when combined can model more complex boolean functions. Therefore, a neural network can be thought of as a cascade of many perceptrons and thus is capable of computing very interesting nonlinear functions. A neural network is a two-stage regression or classification model, typically represented by a *network diagram* [2] as in Figure 7.

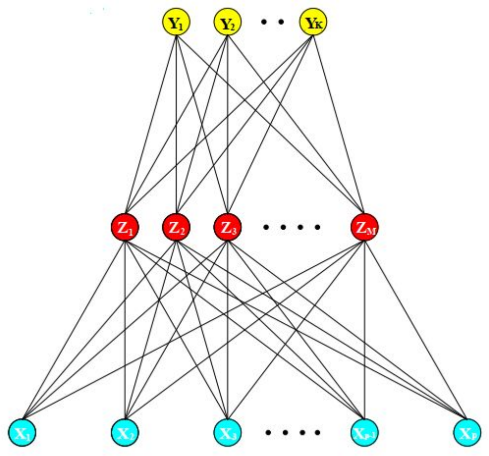
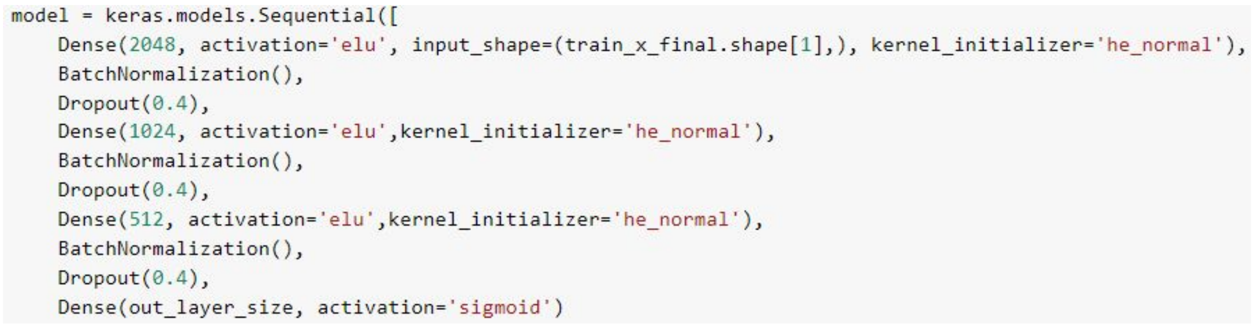


Figure 7: Schematic of a single hidden layer, feed-forward neural network.

A neural network with more than one hidden layer is termed a deep neural network. The idea is that deep neural networks in most cases can capture more complex nonlinearities down the line as we go from input to output of the neural network. Neural networks can be also classified as wide networks. In some applications, wide networks may be preferred. It can be proven statistically that a network with fewer number of neurons than the minimum required to model the function cannot model that function hence the need to go deep, wide, or wide and deep.

We define the deep net as shown in Figure 8, using the Tensorflow and Keras libraries. Various architectures are tried and it is discovered that wide networks fared better than much deeper ones, hence the use of large numbers of neurons per layer (2048, 1024, 512). We use an exponential linear unit (ELU) activation function and the “He_normal” initialization which are chosen after testing various activation functions and initializations.



```
model = keras.models.Sequential([
    Dense(2048, activation='elu', input_shape=(train_x_final.shape[1],), kernel_initializer='he_normal'),
    BatchNormalization(),
    Dropout(0.4),
    Dense(1024, activation='elu',kernel_initializer='he_normal'),
    BatchNormalization(),
    Dropout(0.4),
    Dense(512, activation='elu',kernel_initializer='he_normal'),
    BatchNormalization(),
    Dropout(0.4),
    Dense(out_layer_size, activation='sigmoid')
```

Figure 8: Model architecture definition

Training a neural network assumes the training data are similarly distributed but in reality the mini-batches used may have a different distribution (covariate shift) and hence the need to normalize within batches. This is the idea of batch normalization which we apply after each dense layer.

Neural networks are overparameterized models and need adequate regularization to prevent overfitting. Dropout is an effective regularization technique that involves “dropping” or turning off each neuron at each training iteration with a certain probability. We use this technique in the network in Figure 8 with a probability of 0.4, thus, we dropout neurons at a rate of 40%.

The output layer uses a sigmoid activation function since the individual targets are binary and independent of each other. If they had some dependence (probabilities summing to one for all classes), then we would use a softmax activation function instead.

Training

Here, we highlight some implementational details in training the neural network. The evaluation metric used in this task is the cross-entropy or log loss function [3] but for a multilabel classification problem as defined in equation 1.

$$\text{score} = -\frac{1}{M}\sum_{m=1}^{M}\frac{1}{N}\sum_{i=1}^{N}\left[y_{i,m}\log(\hat{y}_{i,m}) + (1-y_{i,m})\log(1-\hat{y}_{i,m})\right]$$

Equation 1: Overall Log loss function

where:

- N is the number of observations in the test data (i=1,…,N)
- M is the number of MoA targets (m=1,…,M)
- $y^{\wedge}_{i,m}$ is the predicted probability of a positive MoA response for a drug sample
- $y_{i,m}$ is the ground truth, 1 for a positive response, 0 otherwise
- log() is the natural (base e) logarithm

The log loss heavily penalizes confident and wrong predictions thus takes into account the probabilities unlike less strict loss functions such as the zero-one loss function.

For model validation, we employ cross-validation, a statistical technique that allows for evaluating models using the data efficiently. We shuffle and split the data into 7 folds and use 2 seed values. Hence a total of 2*7 = 14 models trained. For each trained model, we obtain an out-of-fold loss by evaluating on the current validation set (the 7th split).

The hyperparameters used are as follows:

- K (cross-validation) = 7
- Batch size = 128
- Epochs = 50
- Adam optimizer (learning rate = 0.0005)

- Label smoothing (0.001)

Because the log loss penalizes heavily confident and wrong predictions, we do not want our model to become too confident in its predictions. This is where label smoothing helps. It relaxes the confidence on predictions and helps the model train around mislabeled data and by extension, improves the performance of the model.

We use callbacks to monitor the progress of model training. A summary of callbacks used in the model are listed below.

- Reduce learning rate on plateau
- Early stopping
- Model checkpointing

Reducing the learning rate gradually generally helps to optimize the convergence of model training. Early stopping is a technique used to optimize training by exiting the training process once the model plateaus for a set number of consecutive steps. The best weights are restored after the exit. Model checkpointing saves the best model after training completes.

Results and Discussion

Before discussing the results from the final model, let us address the supplementary non-scored targets mentioned in earlier sections. Since we have much more targets in the non-scored targets we try to capitalize on these with transfer learning. We train on the non-scored targets and use the weights to train on the actual targets. This yielded an overall log loss of approximately 0.0167 slightly worse than that obtained without using the non-scored targets so we decided not to use this approach.

After training with the regular targets for 2 seeds and 7 folds, we obtain best results of log loss as approximately 0.0166. The neural network also did not overfit as seen in the learning curves of Figure 9.

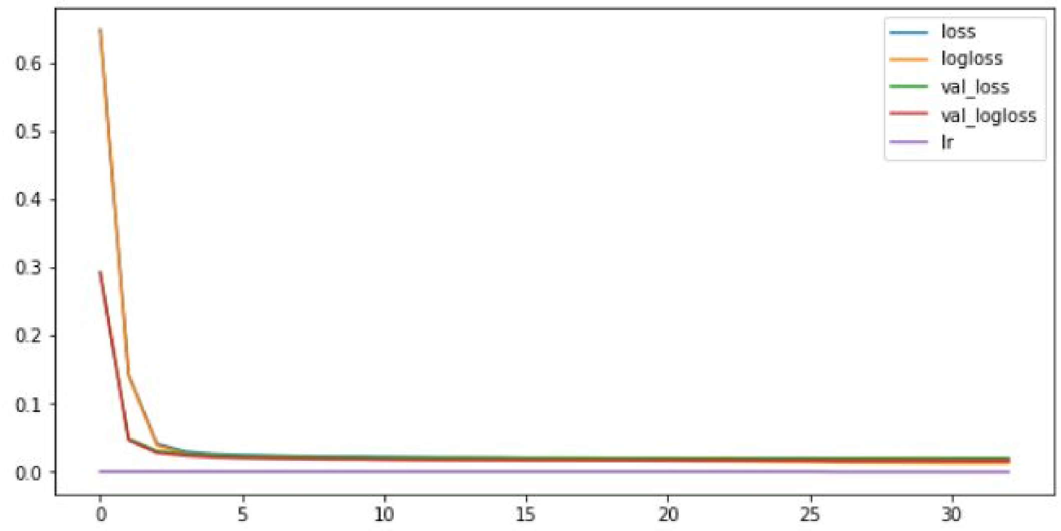


Figure 9: learning curves for model training

To handle the data imbalance, we use multilabel SMOTE (Synthetic Minority Oversampling Technique) to upsample the minority classes of the targets. The results turn out to be worse and we thus choose not to go with that approach.

It is also important to note that this modeling problem is not fully explored yet due to time constraints but will continue to be updated (in the Github repository) and improved upon post-submission of this report. Other deep learning methods left to be explored and possibly used in an ensemble include Residual Networks (ResNets), Inception Networks, MobileNets, etc. Ensemble methods like stacking and blending will also be explored.

Conclusion

We presented in the study, one of the many applications of deep neural networks to problems in various domains; in this case, Pharmacology. Neural networks are powerful statistical models capable of modeling complex non-linear functions that more traditional models may not be able to. We review an exploratory analysis of the data and segue into the modeling process used. We also highlight selection of model hyperparameters that result in good evaluation metrics and justify why various techniques are used in the modeling stage. While there’s more to be done, the best logarithmic loss value so far of 0.0166 is rather good given it is a very sparse multilabel problem with many target classes.

GitHub Repository

If you’re interested in following the progress of this study, please visit the repository here: https://github.com/Hackman-git/Mechanisms_of_action

References

[1] https://www.kaggle.com/c/lish-moa/data

[2] https://web.stanford.edu/~hastie/Papers/ESLII.pdf

[3] https://www.kaggle.com/c/lish-moa/overview/evaluation