Mechanism of Action (MoA) Prediction - Kaggle Competition

Lombardi Alessandro, Polvani Niccolo', Zacchei Filippo
Department of Mathematics, EPFL, Lausanne, CH

Supervisors: Krapp Lucien Fabrice, Dal Peraro Matteo
Department of Life Sciences, EPFL, Lausanne, CH

***Abstract*—Drugs research is rapidly evolving and latest progress is partially thanks to the introduction of Machine Learning tools. A variety of machine learning methods such as naive Bayesian, support vector machine and, more recently, deep neural networks are demonstrating their usefulness in drug discovery and development. The aim of this paper is to propose various Machine Learning algorithms in order to predict the biological activity of a molecule, which is referred to as mechanism-of-action, or MoA for short. The model selection has been carried out through a online public competition that elected the best performing models for the given dataset. We describe here the models, based on the Neural Network framework, that we utilized and implemented during the challenge and that made us achieve the top 2% of the final leaderboard, with a position of 72 over 4373 teams.**

I. Introduction

Only a century ago, many drugs have been widely used decades before the biological mechanisms driving their pharmacological activities were understood. Today drug research has moved from the fortuitous approaches of the past to a more targeted model based on an understanding of the underlying biological activity of a particular disease.

Over the past decade, progresses in information technology and growing automation have led to the production of huge and hard-to-handle compound activity and biomedical data [1], [2]. The idea of this project is to perform supervised training on a classifier and then predict the labels of unknown compounds, as done in [3], [4]. The dataset we were provided is based on a new technology that measures simultaneously (within the same samples) human cells' responses to drugs [5],[6], in a pool of 100 different cell types (thus solving the problem of identifying ex-ante, which cell types are better suited for a given drug). This is an increase in the complexity level of this kind of tasks: the majority of examples of compound MoA prediction are restricted to a single cell type, often selected because of its suitability for simple image analysis. The goal is to predict 206 independent possible MoAs given the cellular signature data. Due to the high dimension of the input features and output labels, the best suited algorithms we have found were based on a deep learning structure. The one that we have chosen for the final submissions are a Deep Neural Network (NN), a Tabular NN (TabNet) and a Residual NN (ResNet).
Here is a brief description of the structure of this paper, where we summarize the process that led us to the choice of these models:

Section II: **Exploratory Data Analysis**
  Description of the dataset features and initial insights
Section III: **Feature Engineering**
  The data transformations we have introduced to improve model performances
Section IV: **Evaluation protocol**
  Cross validation technique and evaluation metric
Section V: **Models**
  Description of the implemented models, and the strategies used to optimize them
Section VI: **Results**
  Blending strategies and final model performances evaluation

II. Exploratory Data Analysis

The data we are provided has been collected through a cell-based assay designed to capture gene expression and cell viability levels. The main features of the dataset are *Gene Expressions* and *Cell Viability*; the specific cell types of the columns were not provided. Both their values are numeric and they both have been clipped between -10 and 10 beforehand.

The measurement of the gene expression is based on the L1000 assay[6]. Each gene feature represents the expression (presence/absence) of one particular gene, therefore there are 772 individual genes being monitored in the assay. The values are numeric and the data is clean, centered and is within the same scale, see Fig. 1, where we plot the distribution of the first four columns.

The measurement of the cell viability is based on the PRISM assay [5]: it is a measure of the proportion of live, healthy cells within a population. Cell viability assays are used to measure cell survival following treatment with compounds. Also in this case the distributions show no skewness, but, due to the clipping at -10, a significant negative tail is often present.

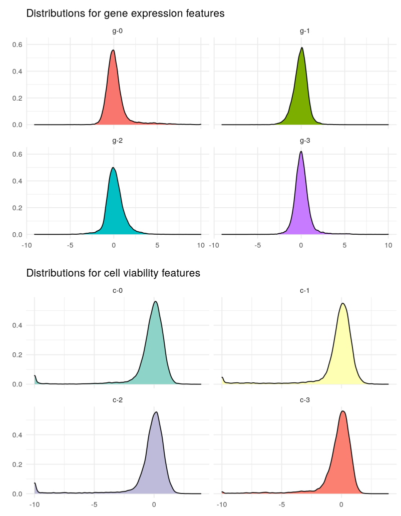

Fig. 1. Gene expressions (g-) and cell viability (c-) distributions

In addition to the data collected from the assay, we are also provided three columns detailing the experiment type, dosing, and measurement time:

- Treatment/Control - The cp_vehicle column indicates whether the experiment is a treatment (contains drug) or control (contains no drug).
- Dosage - The cp_dose column indicates the dose level used in the experiment.
- Timing - The cp_time column indicates the amount of time elapsed between adding the drug and when the measurement was taken.

The main insight that we get from data analysis regards the targets. They consist of 206 possible mechanisms of action, independent from one another. With so many target classes, MoAs are rare; and the targets dataframe is mostly sparse. The percentage of non-zero target class values is 0.343% and each column is very unbalanced towards negative values. Most classes have between 10 and 200 positive MoAs within the 24k rows of training samples (Fig. 2 left). The class with the most positive entries has around 800 whereas the one with the least has only 1. In addition, 400 ”non scored” targets (other possible MoAs) were provided, and they have been used in the pre-training of our Deep Neural Network.

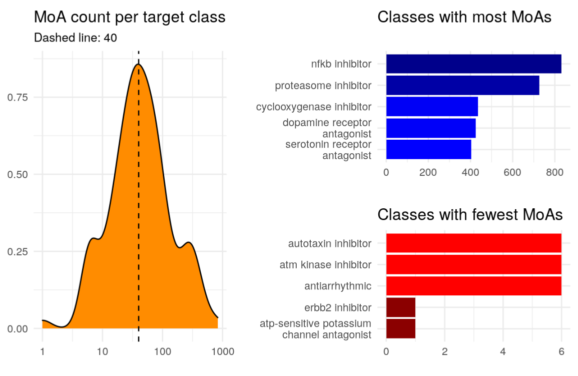

Fig. 2. MoAs positive appearances. The dashed line marks the peak at 40.

III. Feature Engineering

The feature engineering phase consists in four steps. Each of the following takes as input the data returned by the previous step:

I *Control data:* our data contain samples (cp_type=control) that come from controls, were no compounds were used. They are filtered from the training since they do not generate any Mechanism of Action.

II *Scaling:* the features are scaled using a Quantile Tranformer to follow a normal distribution. For a given feature, this transformation tends to spread out the most frequent values. It also reduces the impact of (marginal) outliers.

III *PCA:* since our continuous features were divisible into two categories, Gene Expression and Cell Viability, the PCA analysis is performed separately on these groups. Separating the dataset in two parts leads to lower dimensionality space after PCA is used, and we obtained: 30 components for cell viability features and 350 components for gene expression features which explained 90% of the variance in both cases. However the results obtained using the PCAs variables as training features have been worse than using original data. Instead, concatenating PCAs transformed features to the original dataset led to improved performance.

IV *Feature Augmentation:* In addition to the original features we have introduced the sum, mean, standard deviation, kurtosis and skewness over a row of the gene expression features and cell viability, both looking at them independently and together. In addition the squares of the features have been added.

IV. Evaluation protocol

A. Cross Validation

Because of the sparsity of the targets, a good cross validation technique is necessary in order to perform reliable evaluations. We decided to use a Multilabel Stratification, which divides our dataset into multiple folds by distributing positive targets across all folds as evenly as possible. This choice allowed us to have at our disposal a more reliable scoring method for our models, granting us better prediction score in the unknown complete test set which was used to evaluate the final leaderboard. We implemented stratified splitting also to make predictions: we split the dataset in n folds and train a model on each one; the final prediction is the average across all n models. Each out of fold not used for the training is used for evaluating cross validation score.

B. Evaluation Metric

Evaluation metric for the leaderboard is log loss (aka cross entropy), that is the same loss used to to train the model. The formula is:

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

where M is the number of MoAs, N is the number of samples, $y_{i,m}$ is the true target for m-th MoA, while $\hat{y}_{i,m}$ is the predicted probability.

V. Models

A. Neural Network

Baseline Model: Our baseline model consist of a NN with the following architecture: an input layer, only one hidden layer and a final output layer, each composed by 1000 neurons and inserting a dropout after each layer. The activation function that performed best is the leaky rectified linear unit (LeakyReLu) function, while the optimization phase has be done using Adam[9] optimizer and mini-batch descent, taking as input the whole training set and performing a simple train-test split for cross-validation. The loss to be minimized is the log-loss, the same used for leaderboard evaluation. The choice of the NN architecture was inspired by some public notebooks which were performing well in the competition.
In order to improve the performances of the model we implemented the following techniques:

1) *BatchNorm*: The outputs of each layer have been normalized using a batch normalization, i.e. performing the normalization for each training mini-batch. This accelerates the learning phase and acts as a regularizer [7].
2) *Architecture Search*: due to the long training time we performed a manual architecture search converging to a neural network with 5 layers, see Fig. 3.
3) *L2 regularization*: the log-loss has been regularized adding the L2 norm of the weights, using a weight decay of 1e-5.
4) *New cross validation and training approach*: a huge problem that arised in the training phase was a consistent overfitting. Regularization has made a big improvement, but did not completely solved the problem. In addition cross validation was not reflecting leaderboard results. This issue has been solved introducing the stratification technique described in subsection (II.A), however overfitting was still present. To contrast it, we have changed the learning phase of the model. Instead of taking as input the whole training set, we first split the set in 7 different folds using Stratified Splitting. Then we train a model for each fold and average the predictions of the models for unseen data as described in the same paragraph.
5) *Pretraining*: Slightly better results have been obtained pretraining our NNs looking at both scored and non-scored targets. Then the last layer of the NN is substituted with a new one that is

trained looking only at the scored targets, while the weights of the other layers are initially frozen. After one portion of epochs (one portion = total number of epochs / total number of layers) also the weights of previous layer become trainable, starting from the second last and unfreezing another layer after the same amount of epochs.

We tried some techniques which did not lead to better results:

- *Balancing*: we introduced a weighted random sampler for the mini-batch composition, giving higher probability to be extracted for samples that presented positive values among the targets. However no choice of the weights have managed to improve the final score.
- *Data Augmentation*: we have tried to implement a GAN to generate new data, but the new ones did not give any information to the model, having the same results as using original data.

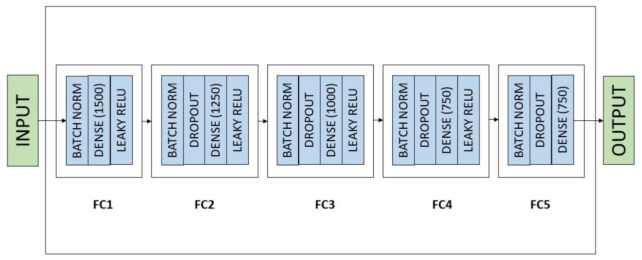

Fig. 3. Schematic of the NN final architecture, FC = fully connect layer

The improvements for each addition to the model and the feature engineering process in validation scores are shown in the table in Fig. 4. The final NN model used in our submission includes all the feature engineering steps and the architecture of Fig. 3, batch normalization, regularization and pretraining, giving the highlighted score in Fig. 4.

Models	LogLoss CV	Public LB	Private LB	Accuracy
Baseline	0.01709	0.02022	0.01988	0.99722
+ Feature Engineering II	0.01697	0.01918	0.01698	0.99722
+ Feature Engineering III	0.01682	0.01923	0.01694	0.99728
+ BatchNorm	0.01662	0.01923	0.01694	0.99723
+ Final NN architecture	0.01641	0.01906	0.01684	0.99722
+ Regularization	0.01639	0.01861	0.01655	0.99725
+ Feature Engineering IV	0.01630	0.01857	0.01647	0.99727
+ Pretraining	**0.01591**	**0.01835**	**0.01626**	**0.99730**
+ Balancing	0.01595	0.01838	0.01627	0.99725

Fig. 4. Table of the improved scores obtained by progressively introducing the described techniques to the model. See section III (feature engineering) and V.A (NN) for references. Cross validation log-loss and accuracy are evaluated as described in section IV. Public LB and Private LB are the scores on public test set and private test set (given after competition deadline).

B. TabNet

The second model we used in this competition is a TabNet. Proposed in [8], TabNet is a deep tabular data learning architecture. It uses sequential attention to choose which features to focus on at each decision step, enabling interpretability and more efficient learning as the learning capacity is used for the most salient features. In Fig. 5 you can see an example of a TabNet architecture, composed of a feature transformer and an attentive transformer at each decision step:

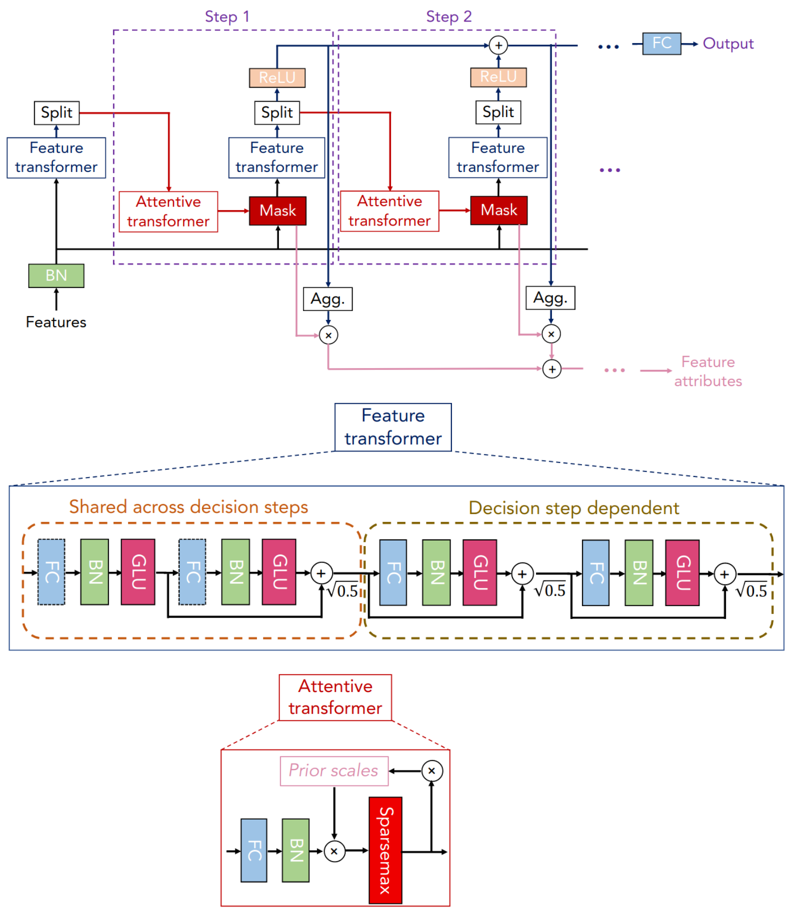

Fig. 5. Tabnet architecture (top), feature transformer (middle), attentive transformer (bottom); source: [8]

- A feature transformer consists in a sequence of layers, where each one is composed of a fully-connected layer, batch normalization and gated linear unit ($GLU(a) = a \otimes \sigma(a)$), eventually connected to a normalized residual connection with normalization. The output is split into two groups. One group is aggregated to the overall decision, while the second one is used as input for the attention phase to gain additional information.
- The Attention Mechanism is instead used to perform feature selection. In the illustrated example, a fully connected layer with batch norm output, modulated with prior scale information, is taken as input by a Sparsemax (or Entmax) activation function. It is used to predict a mask to select only a subset of features for use in subsequent layers.

TabNet model is easily usable as it is included in Pytorch as TabNet.regressor. Our TabNet is pretty shallow, the setup is the following: a width of 32 for the decision prediction layer, and a width of 32 for the attention embedding for each mask, 1 step in the architecture, a gamma value of 1.3, Adam optimizer with a learning rate of 2e-2 and a weight decay of 1e-5, a sparsity loss coefficient of 0, and entmax as the masking function. It was trained with a batch size of 1024 and a virtual batch size of 128 for 200 epochs before early-stopped by a patience parameter of 50 epochs.

C. ResNet

The final model we implemented is a ResNet. A residual neural network (ResNet) is an artificial neural network (ANN) that utilizes

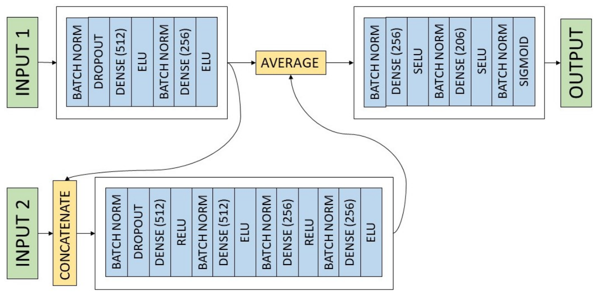

Fig. 6. Schematic of the ResNet architecture

skip connections, or shortcuts to jump over some layers. You can see the architecture of this model, for which we took inspiration from [10], in Figure 6. The feature engineering for this model introduced one additional step since the ResNet, as you can see from the image, has two heads. The architecture is indeed based on ResNet-like Residual NN with two inputs, one is from previous described features, as for the other models, and the other one includes the 447 important features selected by a Student's t-test based on the 25% quantile of the p-values for the test of sample means between training rows, with and without MoA. All features are normalized by Standard Scaler. Using a ResNet-like topology brings the benefit of skip connections in the fully connected (FC) layers, which allows the flow of information to be passed from one layer to next layers. In this way, the model gains better capability of learning with more hidden neurons and layers without the early degradation problem. The training process can be divided into two phases:

- First, the model is trained to predict non-scored targets with Adam optimizer for 50 epochs minimizing the binary cross entropy loss, with a learning rate of 10^{-3}, a batch size of 128. ReduceLROnPlateau learning rate scheduler is used with a factor of 0.5 and a patience of 4 epochs.
- In the second phase, the obtained model weights from non-scored targets are being reused, via transfer learning, to predict scored targets using loops of freezing and unfreezing process as described previously. The rest of the training process is analogous to the one used for the Neural Network.

VI. Results and Final Blending

The results we have obtained so far for the NN, Tabnet and ResNet are 0.0159, 0.0150 and 0.0147 respectively (log-losses computed on the validation set). They show a big decrease with respect to the log-loss of 0.13 for a dummy predictor, that predicts all 0 probabilities for each input. Since the performances are very similar, we ask ourselves if their predictions are similar too. The correlation heatmap in Fig. 7 gives a visual representation of the diversity in our best models. We took the mean of target-wise correlation coefficients as the final value of each model. The correlation between the predictions of our models on the validation set is impressively low, even though they all scored very well. The idea is then to blend the three models in one ensemble, that produces the weighted average of the outputs of the models given the same input. The weights are obtained minimizing the cross validation loss of the ensemble. In order to achieve it, we load the "out of the folds" predictions of our models, and we optimise the following loss using gradient descent w.r.t. the weights (w_k) of the average between the output ($\hat{y}_k$) of the $K = 3$ models:

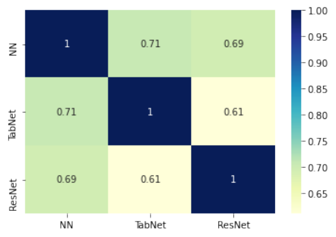

Fig. 7. Correlation Matrix of the outputs of the models

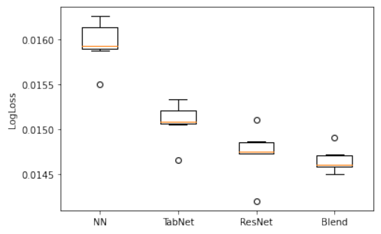

Fig. 8. Comparison between the log-losses of our models (**dummy predictor score: 0.13**), splitting the validation dataset in 7 parts;

$$F = -\frac{1}{NM}\sum_{m=1}^{M}\sum_{i=1}^{N}[y_{i,m}\log\left(\sum_{k=1}^{K} w_k\hat{y}_{i,m,k}\right) + \\ + (1-y_{i,m})\log\left(1-\sum_{k=1}^{K} w_k\hat{y}_{i,m,k}\right)]$$

In Figure 8 you can see the boxplot representing the cross validation scores of the models, compared with the ensemble score, splitting the validation set in 7 parts. We can clearly see that the ensemble model is more robust than the others showing less variance in prediction scores, and has a lower mean loss. The scores on final leaderboard are: 0.01626 (NN), 0.01633 (TabNet), 0.01644 (ResNet) and 0.1610 (Ensemble), showing the better performance of the blended models again.

We concluded the challenge in position 72 among 4373 competing teams (top 2%), earning a Kaggle Silver Medal, and we are very satisfied with our result.

VII. Conclusion

Without any prior knowledge about MoA or the medical domain, we have done our best to apply the ML/DL techniques that we have learned to this difficult dataset and we have shown the results of three different deep learning models, and a possible ensemble in order to achieve better performances on MoAs prediction.

This challenge pushed us to look for models that we did not know before and allowed us to better understand how to define the architecture of different neural network in Pytorch and tensorflow, optimizing the hyperparameters, doing pre-training and fine-tuning a model. A limiting factor for us was that we joined the challenge just 3 weeks before its conclusion. Had we had more time, we would have tried adding to our blend another model: the EfficientNet, which requires data to be converted into images using t-SNE algorithm. Another one would have been using the control observation, which we discarded from our dataset, to perform data augmentation.

VIII. Acknowledgments

A special thanks goes to our mentor Lucien Krapp that helped us with his ML knowledge throughout the competition, and to Kaggle community, that was really helpful for sharing powerful insights and answering to personal doubts. Hoping for gold next time!

References

[1] G. Papadatos, et al. *Activity, assay and target data curation and quality in the ChEMBL database*, J. Comput. Aided Mol. Des., 29 (9) (2015), pp. 885-896

[2] S. Kim, et al. *PubChem substance and compound databases Nucleic Acids Res.*, 44 (D1) (2016), pp. D1202-1213

[3] Godinez, W. J., Hossain, I., Lazic, S. E., et al. *A Multi-Scale Convolutional Neural Network for Phenotyping High-Content Cellular Images*, Bioinformatics 2017, 33, 2010–2019.

[4] Kraus, O. Z., Grys, B. T., Ba, J., et al. *Automated Analysis of High-Content Microscopy Data with Deep Learning*, Mol. Syst. Biol. 2017, 13, 924.

[5] Corsello et al. *Discovering the anticancer potential of non-oncology drugs by systematic viability profiling*, Nature Cancer, 2020, https://doi.org/10.1038/s43018-019-0018-6

[6] Subramanian et al. *A Next Generation Connectivity Map: L1000 Platform and the First 1,000,000 Profiles*, Cell, 2017, https://doi.org/10.1016/j.cell.2017.10.049

[7] Sergey Ioffe, Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training byReducing Internal Covariate Shift*, arXiv:1502.03167

[8] Sercan O. Arık, Tomas Pfister. *TabNet: Attentive Interpretable Tabular Learning*, arXiv:1908.07442v5, 9 Dec 2020

[9] Diederik P. Kingma, Jimmy Ba *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980v9, 30 Jan 2017

[10] Resnet public notebook: https://www.kaggle.com/demetrypascal/2heads-deep-resnets-pipeline-smoothing-transfer