Applications of Machine Learning for the prediction of drug target interactions in the kinome

Georgios Kalantzis

Systems Approaches to Biomedical Science Centre for Doctoral Training
University of Oxford
`georgios.kalantzis@seh.ox.ac.uk`

Supervised by Prof. Garrett Morris, Dept. of Statistics, University of Oxford
In collaboration with Dr. Thierry Hanser, Lhasa Ltd, Leeds

October 17, 2019

“All generalizations are dangerous. Even this one.”
Alexandre Dumas

ABSTRACT

Random forests constitute a widely used prediction technique in machine learning and protein-ligand binding affinity in particular. However, when it comes to multiple targets and thousands of compounds, training and prediction can be rather slow. Multi-task neural networks, on the contrary, have been proved to offer significant improvements in classification problems, both in terms of accuracy and efficiency. This study explores a variety of computational methods for predicting binding affinities between thousands of compounds and hundreds of proteins, while focusing on the benefits of multi-task learning for regression problems. Targets are selected from kinases, an attractive for drug discovery protein family because of their involvement in numerous diseases and types of cancers. By using cross-validation for parameter selection and model evaluation, we reproduce what is known already for the efficacy of random forests, but also show that multi-task neural networks can perform equally well and by two orders of magnitude faster. Our results indicate a great promise for multi-task learning in the advancement of computer-aided drug discovery.

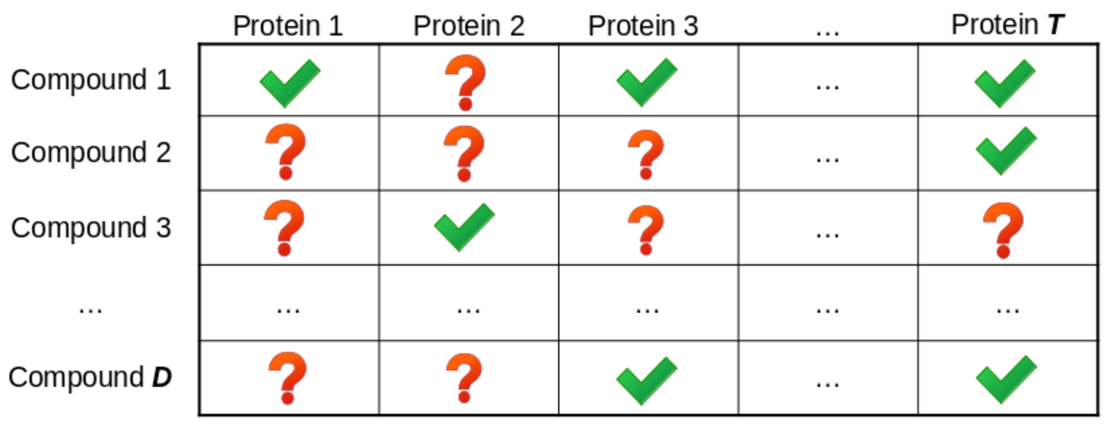
***K*eywords** protein-ligand binding · bioactivity prediction · regression · ChEMBL · IC_{50} · machine learning · neural networks · multi-task Learning · single-task learning · matrix completion

1 Introduction and Background

Drug discovery is a long, challenging and expensive process that incorporates specialised knowledge from medicine, biochemistry and pharmacology. Despite the advances in each of these fields, releasing a new drug to the market can cost up to 1.8 billion US dollars and require 13 years on average [1]. To reduce this complexity, both in terms of effort and money, many computational methods have been developed during recent decades for virtual screening and computer-aided drug-design. These methods are becoming increasingly effective due to the advent of large databases of experimental data and revolutionary applications of machine learning in every scientific domain.

More than 20 thousand proteins exist in the human proteome while every pharmaceutical company retains libraries with millions of compounds, some of which could act as drugs dealing with a specific pathogenicity. This creates a space with billions of potential interactions between targets and compounds which the aforementioned databases aim to describe [1, 2, 3]. Unfortunately, only a tiny fraction of all pairs of proteins and small molecules have been assayed, partially because of cost. Although the vast majority would not offer anything interesting, among all these interactions some are effective, meaning that a compound can bind to a protein and activate the biological pathway for a desired effect. It is hypothesized that new drugs are still hidden somewhere in that space waiting to be discovered.

This project aims to explore the use of various computational methods, including but not restricted to deep neural networks, for predicting the binding affinity of protein-ligand interactions. I will use data from ChEMBL, a publicly available database with protein-ligand binding information, and deploy common techniques for regression. I will investigate the effectiveness of methods that are known to work accurately, like random forests, and implement specialised techniques, like multi-task learning, to bypass specific bottlenecks. Assuming that all the available data are placed in a drugs$\times$targets table, I aim at imputing new values in the initially sparse matrix, a problem that can be described as *matrix completion* and visualised in Fig. 1. One way to tackle this is using non-negative matrix factorization and convex optimisation, techniques that do not require any features. In our case, however, interactions are accompanied by rich data about targets and compounds that can be used for increased accuracy. I will follow a ligand-based approach and use chemical fingerprints as features for my models.



	Protein 1	Protein 2	Protein 3	...	Protein ***T***
Compound 1	✓	?	✓	...	✓
Compound 2	?	?	?	...	✓
Compound 3	?	✓	?	...	?
...	...	...	...	...	...
Compound ***D***	?	?	✓	...	✓

Figure 1: A visual example of matrix-completion problems. Initially, only a subset of all the available interactions are known (*green ticks*) and, based on those, we aim to estimate the missing values (*red question marks*).

My framework will be designed in a way to deal with hundreds of kinase proteins. This protein family has recently become an attractive target for drug discovery efforts because kinases are involved in a wide variety of human cancers as well as other diseases like inflammation and

Alzheimer's disease [4, 5]. When a ligand binds to a protein, the chemical conformation of the protein changes, allowing it to interact with other biomolecules and activate specific biological pathways. An association between a protein (P) and a ligand (L) could lead to the formation of a protein-ligand complex (PL) which can be formalised as $P + L \rightleftharpoons PL$. Thus, the association constant is defined as:

$$K_a = \frac{[PL]}{[P][L]}$$

and the dissociation constant K_d is the inverse of that. K_a measures the propensity of P and L to bind and form a complex. Another important measure of potency is the half maximal inhibitory concentration, IC_{50}, usually measured in nanomolar units ($1nM = 10^{-9}$ mol/L). IC_{50} depends on the conditions of the assay but in general, when the affinity is high, the concentration of the ligand does not need to be high to achieve its maximum potential.

Here, I use the terms *drugs*, *ligands* and *compounds* interchangeably, as well as *proteins* and *targets*. The rest of this report is structured as follows: Section 2 gives a short review of the literature, followed by an extensive description of my approach in Section 3. Then, Section 4 presents the design and results of the computational experiments I conducted. My study is finalised with conclusions in Section 5, pieces of code and additional results in the Appendix.

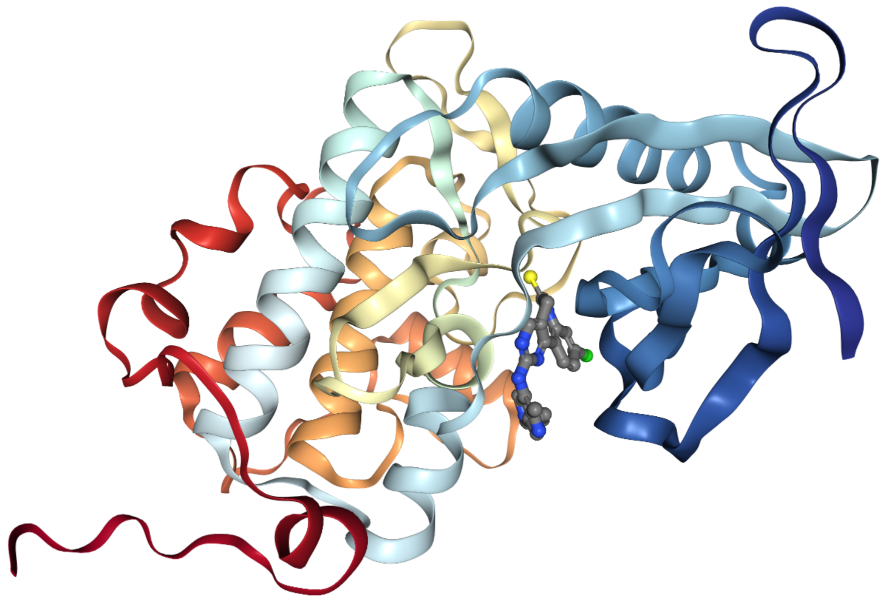


Figure 2: An example of kinase inhibition. A benzolactam-derived inhibitor binds to the pocket of polo-like kinase 1 (PLK1, PDB:3THB), a protein that has been reported to be over-expressed in numerous cancers, with a corresponding $\text{IC}_{50} = 2 \pm 0.8$.

2 Related Work

Machine learning methods for predicting protein-ligand interactions can be divided in two broad categories, classification and regression. For the first family of approaches, a compound is classified as active or inactive according to whether it binds to the target or not. It is important to use a threshold, that is appropriate for a particular target or type of assay, to transform a measurement to 1 or 0, indicating an active compound or inactive respectively. For the second family of methods, a regression algorithm is utilised for predicting the binding affinity between a target and a compound. In both cases, a variety of methods have been developed, ranging from partial least squares to support vector machines and deep neural networks.

The biggest advantage of viewing the problem as a classification task is the ability to model known interactions in many ways, one of which being the probabilistic matrix factorisation (PMF) [6]. This approach is purely quantitative as the only input is the binary matrix, $\mathbf{R}_{N\times M}$, given N compounds and M targets, where 1 indicates a binder and 0 a non-binder. According to PMF, $\mathbf{R}$ can be factorized as $\mathbf{U}^{\top}_{N\times D} \cdot \mathbf{V}_{D\times M}$ with matrices that express drug-drug and target-target correlations in terms of D latent variables. A probabilistic linear model with Gaussian noise is used to model interactions, and matrices $\mathbf{U}$ and $\mathbf{V}$ are such that a log-likelihood function is maximized, under constraints for failing to capture known interactions. This technique inherits the advantages of Collaborative Filtering [7], since every compound is represented by the targets with which it interacts and compounds with similar representations ought to have similar chemical properties.

To validate the intuition behind the efficacy of this model, the authors of [6] ran two sets of experiments. Firstly, they clustered the drugs within the latent space and compared the corresponding partition with that obtained by checking chemical or structural similarities. In brief, drugs that share common activities (i.e. lie close to each other in the latent space) shared common chemical characteristics for the majority of the cases. The next experiment concerned the ability of the model to predict new interactions. By adopting a 5-fold cross-validation (and repeating 100 times) they produced predictions, independently for four target types, and compared against two other published methods. They also compared a standard with an active learning version of the model by hiding 70% of the dataset and incrementing it iteratively. In brief, the active learner was 1.44 times more accurate than the passive learner and also better than other methods for the majority of the targets; performance, measured by the area under the receiver operating curve (AUC), varied from 0.650 to 0.904.

Neural networks constitute an increasingly used technique in cheminformatics [8, 9, 10, 1] as publicly available data sets grow in size. Normally one would need to train a single model per target, thus the broader a study is with respect to the number of proteins studied, more computational effort will be needed. However, [11] is a promising approach that uses multi-task neural networks to predict novel interactions after training one model on an ensemble of data. The idea of using one "big" model instead of many separate ones, comes from the fact that single-task models use the same type of input, produce the same type of output, thus have to behave similarly. According to the authors, the most accurate approach is a pyramidal network with a wide shared layer to process the input, attached to a number of narrow layers, independently, which are then followed by a *tanh* activation function each.

Another recent application of neural networks is that of Whitehead et al. [12] for regressing drug-target binding affinity. The authors present a network that uses 320 chemical descriptors and sparse bioactivity data to predict novel pIC_{50} values. They use a rather shallow neural network with one hidden layer and an iterative tool to impute missing values and sequentially expand the training data with the most accurate predictions. Accuracy here is measured by the coefficient of determination, R^2. The missing values are initially replaced by the averages of the known values for each assay and then they are computed iteratively as follows:

$$\mathbf{x}^{n+1} = \frac{\mathbf{x}^n + f(\mathbf{x}^n)}{2}.$$

Uncertainty is estimated by using an ensemble of twelve similar networks and hyperparameters for the NN architecture were selected after a random holdout validation. A more detailed review of deep learning techniques for drug discovery can be found in [1].

Aiming for novelty, I will work on regression as the case of classification has been extensively studied. My approach will be similar to that of [5] which, however, relies partially on private data and methods. The main contribution of the current study will be the application of multi-task learning to regression and protein-ligand interaction prediction.

3 Data and Methods

In this section I will present the pipeline I created, in terms of data, features, and algorithms, for evaluating a method that accurately predicts new bioactivities. I will also describe how my models were trained and tested, and which performance criteria I used.

3.1 The dataset

I prepared a curated data set, that is large enough, robust and suitable for training with cross-validation. I worked with the version 25 (the latest currently) of ChEMBL, an online publicly available chemical database of bioactive molecules with drug-like properties [13, 3, 2]. While ChEMBL contains assay data from more than twelve thousand proteins, I focus only on kinases for reasons that were explained previously. In particular, there are almost 800 kinases in the database and fetching all the interactions for them resulted in a set with more than $800,000$ activities; this is another indicator of the importance of this protein family in drug development and, possibly, an indicator of research bias. A summary of the total number of activities per type of measurement is shown in Fig. 3a. Then I selected those bioactivities measured with IC_{50}, the inhibitory concentration at half-maximum, and filtered the data according to the following rules:

1. Select compounds with SMILES representation so that fingerprints can be calculated
2. Remove questionably large values (there were cases with $\text{IC}_{50} \geq 10000nM$)
3. If there are many assays for the same drug-target pair, keep the one with the tightest binding.
4. Select only those targets with at least 100 bioactivity values.
5. Select only those compounds interacting with at least 2 targets.

The last two steps were done iteratively as they depend on each other. At the end of this process, the data set was reduced to 110 targets, $23,361$ compounds and a total of $62,656$ interactions. On average, there are 569.60 interactions per target and 2.68 per compound.

A family-specific threshold for determining whether a compound binds to a kinase is that of $30nM$[1]. Using this value, $21,262$ interactions are active, which is roughly one third of the entire data set; there is clearly a bias towards inactive compounds. The next step of data preparation is the transformation to $-\log_{10}(\cdot)$ values, which is denoted as pIC_{50}. This rescaling is helpful for regression and the corresponding distribution is showed in Fig. 3b.

3.2 Features

After selecting which targets to study and what type of data to analyse, the next thing to answer is what type of features should be used. Again, there are many options available, mainly categorised as ligand-based, target-based, or mixed. I followed a ligand-oriented approach by using chemical fingerprints as the only type of features for regression.

Chemical fingerprints are a way of describing the chemical topology and composition of compounds. A molecule is decomposed into a set of fragments, each centered at a non-hydrogen atom, which are then iteratively extended to neighboring atoms. Each fragment is assigned a unique integer-identifier which is finally hashed into a fixed-length binary vector [11, 5]. Such tools are commonly used in cheminformatics, especially when for measuring similarity between compounds. I worked with

[1]Check more at `https://druggablegenome.net/ProteinFam`

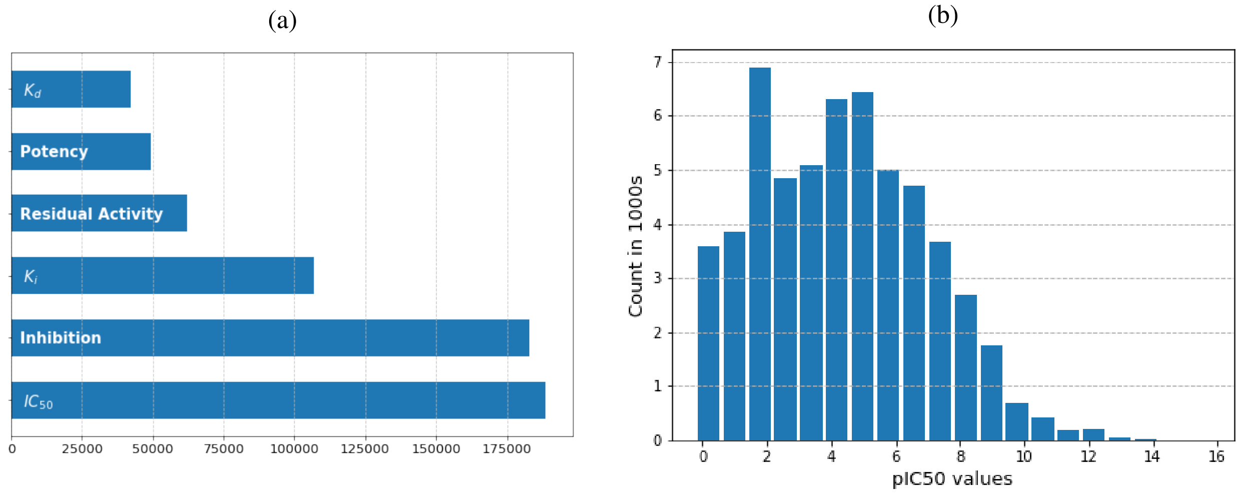


Figure 3: a) Absolute count of occurrences for each type of activity for the 797 kinases in ChEMBL before pre-processing. b) Distribution of the obtained pIC_{50} values, the target-variable for the regression.

extended connectivity fingerprints (ECFP4) with 2048 bits, which were calculated using `RDkit`[2] and the function `GetMorganFingerprintAsBitVect` with radius 2.

3.3 Single Task Learning

I used single-task learning (STL) techniques as a reference for assessing the methods that will be implemented later on. A common method for regression relies on *Random Forests* (RF) [5, 12, 10], which are an ensemble of decision trees with numerous advantages for many applications. One reason, among others, that RF are commonly used by cheminformaticians is the feature selection process which they incorporate and which can be easily used for interpretation.

When regression needs to be performed on high dimensional data, as with fingerprints, *Lasso* regression (LR) [14] is one reasonable tool. LR is a shrinkage method that performs both variable selection and regularisation in order to enhance the prediction accuracy and interpretability of the statistical model it produces. It uses a penalisation scheme with norm-1 which forces the coefficients of the linear model to act as "binary" objects that either have a meaningful value or are very close to zero. An important feature of LR that makes it convenient is that there is only one parameter that needs optimisation, α, the level of regularisation.

A third method I used was based on artificial neural networks (NN), an approach that has been proven to be accurate in many domains and that has recently gained tremendous amounts of attention, as indicated in Section 2. A NN consists of many layers with node-weights for information processing. The first layer receives the input data, extracts some high level features and passes them to the next hidden layer, and so on, until the final layer which produces the output, that can be a real-value prediction for regression problems, or a probability for classification problems. The more hidden layers a network has, the "deeper" it is. Non-linear functions are applied element-wise, between layers, on the outputs of each node; such functions are usually *tanh, ReLU*, or *softmax*[3] which make NNs a non-linear method. Finally, there are many types of architectures regarding the number of layers, the arrangement of nodes per layer, or the way information flows in the network. I focused

[2]RDkit: Open-Source Cheminformatics Software, `https://github.com/rdkit/rdkit`
[3]`https://en.wikipedia.org/wiki/Activation_function`

only on fully-connected networks where each node of a layer is connected to every node of the next layer. An example of such a NN is illustrated in Fig. 4.

For the application of RF, LR and NN, I use the implementations of Scikit-Learn [15]. As I will use a variety of NNs, the aforementioned one will be denoted as `skNN`. The optimal parameters for each method are selected after a grid-search with cross-validation (CV); more details are given in Section 4.1. Although it offers many methods and tools, Scikit-Learn did not offer many options for designing NNs in particular. Keras [16], on the other hand, is a library specialised for deep learning which offers the ability to define every detail of the model's architecture and training process. Thus, I deployed a fourth STL method, `myNN`, which is a NN with two hidden layers of 200 and 20 nodes respectively – sizes that were selected by hand for a simple yet accurate model – and a third layer with a single output-node. I use `tanh` as activation for the two hidden layers since this function has a wide range of output, something that is useful for dealing with pIC_{50} values. Activation for the last layer is done with a linear function to keep the model simple. The level of regularisation for the weights of each layer was optimised by grid-search and cross-validation.

3.4 Multi Task Learning

The approaches described in the previous section were characterised as STL since they require one model per target, *i.e.* we need to train 110 independent RF, 110 lasso regressors *etc*. Since all these models have the same types of input and output, one question that arises is how we can design one model to capture the important features but adapt to each target individually; multi-task learning (MTL) is one answer. A multitask NN consists of a set of shared layers and a number of *sub-networks* with hidden layers, one per task, which are attached in parallel to the last shared layer, as illustrated in 5. The first part receives the input and extracts basic features, common for every task. Then, each of the sub-networks adapts the model to each target by extracting more appropriate features.

I used one larger NN, referred to as `MTL`, to deal with every task instead of many smaller models. Such a holistic approach is expected to make the analysis easier and offer a significant speed-up in training times. To be more precise, a simple MTL network would need to train $429,410$ weights whereas 110 STL similar models would require more than 22 million parameters. Moreover, programming for the training and evaluation of a MTL method is by far easier, in terms or lines of code, than doing the same for 110 models. Similarly to what I used as a STL approach, I created a network with one shared layer and 110 sub-networks with one short layer (attached to the shared) followed by one single-noded layer for predictions, a model which is presented in Fig 5. Again, the activation functions between the layers are `tanh`. This time I optimised the level of regularisation as well as the sizes of the hidden layers with cross-validation.

3.4.1 Drop-out

Recently, a probabilistic technique called *drop out* has been increasingly used in deep NNs as a regularisation step to avoid overfitting [8, 11]. Here, a node is randomly deleted so the information from that node is not passed on to the next layer. It is usually expressed as a hyper parameter of the model as a drop-rate of, e.g., 30%, between the first two layers means that roughly one-third of the nodes in the first layer will be dropped. Drop out can be applied between any two layers or in combinations of pairs. As this can have a significant effect in performance, drop out needs to be careful selected. To that end, I trained another model, called `MTLD` and a similar to `MTL`, to assess the use of drop out in our problem. `MTLD` includes a drop out on the shared layer and a fixed kernel regularisation on the following layers. Thus, the parameters to select are the sizes of the hidden layers and the dropout rate.

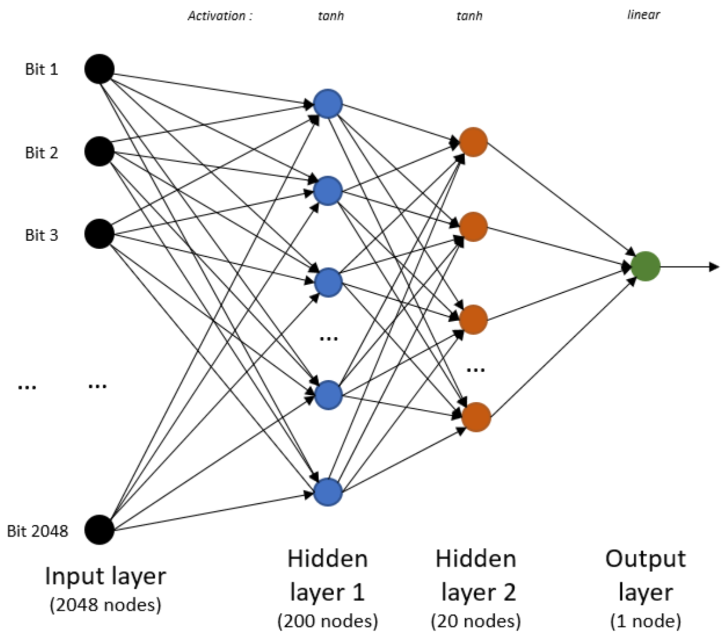


Figure 4: Network architecture for a STL model; 2 hidden layers followed by one node for predictions

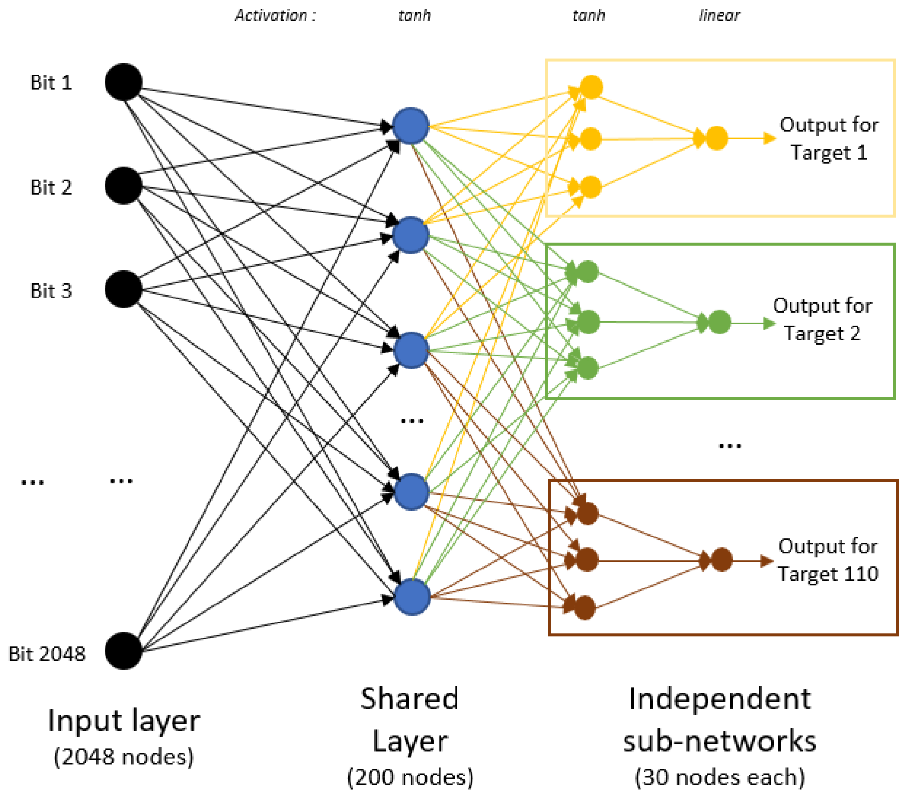


Figure 5: Network architecture for a MTL model; one hidden-shared layer followed by 110 sub-networks attached to it independently. Each of those, represented with boxes, is formed by a hidden layer of 30 nodes and a final layer with one node for predictions.

4 Experimental Evaluation

In this section I present the performance of the aforementioned methods in the prediction of protein-ligand interactions. We start by describing the evaluation methodology, continue by assessing the STL methods and then investigate the advantages offered by MTL. We will also explore the use of drop out and ways to enhance accuracy by self-training.

4.1 Methodology

In order to achieve maximum transparency while training, and fairness during evaluation, I partition the data set in three (disjoint) sets of data. Firstly, I randomly select $3,132$ from the set of active interactions and the same number from the set of inactive, which in total is 10% of the data, and form a test set that will be used for a final assessment. Then I randomly select 20% of the remaining 90% (or 18% of the whole data set) for the validation set and the rest (72%) acts as training data.

Using Scikit-Learn's `GridSearchCV`, I used a 4-fold cross-validation to select parameters for the STL methods through a quick grid search. I selected those that have the highest average accuracy across the 4 folds, then fit each algorithm to the 72% and evaluate with the validation set. For RF I searched for the ideal number of trees in the range $[10, 25, 50, 100, 150, 300]$ and the maximum depth from $[10, 100, 200, 500]$. For the case of `skNN`, the unknown parameters are the number of hidden layers and the number of nodes within each layer. Since training one model for each target is an expensive procedure, I ran a quick search among $[(50), (100, 20), (100, 50), (500, 20, 10)]$, which stand for one, two, two, and three hidden layers respectively. The activation function is set to *tanh* and the selected solver is *Limited-memory BFGS*. For the case of LR I searched for the optimal level of regularisation among the values $[0.01, 0.1, 0.5, 1]$. Similarly for `myNN`, I used the range $[0.001, 0.01, 0.1, 0.2]$ and trained each model for 250 epochs, in batches of 20 (as some targets contain only a few dozens of observations) with a learning rate of 0.001.

For the two MTL approaches, I used a variety of combinations for designing a model, namely the Cartesian product of the sets $[2000, 1000, 300, 200]$ for the shared layer, $[200, 100, 50, 20]$ for the parallel hidden layers and $[0.02, 0.05, 0.1, 0.2]$ for regularisation or $[0.05, 0.1, 0.2]$ for the drop-out. One important difference between the STL versions and these models is that there are tens of thousands of observations available as they are not grouped. To keep training duration on reasonable levels, I used batches of 64 or 128 observations and 30-50 epochs.

Accuracy is measured with the coefficient of determination, a commonly-used metric in regression, defined as:

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2},$$

where y_i is the true value, $\hat{y}_i$ is the corresponding prediction, $\bar{y}$ is the mean of y, and the sum runs for all the observations of a target or for the whole validation set. Values for R^2 range from 1, indicating a perfect fit, and arbitrarily negative values, when prediction is poor. In some cases I also use the mean squared error (MSE), which is also very common.

4.2 Results on Single Task Learning

On a target-wise perspective, Fig. 6a gives some insight on the performance during training for each of the four STL methods and Table 1 summarises the results after all the experiments. The two NN approaches achieve a great fit to the data, with an almost-perfect $R^2 = 0.9930$ for `skNN` and $R^2 = 0.9715$ for `myNN`, confirming what is common experience for such methods. RF comes next with a quite robust R^2 of ~ 0.8976 on average and last is LR which presents high oscillations, with

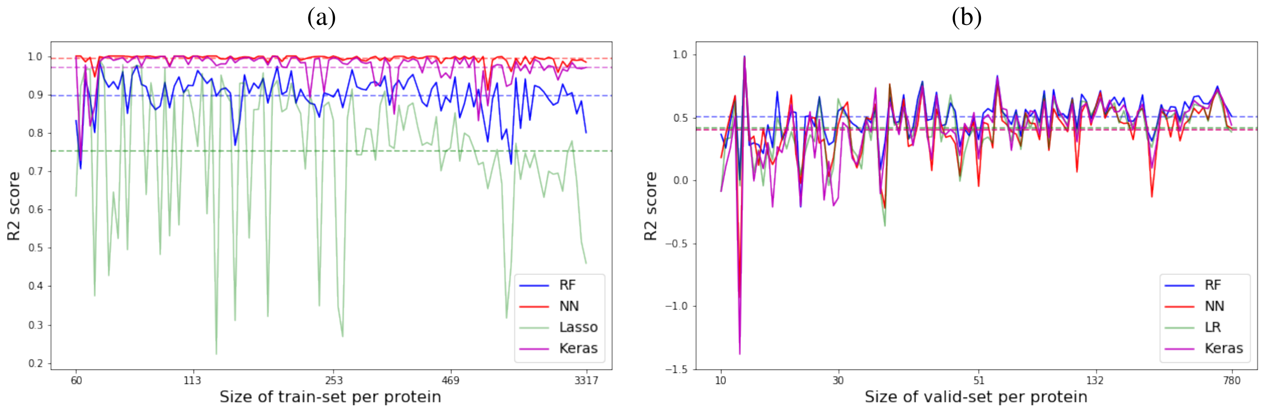


Figure 6: a) R^2 scores for each method and every target after fitting on the training data and selecting the best parametrisation with CV. Horizontal lines indicate the average performance across the 110 targets. b) As in (a) but for the validation set. Targets are ordered with respect to the number of observations in the train set, ranging from 60 to 3317 interactions, and validation sets are proportional. NN methods (red and purple) achieve a near-maximum accuracy for training but fail to generalise during validation. RF and LR behave more consistently. There are a few cases for which each every method has a poor score.

an average of 0.7524 and a decreasing pattern of accuracy as the size of training set is increased, *i.e.* the right-most part of Fig 6a.

Nevertheless, things are quite different when it comes for validation, as Fig. 6b and columns 3 and 4 of Table 1 indicate. RF outperforms every other method with an average R^2 of 0.5014, and a quite narrow standard deviation (std) of 0.1661. NN approaches seem to overfit the training data as their accuracy has a significant decrease on the validation set. In particular, LR comes second with a score of 0.4190 and then are the NN methods with much lower R^2 scores and quite high standard deviations. One reason that all the methods seem unable to generalise effectively is the existence of outliers; there are a few cases for which they all fail dramatically, getting a very low R^2 which pulls the average score down.

The results are more positive on an interaction-wise perspective, *i.e.* without grouping per target but averaging for every observation in the validation set. Ranking changes slightly as the R^2 scores are $0.6474, 0.6081, 0.5911$ and 0.5781 for RF, `myNN`, LR and `skNN` respectively. This means that the outliers are minor and manage to get hidden among a large set of quite good predictions. After comparing `myNN` with `skNN`, it is clear that NNs are prone to overfitting; they can clearly capture perfectly what is expressed in the training data but need careful parametrisation in order to generalise accurately. This is discussed further in comments are given in Section 5.

4.3 Results on Multi Task Learning

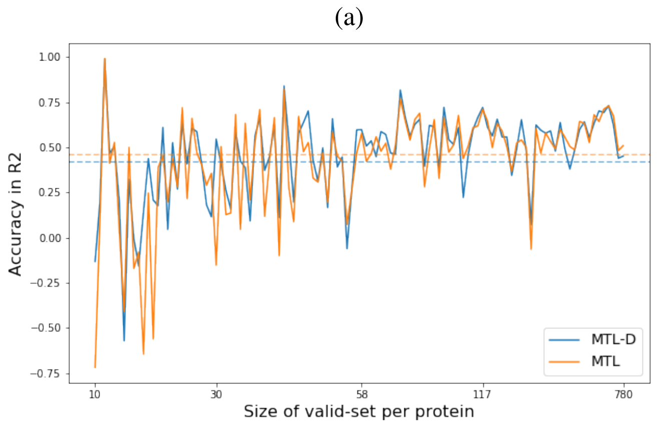
We proceed with the experiments about the advantages of MTL. As indicated earlier, this model ought to be different than the “optimal” STL version and it took many trials to come up with an effective architecture. Training took place using a masked mean squared error as a loss function which effectively omits the missing values and calculates the divergence of the predictions from the true values only for the training points. After fitting a variety of models in the training set – as described in a previous paragraph – I selected as optimal the parametrisation with the maximum mean performance, i.e. the minimum average MSE.

Method	Training R^2, per target	Mean R^2, per target	Standard dev. R^2, per target	Validation R^2	MSE IC_{50}	Duration seconds	Relative speed
RF	0.8976	0.5014	0.1661	0.6474	2.3344	3.6370	145.23
skNN	0.9930	0.3982	0.2390	0.5781	2.7932	0.7278	29.06
LR	0.7524	0.4190	0.2135	0.5911	2.7077	0.1139	4.55
myNN	0.9715	0.3705	0.4499	0.6081	2.5946	4.7783	190.80
MTL	0.7911	0.4572	0.2645	0.6305	2.4469	0.0250	1.00
MTLD	0.9041	0.4395	0.2546	0.6245	2.4861	3.5899	3.59

Table 1: A summary of evaluation for every method. Training, mean and standard deviation are reported after averaging across the 110 targets whereas validation and MSE are averaged across every observation in the validation set. Duration is reported per 1000 predictions, measured on the same machine (LINUX Fedora 30 with Intel® Core i7-9700 CPU and 32GB of RAM), and relative speed is the fraction of duration with respect to that of MTL.

For the MTL, the best performance was obtained using 200 nodes for the shared layer, 30 for each of the independent layers and 0.1 for regularisation, trained with a learning rate equal to 0.0001, for 30 epochs and in batches of 64 observations. MTLD required a 200×20 architecture with a drop out rate of 0.1%, trained similarly to the MLT but for 50 epochs. More details are given in the appendix. The fact that the two models have a similar architecture indicates that drop rate practically replaces the role of kernel regularisation.

Figure 7b, which is similar to that of 6b, shows the performance of the MTL methods after grouping the predictions for each target. It is clear that MTL and MTLD behave similarly with a mean R^2 of 0.4572 and 0.4395 respectively, and a std of 0.2645 and 0.2546 respectively. Moreover, averaging across the validation set, MTL and MTLD achieve a R^2 of 0.6305 and 0.6245 respectively, which is similar to that of RF and even better than every other STL approach. This is a surprising result that validates the effectiveness of multi-task models for protein-ligand binding prediction. Finally, the last column of Table 1, which presents an assessment of the time-complexity of each method, indicates a striking benefit of MTL in terms of computational cost. RF, the leading method in terms of R^2, are two orders of magnitude worse than the MTL approaches in terms of time. I observed similar results regardless the machine I used.



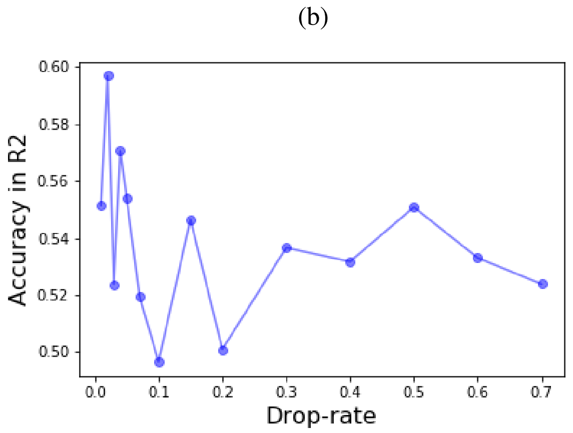


Figure 7: a) Performance of MTL methods on the validation set, similar to Fig. 6b. As the size of the train-set is increased (right-most part of the figure) each model adapts better to the data and generalises more accurately. b) Accuracy of MTLD for a variety of drop out rates and for fixed hidden layers, an experiment conducted after the cross-validation.

4.4 Improving models with self-training

Another experiment concerned the benefits gained by iteratively expanding the training set with confident predictions and re-training. This is essentially the ability of a model to train itself in order to become more accurate. The essential tool here is the complement of each prediction with a level of uncertainty or confidence. There are many techniques to do so, either by modifying the existing method, like adding some type of noise, or by producing multiple copies of the same procedure, like training many models on different subsets of the training set. In any case, each prediction would be an ensemble of the obtained values and the uncertainty could be estimated by the amount of variance or by the width of a confidence interval.

Random forests offer a direct way of producing different estimations of one prediction. They work as an ensemble of decision trees by averaging the individual predictions. Each decision tree acts as an estimator of the prediction and analysing these values can provide the desired confidence. Using the RF models that were trained for the previous task, I assessed their potential to become more accurate. There is a serious drawback for this procedure which lies in the nature of STL; the aforementioned procedure had to run for all 110 targets.

On the other hand, `MTLD` can bypass this bottleneck. This is conditional on a way to assess how confident the NN is for each prediction. Whitehead et al. [12] train twelve separate networks and use the corresponding variance as a level of uncertainty; they claim that this methodology is advantageous as they manage to increase the accuracy of their model. As I think that such an approach loses in efficiency, I turned my attention to drop-out and the seminal work of Gal et al. [17] for studying uncertainty in deep learning. During the testing phase, drop out makes the model stochastic. Repeated runs with the same input will have different nodes drop out, thus the output will be a sum of different weights each time. Provided the model is converged to confident weights, the diversity of predictions for the same input should be small.

For the two approaches, I examined the width of a 95% confidence interval for each prediction and used a threshold to discriminate the confident from the unreliable cases. I imputed the first in the dataset, re-trained each model and repeated until no more predictions are confident or until there are no more missing values. The results are unfortunate for the RF which, no matter the threshold, stays on the same levels of accuracy or behave even worse. On the contrary, `MTLD` is a promising technique since it manages to get an improvement and eventually complete all the matrix with confidence. This resulted in a new method, `MTLD-SF`, a self-trained version of the MTL network which is used further assessed in the next experiment.

4.5 Final evaluation

The last computational task simulates a realistic application where we are asked to extrapolate, i.e. impute the missing values of the so-called DTI matrix. No matter what an algorithm predicts for the potency between a ligand and a protein, there is no way to assess it unless someone runs the corresponding assay. That is the role of the test set that was initially hold out. Normally we should apply only one method – the one that would be our model-solution to the problem – but, as this project aimed at exploring a variety of techniques, I am applying `RF`, `MTL` and `MTLD-SF`, on the 90% of the data set that is not hidden.

Interestingly, the results are similar to those for the validation set indicating that each method is robust enough to generalise. In particular, in terms of R^2, RF obtained an accuracy of 0.6783, the self-trained version of `MTL` had a similar performance with 0.6593 and `MTL` comes next with 0.6303, values which are averaged across the test set. On a target-wise perspective, RF, `MTLD-SF` and `MTL` had an average of 0.5313, 0.4915 and 0.3831 respectively. This is a surprising result which shows that a self-trained MTL model efficient can become very accurate while staying efficient.

5 Conclusions

I have carried out a detailed investigation of regression methods for predicting binding affinity between compounds and targets. This project involved the application of machine learning techniques, including neural networks and multi-task learning, using chemical fingerprints as features for predicting IC_{50} valuesin order to impute missing values in a compound-target matrix. The outcomes of my experiments made clear that the problem is tractable and maybe even solvable, since my framework could be further optimised.

It was clear that random forests and neural networks are more suitable for single targets but for analysing more proteins, like a kinome-wide study, multi-task models are essential. They offer a speed up during training, simpler pipeline implementation, and can achieve equally good performance. Moreover, the drop out, which is mainly used as a tool for regularisation, offers an efficient method to estimate the uncertainty of each prediction.

5.1 Future work

Throughout my study, I worked with fully-connected neural networks, which is maybe the simplest form of deep learning. Thus, one could increase accuracy with more complicated models like Convolutional NN or LSTM. The same holds for the case of random forests for which my optimisation could have been more careful and there are improved and more efficient algorithms available (*e.g.* XGBoost), as well.

I noticed similar results while using feature-based fingerprints (FCFP) but the details are omitted due to space restrictions. Thus, one direction worth exploring is to find out which type, and why, offers more information. Moreover, I believe it is possible to unmask important features with reverse-engineering. For instance, Fig. 8 shows the existence of only a few bits with high impact on regression.

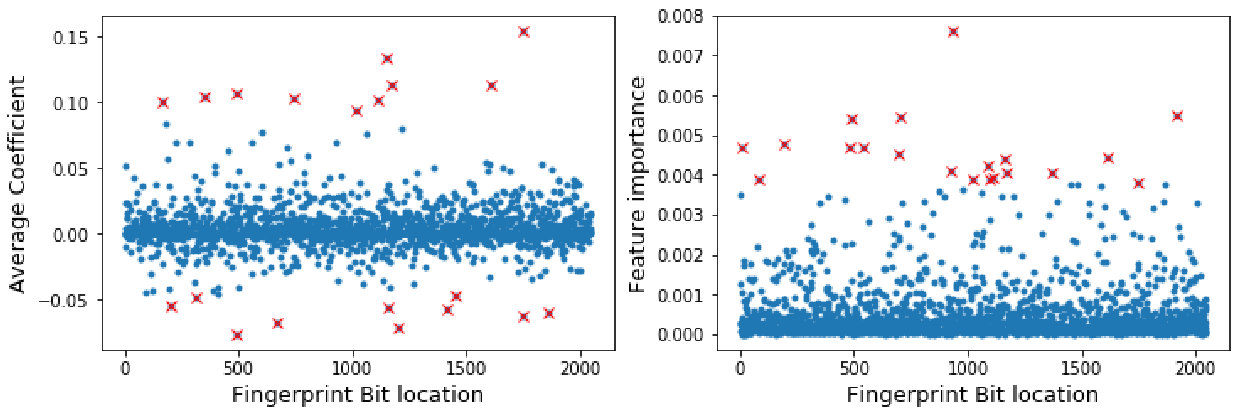


Figure 8: a) Coefficients for ECFP bits obtained by Lasso linear regression, averaged for all target-models. b) Importance of features obtained by analysing RF models. In both cases, the vast majority is close to zero except some with consistently larger values, which are shown with crosses.

Finally, another interesting extension of this project would be the application of this framework on another dataset. Either to a different set of proteins (like GPCRs) or to another data set, like the one proposed by Martin et al. [5] with 159 assays from ChEMBL. The latter would make possible a direct comparison of multi-task learning with other previously published techniques.

References

[1] A. S. Rifaioglu, H. Atas, M. J. Martin, R. Cetin-Atalay, V. Atalay, and T. Dogan, "Recent applications of deep learning and machine intelligence on *in silico* drug discovery: methods, tools and databases," *Brief. Bioinform*, vol. 10, 2018.

[2] A. Gaulton, L. J. Bellis, A. P. Bento, J. Chambers, M. Davies, A. Hersey, Y. Light, S. McGlinchey, D. Michalovich, B. Al-Lazikani, *et al.*, "ChEMBL: a large-scale bioactivity database for drug discovery," *Nucleic Acids Research*, vol. 40, no. D1, pp. D1100–D1107, 2011.

[3] A. P. Bento, A. Gaulton, A. Hersey, L. J. Bellis, J. Chambers, M. Davies, F. A. Krüger, Y. Light, L. Mak, S. McGlinchey, *et al.*, "The ChEMBL bioactivity database: an update," *Nucleic Acids Research*, vol. 42, no. D1, pp. D1083–D1090, 2014.

[4] F. A. Sorgenfrei, S. Fulle, and B. Merget, "Kinome-wide profiling prediction of small molecules," *ChemMedChem*, vol. 13, no. 6, pp. 495–499, 2018.

[5] E. J. Martin, V. R. Polyakov, L. Tian, and R. C. Perez, "Profile-QSAR 2.0: Kinase virtual screening accuracy comparable to four-concentration ic$_{50}$s for realistically novel compounds," *Journal of Chemical Information and Modeling*, vol. 57, no. 8, pp. 2077–2088, 2017.

[6] M. C. Cobanoglu, C. Liu, F. Hu, Z. N. Oltvai, and I. Bahar, "Predicting drug–target interactions using probabilistic matrix factorization," *Journal of Chemical Information and Modeling*, vol. 53, no. 12, pp. 3399–3409, 2013.

[7] B. M. Sarwar, G. Karypis, J. A. Konstan, J. Riedl, *et al.*, "Item-based collaborative filtering recommendation algorithms.," *10th Intern. World Wide Web Conference*, pp. 285–295, 2001.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[9] E. B. Lenselink, N. Ten Dijke, B. Bongers, G. Papadatos, H. W. Van Vlijmen, W. Kowalczyk, A. P. IJzerman, and G. J. Van Westen, "Beyond the hype: deep neural networks outperform established methods using a ChEMBL bioactivity benchmark set," *Journal of Cheminformatics*, vol. 9, no. 1, p. 45, 2017.

[10] L. J. Colwell, "Statistical and machine learning approaches to predicting protein–ligand interactions," *Current Opinion in Structural Biology*, vol. 49, pp. 123–128, 2018.

[11] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, and V. Pande, "Massively multitask networks for drug discovery," *arXiv preprint arXiv:1502.02072*, 2015.

[12] T. Whitehead, B. Irwin, P. Hunt, M. Segall, and G. Conduit, "Imputation of assay bioactivity data using deep learning," *Journal of Chemical Information and Modeling*, vol. 59, no. 3, pp. 1197–1204, 2019.

[13] M. Davies, M. Nowotka, G. Papadatos, N. Dedman, A. Gaulton, F. Atkinson, L. Bellis, and J. P. Overington, "ChEMBL web services: streamlining access to drug discovery data and utilities," *Nucleic Acids Research*, vol. 43, no. W1, pp. W612–W620, 2015.

[14] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[17] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *International Conference on Machine Learning*, pp. 1050–1059, 2016.

Appendix I

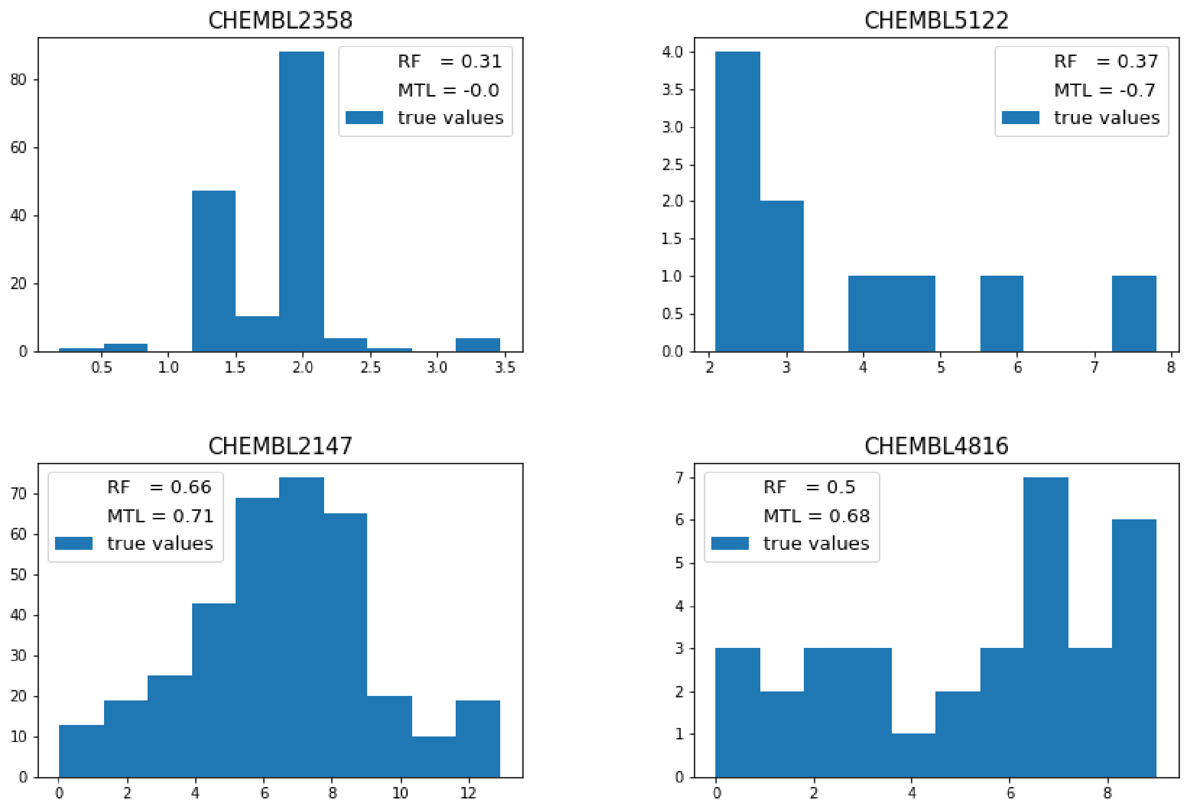


Figure 9: A few representative cases for which the MTL method fails dramatically to produce meaningful prediction, or outperforms the standard approach. These figures show the distribution of pIC_{50} values, clustered in 10 bins, for four targets along with he accuracy, in terms of R^2, for `MTL` and RF. The key observation is the different ways the values are distributed as well as the sizes of bins.

Appendix II - Keras implementations of MTL

Here I collocate my code for implementing the multi-task neural networks in Keras. The `MTL` and `MTLD` models are defined as functions, taking as input the sizes of each layer, the learning rate and the level of regularisation or drop out respectively. Both models require a custom loss function which bypass the missing values for calculating the MSE.

```
def masked_loss_function(y_true, y_pred, MissingVal=10):
    # This function masks the elements of the vectors with true/predicted values
    # so that the model focuses only on the known data.
    # By default, missing values are represented by 10
    mask = K.cast(K.not_equal(y_true, MissingVal), K.floatx())
    return keras.losses.mean_squared_error(y_true * mask, y_pred * mask)

def MTL( wsl=200, whl=30, lamda=0.02, lr=0.0001):
    # This function defines a MTL regression model given the architecture
    # define the input layer:
    inputs = keras.Input(shape=(2048,))
    # define the first shared layer:
    sharedlayer = keras.layers.Dense(wsl, activation='tanh',
                  kernel_regularizer=regularizers.l2(lamda) )(inputs)
    # define the way weights will be initialised:
    myinit = keras.initializers.Constant(4.)
    hidden = []
    for i in range(len(Targets)):
        # start attaching sub-networks to the shared layer
        hl = Dense( units=whl, activation='tanh',
                    kernel_regularizer=regularizers.l2(lamda) )(sharedlayer)
        hidden.append( Dense(1, kernel_initializer=myinit,
                       activity_regularizer=regularizers.l1(0.0001) )(hl) )
    # define the model and compile it:
    MTL = Model(inputs=inputs, outputs=hidden)
    MTL.compile(loss=masked_loss_function, optimizer=keras.optimizers.adam(lr=lr))
    return MTL

def MTL_Drop( wsl=200, whl=20, drop_rate=0.1, lr=0.0001):
    # a function that creates a NN with dropout between the first two layers
    # define the input layer:
    inputs = keras.Input(shape=(2048,))
    sharedlayer = keras.layers.Dense(wsl, activation='tanh' )(inputs)
    dropout= keras.layers.Dropout(drop_rate)(sharedlayer, training=True)
    # define the way weights will be initialised:
    myinit = keras.initializers.Constant(4.)
    hidden = []
    for i in range(len(Targets)):
        # start attaching sub-networks to the shared layer
        hl = Dense( units=whl,  activation='tanh',
                    kernel_regularizer=regularizers.l2(0.05) )(dropout)
        hidden.append( Dense(1, kernel_initializer=myinit,
                       activity_regularizer=regularizers.l1(0.0001) )(hl) )
    # define the model and compile it:
    MTL=Model(inputs=inputs, outputs=hidden)
    MTL.compile(loss=masked_loss_function, optimizer=keras.optimizers.adam(lr=lr) )
    return MTL
```

Appendix III - Results of cross validation for MTL models

MSE after training `MTL` with a variety of parametrisations and validating with 25% of the training data. Each tuple contains values for the width of the shared layer, the width of each independent layer and the level of regularisation.

```
(200,20,0.02)  :  0.1075
(200,20,0.05)  :  0.0873
(200,20,0.1)   :  0.0866
(200,30,0.02)  :  0.1306
(200,30,0.05)  :  0.0953
(200,30,0.1)   :  0.0852
(300,20,0.02)  :  0.1043
(300,20,0.05)  :  0.0864
(300,20,0.1)   :  0.0868
(300,30,0.02)  :  0.1260
(300,30,0.05)  :  0.0917
(300,30,0.1)   :  0.0862
(400,20,0.02)  :  0.1037
(400,20,0.05)  :  0.0859
(400,20,0.1)   :  0.0868
(400,30,0.02)  :  0.1227
(400,30,0.05)  :  0.0901
(400,30,0.1)   :  0.0867
```

MSE after training `MTL` with drop out with a variety of parametrisations and validating with 25% of the training data. Each tuple contains values for the width of the shared layer, the width of each independent layer and the drop out rate.

```
(2000,100,0.05)  :  0.5560
(2000,100,0.1)   :  0.4716
(2000,100,0.2)   :  0.5260
(2000,50,0.05)   :  0.2247
(2000,50,0.1)    :  0.2669
(2000,50,0.2)    :  0.2904
(2000,20,0.05)   :  0.1409
(2000,20,0.1)    :  0.1267
(2000,20,0.2)    :  0.1609
(300,100,0.05)   :  0.3653
(300,100,0.1)    :  1.1295
(300,100,0.2)    :  1.4604
(300,50,0.05)    :  0.1445
(300,50,0.1)     :  0.1231
(300,50,0.2)     :  1.0087
(300,20,0.05)    :  0.1159
(300,20,0.1)     :  0.0992
(300,20,0.2)     :  0.0995
(200,100,0.1)    :  0.1576
(200,100,0.2)    :  3.7611
(200,50,0.05)    :  0.2999
(200,50,0.1)     :  0.0985
(200,50,0.2)     :  0.1284
(200,20,0.05)    :  0.0935
(200,20,0.1)     :  0.0917
(200,20,0.2)     :  0.1125
```