Predicting Mechanism of Action Using the Novel Mosaic Method

Skyler Granatir, Gurpreet Singh

I. Problem Statement

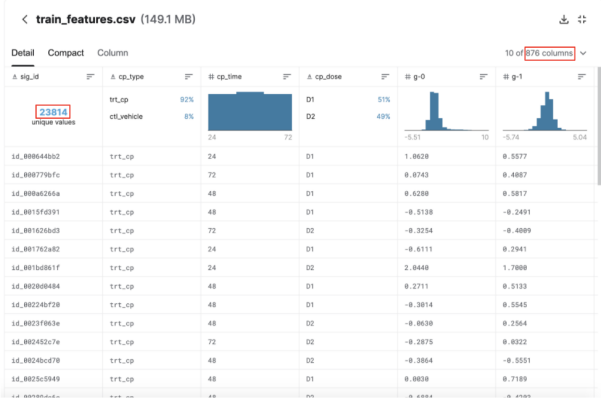
Years ago, therapeutic drugs were derived from natural remedies in which the biological mechanisms were not truly understood. Even more recently, popular drugs such as paracetamol, or commonly known as Acetaminophen, have been clinically established decades before their mechanisms were understood. With the emergence of powerful technologies, we are able to harness gene expression and cell viability data. This data can plausibly be used to measure patients cellular and genomic responses, giving us an indication of possible Mechanisms of Action (MoA) that may explain the underlying process in which the drug works. This targeted method may also provide us knowledge of the pathology of a given disease. With this framework, researchers may identify relevant protein targets associated with a certain disease, and be able to modulate this protein target.

Our data comes from a new technology that simultaneously measures cellular responses from a pool of 100 cell types, distributed as a part of a Kaggle competition [1]. By pooling different cell types, we eliminate the need to identify ex-ante–that is–having to identify which cells are better suited for a given drug. Our training data (see Figure 1a) is a (23814, 876) frame of data, giving 23,814 samples where a dish of cells is treated with a drug. For each of these samples we have gene expression data (g-0.. , g-771) and cell viability data (c-0,c-99) telling us the degradation across the 100 different cell lines. We have approximately 2,700 drugs, each of which with 6 samples that correspond to different dosages and treatment duration times. Thus, with two dosages (D1, D2) and three treatment durations (24,48,72 hrs) we have 6 sample pertubations for each drug.

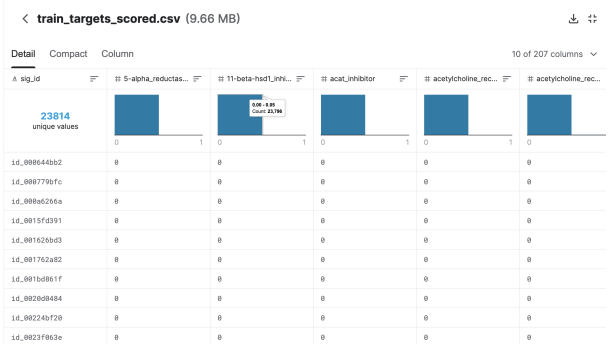
The data offers some problems however, with high variance of many features even within the control group, or being tested by the same drug (see Figure 2a). For example, if we take all the samples associated with a particular drug, we find surprisingly high variance for many genes and cell viability markers, indicating high-noise and randomness due to experimentation.

We attempt to reconcile this by collapsing all samples of a specific drug into a single (1,876) vector. If a drug originally has 6 samples, we convert this drugs data from (6,876) to (1,876). We introduce convolution to collapse this dimension; we learn filters (6x1) for each feature (g-1,g-2... c-99, c-100) in order to more deeply comprehend relationships between samples of the same drug, collapsing the dimension of the 6 dosage and treatment times.

Thus, our project has three aims: i) propose a novel, interpretable model that uses convolution to collapse the six sample pertubations (D1-24, D1-48, D1-72, D2-24, D2-48, D2-72) of a specific drug into a single sample; ii) investigate the efficacy of normalizing the treatment data with control data by subtracting basal drift of control samples; and iii) create a model with sufficient sensitivity, i.e. correctly classify the active MoA when an MoA truly exists. Our framework will be constrained to limited training time and computational resources, with less than 2 hours of training on a cpu.

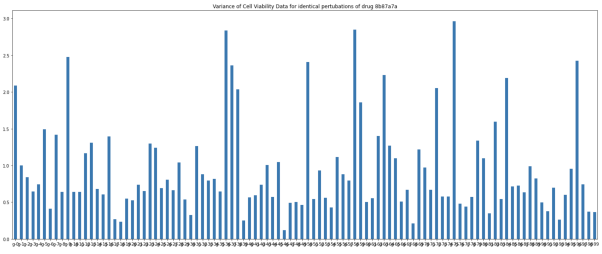


(a) Training samples.

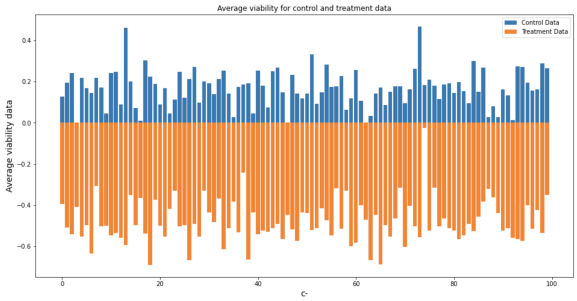


(b) Training labels.

Fig. 1: (a) Training sample data provided by Lab of Innovative Science. Data size of (23814, 876), with 4 categorical columns and 872 numeric columns; (b) Training labels provided by LIS. There are 206 columns associated with different MoA's. Binary entry, where '1' indicates an active MoA.



(a) Variance of cell viability for the different perturbations of Drug 8b87a7a.



(b) Average of each cell viability marker for control group (orange) and treatment group (blue).

Fig. 2: (a) We witness surprisingly large variance of cell viability data for perturbations of the same drug. Some cell types in particular show great variance indicating volatile behavior when treated with the same drug type. We would expect low variance for many cell types across the experiments; (b) We see that control group has a lower average value for the cell viability markers. This makes sense, as without drug treatment, there should be less cell degradation as opposed to the treatment group.

A. 1st stage: No Drug Id

In the first iteration of the competition we only had access to independent rows (with their gene expression and cell viability values) and the corresponding Mechanism of Action. We did not have any more biological information, like what drugs where given or what where the genes. This made the problem a naive multi-label regression/classification problem. We used 2 baseline architectures for multi-label modelling, namely TabNet [2] and a 9 layer neural network. The neural network approach outperformed Tabnet with a factor of 1e-1 on log loss.

B. 2nd Stage: Drug Ids released

Drug ids were released several weeks after the competition began and we reshaped the problem as sequence to sequence with mul-

tilabel prediction. A separate table with drug ids was published and we could now map which drug represented which experiment in the original training data (gene expression & cell viability to Mechanisms of Action).

We could now parse the training data as a sequence to sequence problem, rather than a simple multi-label problem. This is due to the fact that most drugs had 6 rows, with one row for each experiment. Namely $low-dosage$ (D1, D2) and $high-dosage$ (24 hr, 48 hr, 72 hr). Looking at figure [3] we can see cp-dose and cp-time values vary across providing data setup similar to time series analysis.

However the drug ids for test set were not released and our subsequent Mosaic Model could not be tested on the private data. Hence we relied on validation loss for comparison.

sig_id	cp_type	cp_time	cp_dose	g-0	g-1
8bedaf41	trt_cp	24	D1	-0.6920	-1.4790
)c8d6d38	trt_cp	48	D1	-0.7895	0.2293
)9620969	trt_cp	72	D1	-0.9434	0.3857
361a670c	trt_cp	24	D2	0.3067	0.7684
795f4f2fe	trt_cp	48	D2	0.2623	0.0256
)1626bd3	trt_cp	72	D2	-0.3254	-0.4009

Fig. 3: One drug corresponds to 6 experimental conditions

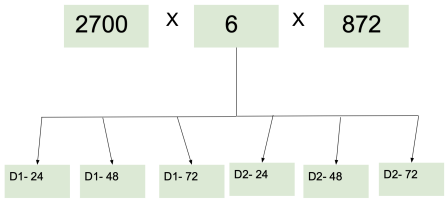


Fig. 4: Converting to sequenced setup across time and dosage

II. Results

Competition scoring was based on log loss of the predicted values based on true target. Target for the model are the activated MoAs. Zero or more MoAs can be activated but generally the target is a sparse matrix with mostly zeros and one MoA activated per row. More specifically, 52% of samples contained exactly one MoA, 8% containing more than one MoA, and 40% contain Zero MoA. At first we started with naive regression models using Scikit learn library. From those we got a log loss of 31.00 with top leaderboard score at the time being 0.016 which is considerably better than ours.

A. TabNet Baseline Results

We introduce a newly curated model, TabNet [2], that was released by Google AI in early 2020. It proposes a method that works on tabular data and offers attentive feature selection to 'attend' to certain data features (g-1,g-2..) at each time step. In a sense, it learns *what to look at* based on *what it has seen*, learning the context when a new data sample is introduced. The results for TabNet were underwhelming, achieving a loss of 0.01933, which is inferior to even our simplistic NN model (to come). It is plausible that TabNet would perform better with longer training time, more sophisticated pre-processing, and different loss functions that ensure stability through training (in the current PyTorch implementation, BCE Loss with Logits is not supported).

B. Fully Connected Baseline Results

We developed another baseline based on the discussion sections on the Kaggle competition [1] and influenced by the approaches that had high score on competition. We developed a 9 layer neural network consisting of 3 fully connected layers, 3 BatchNorm layers and 3 Dropouts with Relu non-linearity in between. We added BatchNorm to normalise the input data since the variance was quite high between rows, as shown previously. It achieves a log loss of 0.00163.

The difference between our model and top winning model on the unseen test set was

cp_type	cp_time	cp_dose	g-0	g-1
trt_cp	24	D1	1.6330	-2.1730
trt_cp	24	D1	0.5223	-1.4420
trt_cp	24	D1	1.1530	-1.2750
trt_cp	24	D1	4.7140	0.1388

(a) Visual variance of g-1, g-2 expressions for exactly similar control dishes.

```
###z-Score normalization of pertubation (glp-glc / var(glc)) (glc is average gene ex for gl in contorl)
###Normalize dependent on the cp_time...
def z_Score_norm(train):
  GENES = [col for col in train.columns if col.startswith('g-')]
  CELLS = [col for col in train.columns if col.startswith('c-')]
  control_data = train[train['cp_type'] == 0]
  control_24 = control_data[control_data['cp_time'] == 0] ##Control data for cp_time = 24
  control_48 = control_data[control_data['cp_time'] == 1]
  control_72 = control_data[control_data['cp_time'] == 2]

  for col in (GENES+CELLS):
    train.loc[train['cp_time'] == 0, col] -= control_24[col].mean()
    train.loc[train['cp_time'] == 0, col] /= control_24[col].var()

    train.loc[train['cp_time'] == 1, col] -= control_48[col].mean()
    train.loc[train['cp_time'] == 1, col] /= control_48[col].var()

    train.loc[train['cp_time'] == 2, col] -= control_72[col].mean()
    train.loc[train['cp_time'] == 2, col] /= control_72[col].var()

  return train
```

(b) Code example for implementing z-Score basal drift subtraction.

Fig. 5

1e-4. Their approach was of stacked Tabnet and SVM. An obvious problem with both such complex approaches is the inability for feature selection or make inference on what the model is focusing on. In this sense, it fails to be *interpretable*.

C. CNN + Fully Connected

Drug ids not released until it was very late in the project timeline, and our previous models treated the rows independently, with no knowledge of related drug experiments. But after the drug id information was released we converted the setup to a sequence problem, with the sequence representing 6 different rows for one drug id. This helps us use techniques like convolution that detects patterns amongst data, where some features might be repeated.

Without the fully connected network, the CNN overfits and the Binary Cross entropy loss explodes. We also tried using other loss functions like MSE Loss (sum of squared error) which did not train well with our model.

III. Pre-Processing and Visualization

As stated before, we have large variance for many cell types when samples are isolated from a particular drug, or even from the control group. That is, when samples from the control group are analyzed, many cell viability markers show volatile, unpredictable behavior. Figure 2a shows the intra-drug variance for each cell type when treated with a particular drug. This can also be seen in a microcosmic example in Figure 5a, where the first five exact samples from control group were looked at for gene expressions g-1, g-2. Perhaps large variance indicates that the cell type is unpredictable, or that the cell type experiences truly different responses under the same drug but different perturbations (D1 vs. D2). Since the latter is plausible, we decide to retain all cell viability markers.

We attempt to explore the baseline effects on cells when they are left in a petri dish for 24, 48, 72 hours. For example, in the control group, we still experience cell death and gene expression changes that are merely due to experimental conditions, and not drug treatment. Thus, we investigate what the basal (natural) drift of cell viability is in the control group, and use this basal drift to subtract and zero-mean our treatment data. Doing so, we expect the remaining viability measurements to be indicative of the cell's response to the drug, rather than portraying basal drift that occurs naturally in the laboratory.

Figure 2b shows the average cell viability (CV) for each cell type in the control and treatment group. We see that CV is notably lower in the control group, which matches our intuition that *some*, but not all cell viability response is due to natural drift, and not strictly drug-induced response. To zero-shift our treatment data, we do z-Score normalization, indicated

in Eq(1) and Figure 5b.

$$g_i^{treatment} = (g_i^{treatment} - g_i^{control})/\sigma(g_i^{control}) \quad (1)$$

We find that when we apply z-Score norm to our various models, we only receive modest improvement (¡5% loss improvement). To make sense of this, we look at some randomly selected cell expression feature distributions in Figure 6a. We see that the distribution actually follows a Gaussian distribution only slightly off from zero-mean.

Retrospectively, we have not altered our distribution by much. We make another attempt at pre-processing normalization by introducing Quantile Normalization, which shrinks each feature distribution to a normal distribution. This is implicit normalization, as we attempting to maintain statistical insight of the distributions while we shrink them. The intuition here is that ML algorithms work with better with normal distributions and smaller numbers (for increased precision). When introducing Quantile Normalization, we again get only slightly better results (6%) improvement for most models).

Thus implicit normalization worked only slightly better than our proposed basal drift method. However, neither of the improvements are on an order of magnitude. It is also worth mentioning that PCA was not of immediate benefit; our data is relatively high rank, so even including 280 principal components results in only 70% of the variance explained. A variance explained vs. components curve is shown in Figure 6b.

IV. Related Work

We reviewed architectures from Kaggle discussion board and several different papers in the field. We did not have information about which genes had their expression value calculated or which cells represented by the viability scores. This led to us treating the problem as multi-label classification.

We then treated the problem as a sequence to sequence problem. There has been some prior work that treats sequence models for clinical data. [3] have used attention framework to model time series data analysis. They encode the input along with its position and this combined structure is parsed into a feed forward (or sometime referred to as fully connected network) but does not employ any recurrence or convolution.

Although it works well on MIMIC - III benchmark dataset made up by de-identified patient records it does not suit our problem statement. (1) All sequences have the same target values. This was because only one MoA is activated per drug id and pertaining to one (6,872) matrix. (2) Gene expression values can have an abrupt shift between dosage or time intervals and, hence one values not lead to another smoothly.

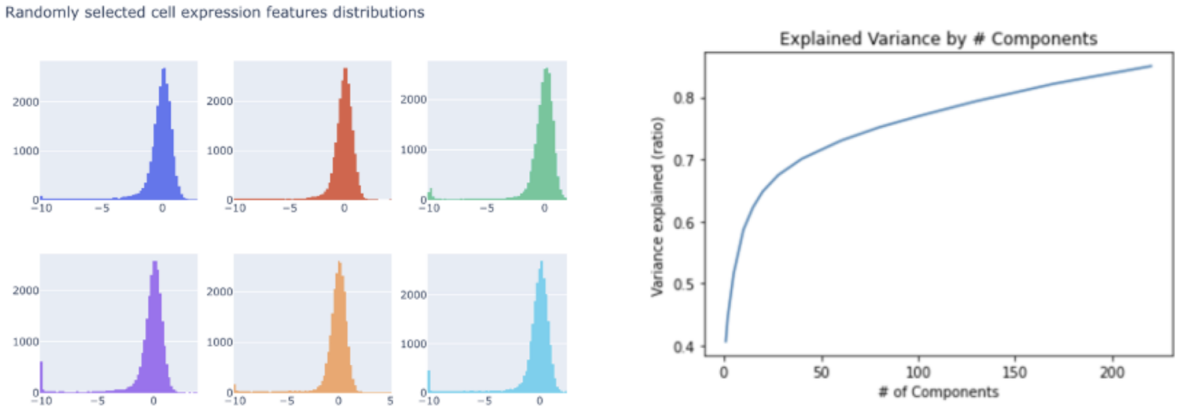
[4] have used auto encoders and support vector machines for drug protein target recognition. The problem with using their approach is that while auto-encoders are really helpful at capturing inherent features of data using encoding / decoding, this usually results in loss of inference. While at the same time, they do not provide performance gains over neural networks.

Looking through review paper on drug discover, [5] we found out that most of the newer models use deep neural network approach. While a more nuanced reinforcement learning could also be used it was out of the scope since public dataset contained de-indentified gene, cells, and drugs.

V. Baseline Method

A. Neural network baseline

Our baseline model was based on top performing neural network models on the competition leaderboard. Please see fig 7 for detailed architecture. It contains 3 batch norm layers, 3 dropout and 3 fully connected layers. We also performed feature selection using Variance



(a) Randomly selected cell expression feature distributions.

(b) Explained variance vs. of principal components considered.

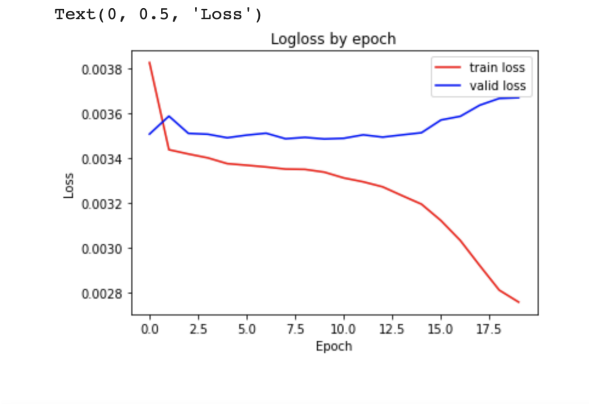
Fig. 6: (a) We see that the original data follows a Gaussian distribution, slightly off zero-mean; (b) PCA graph entails high rank data as hundreds of components must be introduced to explain sufficient variance.

```
Model_NN(
  (cnn): Model_CNN(
    (cnn_1d_1): Conv2d(6, 3, kernel_size=(1, 1), stride=(1, 1))
    (relu): ReLU()
    (dropout): Dropout(p=0.5, inplace=False)
    (cnn_1d_2): Conv2d(3, 1, kernel_size=(1, 1), stride=(1, 1))
    (fc): Linear(in_features=872, out_features=872, bias=True)
  )
  (batch_norm1): BatchNorm1d(872, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dropout1): Dropout(p=0.2, inplace=False)
  (dense1): Linear(in_features=872, out_features=1024, bias=True)
  (batch_norm2): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dropout2): Dropout(p=0.4, inplace=False)
  (dense2): Linear(in_features=1024, out_features=1024, bias=True)
  (batch_norm3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dropout3): Dropout(p=0.5, inplace=False)
  (dense3): Linear(in_features=1024, out_features=206, bias=True)
)
```

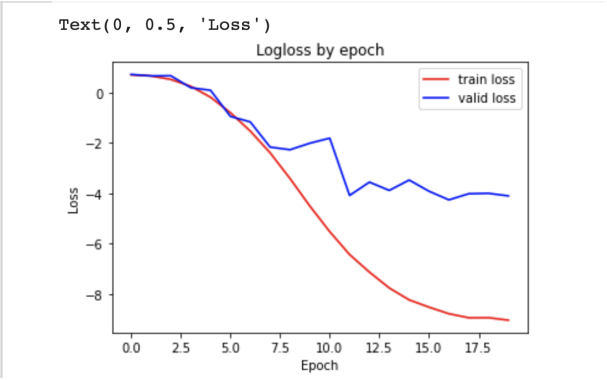
Fig. 7: Neural Network Baseline Architecture

Threshold from scikit learn library. It removes the columns that do not contain minimum threshold variance (0.5) and Adam optimizer.

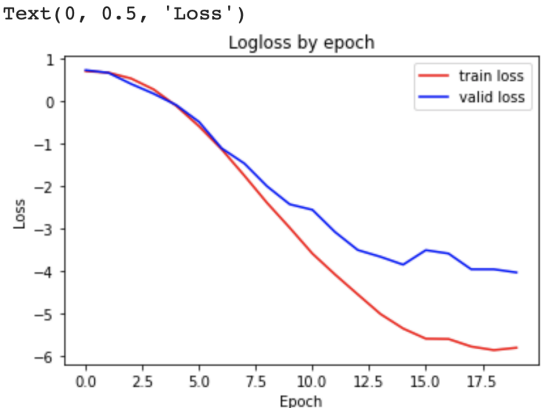
Benefits of Dropout & BatchNorm: With the emergence of Dropout and BatchNorm layers in the current world of Machine Learning, it is important to briefly explain their function. The Dropout layer [6] randomly selects nodes in the adjacent fully connected layer, with probability p, and 'drops' these nodes, temporarily removing it from the network for a single forward and backward pass. Thus the network will update the parameters that remain, and then does this again for new randomly selected nodes, returning the previously dropped nodes back to the network. This prevents certain nodes from carrying too large an amount of responsibility in a layer, and it also encourages nodes with currently small responsibility to take on more responsibility. This also fixes the problem where layers of a network co-adapt and 'make-up' for mistakes made in previously layers, such that complex co-adaptions across layers is reduced. This has been shown to increase robustness and reduce overfitting of a



(a) Training vs Validation Loss [Overfitting]



(b) Training vs Validation Loss [Dropout=0.2]



(c) Training vs Validation Loss [Dropout=0.5]

Fig. 8

network [6].

Overfitting is a problem that describes the situation when a model learns how to strictly represent the training data, however fails to generalize when faced with new data. In a way, overfitting is when the model almost 'memorizes' the training data, but does not truly comprehend the underlying make-up of the classes it is trying to predict, thus an overfitting model fails to have predictive power with a validation set that it has never seen. In Figure 8a, you can see that validation loss is substantially greater than the training loss. However, when introducing Dropout, we see substantial benefits as the validation loss now shares a similar trajectory as training loss.

As mentioned, we also introduce BatchNorm to increase the power of our model and to more effectively train. BatchNorm was originally introduced to reduce the headache of proper initialization. Before sending through a fully-connected layer, BatchNorm forces the data input (before the layer) to follow a unit gaussian distribution. In Deep Neural Nets, without BatchNorm, the later layers may express output activations that are very similar (not much variance), indicating that there is not much learning occurring. By forcing the output activations of each layer to follow a normal distribution, we can more efficiently learn in the *next* layer. Data distribution before and after normalization can be illustrated in the example in figure 9. BatchNorm, in this sense, performs data pre-processing between fully-connected layers.

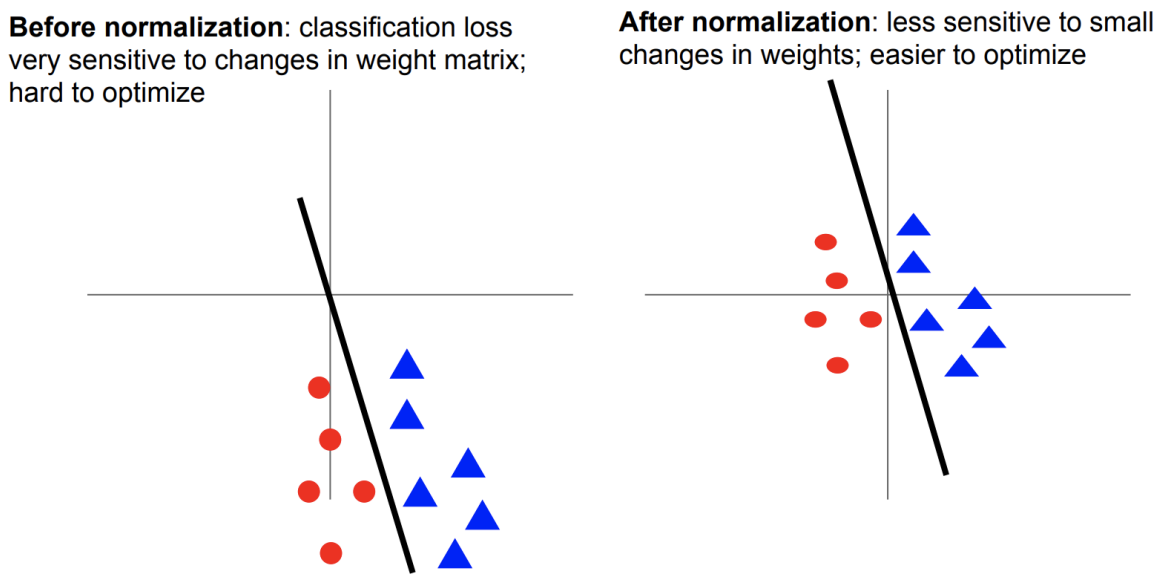


Fig. 9: Data distribution before and after normalization

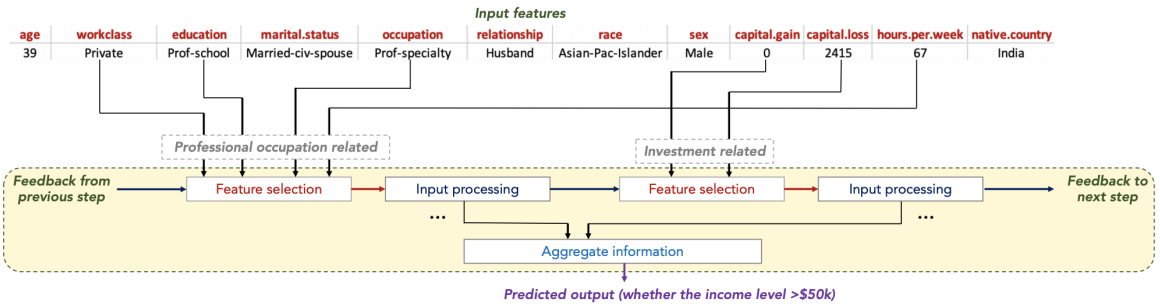


Figure 1: TabNet's sparse feature selection exemplified for Adult Census Income prediction [14]. Sparse feature selection enables interpretability and better learning as the capacity is used for the most salient features. TabNet employs multiple decision blocks that focus on processing a subset of input features for reasoning. Feature selection is based on feedback flowing from the preceding decision step. Two decision blocks shown as examples process features that are related to professional occupation and investments, respectively, in order to predict the income level.

Fig. 10: TabNet's introduction of its feature selection.

B. *TabNet*

We instantiate the model TabNet as a way to introduce attention to our problem. TabNet [2] is an architecture introduced by Google Cloud AI earlier in 2020. Indicated by the name, it works on tabular data and is made to be interpretable and attentive. In the context of this project, TabNet will learn attentive weights between relevant features from the data. Attention is largely used in the field of Natural Language Processing to learn connections between certain words, depending on the context (positional location) of these words. Supposed we are training a machine to interpret the following sentences.

I crossed the river to get to the bank.

OR

I crossed the street to get to the bank.

There are two opposing semantical meanings of 'bank'. In the first example, the machine must attend to the word 'river' to properly understand that the 'bank' refers to a river bank, and

not a monetary bank. Thus, the attention model works here as it learns to induce a weight between 'bank' and 'river', that is, to focus largely on this connection to understand the semantical meaning of the sentence, when the words are positionally nearby.

Consider the potential power in our context of gene expression and cell viability data. A hypothetical: given a high expression of g-1, perhaps we should consider g-47, for this combination of high expression would imply the MoA to be an ACE-inhibitor. With more thorough learning and training time, we would be able to visualize the attentive connections between genes, which may not necessarily improve prediction performance, but can, in itself, unveil important relationships between genes.

We leave this analysis for a future time, and re-focus on our goal at predicting MoA. In TabNet, like most attentive models, we have a decision step where we selectively choose a feature to look at. This process of feature selection is shown more clearly in figure 7. When applying TabNet to our model, we find underwhelming results, resulting in a logloss of 0.01933 on the dataset, which deems inferior to our novel Mosaic Method and many submissions within the Kaggle competition.

Retrospectively, the weak performance of TabNet was found by other data scientists in the community. In fact, the winners of the competition wrote several days ago that TabNet is only beneficial if used after a Stage 1 neural network [7]. Perhaps this be the case, but this also means that the input features to TabNet are processed, indiscernable variations of our original features. It is an important note–under this regime, the input to TabNet *is no longer gene expression or cell viability data*. It is an indecipherable representation of the data created by the Stage 1 NN. This means that attentive weights learned in TabNet are not of any use to us for future research, and are not particularly interpretable. For the reason of losing interpretability, we move on to our Mosaic Architecture for simplicity. It is also worth noting that TabNet has not been thoroughly created in PyTorch, and many errors occur when trying to alter the training framework of the model, such as changing the loss function to BCE Cross Entropy with Logits for stability.

```
def forward(self, x):
    x = self.batch_norm1(x)
    x = self.dropout1(x)
    x = F.relu(self.dense1(x))

    x = self.batch_norm2(x)
    x = self.dropout2(x)
    x = F.relu(self.dense2(x))

    x = self.batch_norm3(x)
    x = self.dropout3(x)
    x = self.dense3(x)

    return x
```

Fig. 11: Architecture for CNN+Fully Connected

VI. Novel Mosaic Architecture

Our novel Mosiac model makes the following improvements over the naive one. First, it splits the data into drug sequenced matrices of size $(6, 872)$ for each drug. We identified 2700 such drugs in the dataset and had to remove others since they did not fit the criterion and also because of ease of processing. We then sorted the drug rows by $dosage$ and $cp - time$ such that low dosage came before high and similarly with time period. This ordering can be seen in Figure 3.

The second improvement over naive baseline was to use convolution as feature extraction on the data, that collapses our (6,876) frames into a single (1,872) vector representation of that drug. The premise here is that all perturbations of the same drug have the same MoA, which we investigated and found to be true. The convolutional filter is important here because we know that value of gene expression at time $t = 24$ is related to its value at $t = 48$ for the same drug. In the same vein, there is

some relation between values gene signatures and cell viability for same drug, but different dosage.

Mosaic model consisted of neural network defined earlier but on top of it we added 2 layers of 1D CNN. The 1D CNN preserves the height and width of the window but continously downsamples the number of channels. The input had size $batchsize * 6 * 872$ with input 6 channels, corresponding to 6 different experiments per drug. The output of CNN layers is $batchsize * 872$ and we assume they contain the sequential features within them. We also tried a version with just CNN layers but it always overfit the data.

The features learned from CNN are then passed to the original neural network that passes them through its 9 layers. It converts them to target of shape (1,206) representing MoAs labels. We observed that it attains a slightly better performance over simple neural network by decreasing log loss by $1e-4$.

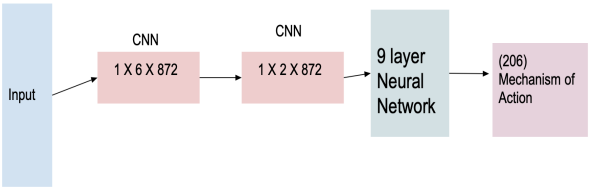


Fig. 12: Architecture for Mosiac Model

We also made a change to the training loss we used with Mosiac Method. Earlier loss function used in training was Mean Squared Loss (MSE) but it proved to be unstable, that is, loss values always blew up when training for significantly longer epochs. We changed it to Binary Cross Entropy Loss (BCE Loss) to stabilize the model fitting. Please note that training loss is different to log loss used by competition to evaluate results. While an easy version of log loss was available for training Pytorch models, we attempted to use negative loss for training but it was unsuccessful since it was also unstable. Nonetheless the Mosaic Method results in a 20% reduction in logloss, and makes it more competitive than our previously well-performing, simplistic model.

```
Model_CNN(
  (cnn_1d_1): Conv2d(6, 3, kernel_size=(1, 1), stride=(1, 1))
  (relu): ReLU()
  (dropout): Dropout(p=0.5, inplace=False)
  (cnn_1d_2): Conv2d(3, 1, kernel_size=(1, 1), stride=(1, 1))
  (fc): Linear(in_features=872, out_features=872, bias=True)
)
```

Fig. 13: Architecture for CNN

VII. Discussion/Improvements

We started with TabNet but it needed careful pre-processing, a more stable loss function, and other models stacked alongside to work. As said before however, this would take away from the interpretability as attentive weights would no longer correspond to the original data features. TabNet by itself did not give useful results. We then saw that neural network approach easily outdid TabNet and naive regression models but they too did not account for sequential nature of the data nor could we easily infer from them. Our novel Mosaic models helps alleviate the latter. While Mosaic is also hard to infer but it makes an effort to extract features from drug sequential dataset under the assumption that gene expression values and cell viabilities are related under dosage levels and measure / Ctrl time.

Our model could be improved with attention framework that tried to place weights on sequenced memory and we feel it could be worth trying out. To gain grounds on inference simpler yet effective modelling techniques like SVM and XGBoost could be tried.

References

[1] "Mechanism of action – kaggle competition." [Online]. Available: https://www.kaggle.com/c/lish-moa/overview

[2] S. O. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," 2020.

[3] H. Song, D. Rajan, J. J. Thiagarajan, and A. Spanias, “Attend and diagnose: Clinical time series analysis using attention models,” 2017.

[4] Q. W. et al., “A novel framework for the identification of drug target proteins: Combining stacked auto-encoders with a biased support vector machine,” 2017. [Online]. Available: doi:10.1371/journal.pone.0176486

[5] J. V. et al, “Applications of machine learning in drug discovery and development.”

[6] A. K. I. S. Nitish Srivastava, Geoffrey Hinton, “Dropout: A simple way to prevent neural networks from overfitting.” 2014.

[7] M. P. et al., “Winning solution – kaggle.” [Online]. Available: https://www.kaggle.com/c/lish-moa/discussion/201510