Machine Learning for Bioinformatics Project, Phase 2

Alireza Akbari
95105379

Question 1

Pros:

- One of the major challenges in the previous phase of the project was to extract a proper set of features to utilize them in the machine learning model (XGBoost). We tried to address this problem by feature engineering then. However, there is no guarantee that the produced features would be effective for the regressor and consequently the classifier. But in DeepDTA, which is based on a deep neural network, the model itself is responsible to extract proper representations. The end-to-end learning process of deep learning has been one of its major advantages, which has revolutionized numerous areas such as Deep Reinforcement Learning, Image Classification, etc. Hence, in order to extract the proper features, one just has to provide an initial embedding (representations) for the proteins and drugs, and then, the model would extract features in two independent branches for both proteins and drugs. Subsequently, after combining the two learned representations, one can solve the regression problem by a set of consecutive fully connected layers.

- Another advantage of this paper is that they solved the continuous problem, meaning that they tried to predict the exact number of the affinity instead of reducing the problem to a binary classification problem from the beginning. This is because previous works have shown that solving the classification problem can lead to the loss of information in the analysis of the problem.

Cons:

- The major drawback with the model is the proposed blocks which have the responsibility to learn the representations of proteins and drugs. In DeepDTA, they take advantage of consecutive 1d Convolutional Neural Networks to produce feature maps from the initial embeddings. Generally speaking, CNN blocks are powerful to capture the local patterns of the input. However, proteins and drugs have their own specific structures, meaning that the local patterns would not be necessarily sufficient to extract the efficient set of features. As an illustration, a drug has a chemical structure which can be depicted by a graph. Consequently, although the local patterns of the drug are important (e.g. the local chemical bonds), there are more global patterns that cannot be captured by CNNs. Similarly, proteins have a sequential nature (a sequence of amino acids) which as we know can be best captured by recurrent neural networks (to consider the whole sequence, not the local ones).

- The sequence size of different drugs and proteins are not the same. However, as they utilized CNNs for the feature extractor parts of the model, they had to resize all of the drugs and all of the proteins to a fixed size, leading to a cut for some segments of the input, which in turn can deteriorate the final learned representations.
- This work only takes advantage of the raw data of the drugs and proteins. The general function of a drug is highly dependent on the chemical characteristics of the existing atoms. Leveraging extra information about atoms, chemical bonds, and etc. can be overly effective to extract rich meaningful features.

Question 2

As I mentioned, drugs have a graphical structure which cannot be captured effectively by CNNs. Recent advances in Representation Learning have proposed that graph neural networks (GNNs) are powerful to learn representations of graphical data such as syntax trees, molecules, and etc.. Therefore, one way to improve the model is to replace the CNN blocks with GNN blocks. Generally speaking, GNNs are a more general structure than RNNs where instead of a linear graph (unwrapped through time) we have a more general graph with arbitrary edges between nodes. In this structure, each node has a hidden state which is updated in each time step by aggregating the messages from neighbors and its own previous state. Consequently, when we iteratively apply the updating process, the nodes start to get aware of all of the nodes and learn the general structure of the graph over time. By reviewing the literature, I found two works taking advantage of the same idea to learn the graphical structure of drugs called GraphDTA and DeepGS. I read the GraphDTA paper and take it as the baseline of my study. In the paper, they proposed four variants of GNNs called Graph Convolution Network, Graph Attention Network, Graph Isomorphism Network, and GAT-GCN which are different in their message passing and updating essentially. Based on their results, the GIN models have achieved the best performance on the regression task. The principal updating rule in GIN is:

$$X_i^{(t+1)} = MLP((1+\epsilon)X_i^{(t)} + \sum_{j \in B(i)} X_j^{(t)})$$

where X represents the hidden state, B(j) is the set of neighbor nodes of X_i, ϵ is a learnable or a fixed parameter, and MLP is a multilayer perceptron. They proposed to stack five layers of GIN block and as it can get excessively deep, they take advantage of batch normalization to facilitate the learning procedure. In the final layer of GIN, in order to reach an aggregated representation of the whole graph, they proposed to use a global add pool, which simply aggregates the whole graph by adding node feature across the node dimension.

For proteins, GraphDTA still utilizes CNNs to extract features. In order to test LSTMs, at first I changed the corresponding part of the model in DeepDTA to LSTM. However, I could not reach the result of the DeepDTA at least in the same number of epochs. Additionally, I did the same to the corresponding part of GraphDTA, and it did not lead to any improvement as well, at least with the limited number of epochs. This observation is can be consistent with the results of three papers: DeepDTA, GraphDTA, and DeepGS since none of them took advantage of Recurrent Neural Networks for learning representation for proteins. However, at the end of this report, we will gain insights that can possibly demonstrate why the current CNN model would not be an effective candidate either.

It is worth mentioning that since proteins have a sequential structure as I mentioned, a semantically meaningful embedding layer can be highly effective in the learning procedure (similar to Word2Vec for the words). Hence, I used an embedding layer for the proteins with learnable parameters. There are also advanced embedding layers in this context such as Prot2Vec which has been used by the DeepGS model.

Question 3

We want to apply graph neural networks to the drugs and initially, the drugs are in the SMILES code. PyTorch Geometric is a famous extension for geometric deep learning. In order to convert the SMILES code to the proper objects acceptable by PyTorch Geometric, first of all, we have to convert the SMILES code to the molecule data structure using rdkit package. Then we can extract the nodes (with their features) and the edges of the molecule. For each node, there are five types of features extracted: the atom symbol, the number of adjacent atoms, the number of adjacent hydrogens, the implicit value of the atom, and whether the atom is in an aromatic structure. Now, in PyTorch Geometric, there exist three classes DataLoader, InMemoryDataset, and Data to handle graphical data. In a nutshell, Data.x should contain the atoms and their features, and Data.edge_index should contain the edges (adjacency matrix). Since we have the affinity for the pair of drug and protein, we embed another attribute to the Data class named target. The corresponding affinity should be placed in Data.y.

After converting each pair to a Data object, we can design our dataset by inheriting it from InMemoryDatset, which is used when the whole dataset can fit in the CPU Memory, adding all of the pairs in a dataset. Finally, we can establish the DataLoader in PyTorch Geometric by providing the created dataset for

Accuracy	Sensitivity	Specificity	F1
0.946707	0.560976	0.981087	0.632737

Table 1: Results on Davis Dataset

the DataLoader. (Notice that InMemoryDataset and DataLoader in PyTorch Geometric are not identical to the similar concepts in classical PyTorch because there are a lot to configure in the graphical version)

There is not any new specific preprocessing for the proteins. Similarly, each amino acid will be represented by a 128d vector (embedding) and all of the proteins are resized to a fixed size (with length = 100). Shorter sequences are zero-padded and the longer ones are truncated.

Question 4

The number of epochs used to train the model was 100, but as it is obvious from the log of the training, the model has still the capacity to learn more. On the Davis dataset, the model after the 100th epoch reached: $CI = 0.8578$ and $MSE = 0.2900$ and the best model (on validation data) reached $CI = 0.8679$ and $MSE = 0.2919$. In the DeepDTA model, they reached $CI = 0.878$ and $MSE = 0.261$ on 100 epochs. However, in the graphDTA, their results were: $CI = 0.893$ and $MSE = 0.229$ but they did not report the number of epochs. Nevertheless, by the behavior I saw on the training procedure, it seems one can reach the mentioned result with more than 100 epochs. I only trained the model on Davis Dataset since the KIBA is considerably large, making it hard to train on google colab.

Question 5

The loss function in the previous section was designed to solve a regression task. Now, in order to determine whether a drug and protein would bind, we use a known threshold to classify the pairs into two groups. The known threshold for Davis dataset is 7. Accordingly, the classification metrics are in table 1.

As we observe in table 1, the sensitivity of the model is considerably low, which means that the model has a bias towards the negative class. In the following sections, we try to address the problem.

Question 6

In the figure 1, we see that both datasets are unbalanced and the number of data with negative labels is much higher than the positive ones (the threshold for KIBA dataset is 12.1 and for Davis is 7). However, in the regression task, it is not depicted very well because the model can reach a favorable MSE loss on the majority of the data.

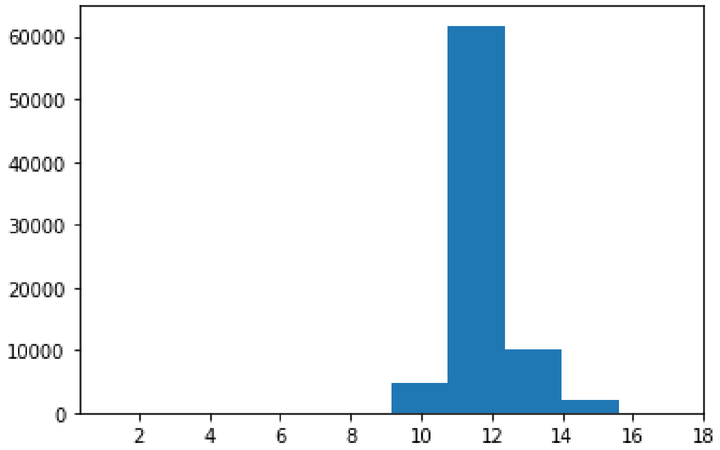


Figure 1: Distribution of Affinities in Kiba Dataset

Accuracy	Sensitivity	Specificity	F1
0.944311	0.612195	0.973913	0.642766

Table 2: Results on Balanced Davis Dataset

A possible candidate to address the problem is to make the dataset balanced by upsampling, meaning that randomly sample from the minor class, whereby, augmenting the dataset. In this case, the model cannot reach a favorable MSE loss anymore. The amount of upsampling is a hyperparameter which can be fine-tuned; however, because of the time limit, I could not test different options. The design choice of upsampling is 0.9, meaning that:

$$|+class| = 0.9 \times |-class|$$

Now, the classification metrics are in table 2. By fine-tuning the amount of upsampling, one can expect further improvement of the F1 measure.

Question 7, 8

From now on, we choose the upsampled model for further investigation.

Note that in the suggested methods for interpretation, the gradient would backpropagate from the correct label. However, our model is essentially a regressor, not a classifier. Hence, we freeze the whole model, cut out the last regressor node, and attach a layer with two neurons that have the responsibility to compute logits for each class. Subsequently, we train the model to learn the weight parameters of the last layer while the rest of the model is frozen. The loss function in this setting is cross-entropy, and since this loss function can be weighted, we assign a larger weight to the positive class.

Now, I explain the fundamentals of the suggested interpretability methods:

- **Saliency Maps**: In this method, our eventual goal is to see when the input and its corresponding label (class) is fixed, which features of the model have the most impact on the current behavior of the network. It is stated in the paper that consider the simplest case namely the perceptron. In this case, the features which have the biggest corresponding weight would have the most impact on the final decision. The formulation in this setting would be:
$$Out_c(I) = w^T I$$
where I is the fixed input. However, in a deep neural network, the relation between input and the corresponding output is overly complex because of the nonlinearities. Nevertheless, one can loosely approximate the relation by first-order Taylor expansion:
$$Out_c(I) = \frac{\partial Out_c}{\partial I}|_{I_0} I$$
which is easy to compute because the derivative part can be computed by backpropagation easily

- **Guided Back Propagation**: This work combines the idea from Saliency Maps and taking advantage of the deconvolution layer. Generally speaking, a deconvolution layer tries to invert a convolution layer. In this work, they proposed that for blocks other than the activation layer, just do the same as saliency maps, but while facing an activation layer like relu, apply the combination of backpropagation and deconvolution rules to obtain the backpropagated vector. As it is clear in figure 2.[1] the entries of the backpropagated vector will get zero both by backpropagation (the entries which were zero in the forward pass) and by deconvolution (the entries which were zero in the backward pass). It was practically shown that this method would be more stable to find the most impactful features.

- **LRP**: The fundamental problem with Saliency Maps is that the gradient can be noisy, making the reported features unreliable. In the guided backpropagation, they tried to alleviate the problem slightly. The idea of LRP is instead of backpropagating the gradient, one can backpropagate a score in a stable manner. Their idea, which is similar to the conservation law in Kirchhoff's circuit laws, is that the backpropagated score which reaches a specific neuron has to be equal to the sum of the scores which backpropagated from the neuron to its previous layer. figure 3 depicts the concept clearly.[2]It can be reformulated in the following way:
$$f(x) = \sum_{k=1}^{n(L)} R_k^L = \cdots = \sum_{k=1}^{n(l)} R_k^l = \cdots = \sum_{k=1}^{n(1)} R_k^1$$

[1] Taken from **STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET** by Springenberg et. al.

[2] Taken from **Explainable AI:Interpreting, Explaining and Visualizing Deep Learning** book by Samek et. al.

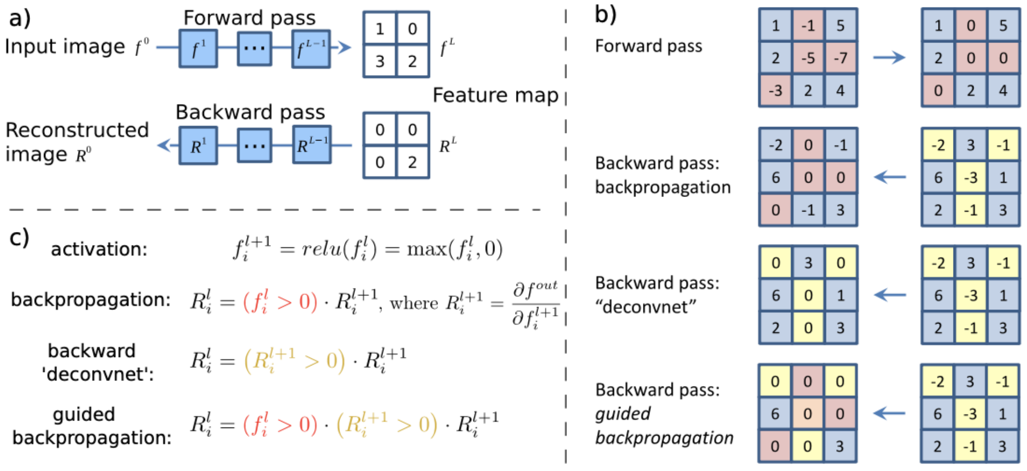

a)
Input image f^0
Forward pass
f^1
f^{L-1}
1 0
3 2
f^L
Feature map
Reconstructed image R^0
Backward pass
R^1
R^{L-1}
0 0
0 2
R^L
b)
Forward pass
1 -1 5
2 -5 -7
-3 2 4
1 0 5
2 0 0
0 2 4
Backward pass: backpropagation
-2 0 -1
6 0 0
0 -1 3
-2 3 -1
6 -3 1
2 -1 3
Backward pass: "deconvnet"
0 3 0
6 0 1
2 0 3
-2 3 -1
6 -3 1
2 -1 3
Backward pass: *guided backpropagation*
0 0 0
6 0 0
0 0 3
-2 3 -1
6 -3 1
2 -1 3
c)
activation: $f_i^{l+1} = relu(f_i^l) = \max(f_i^l, 0)$
backpropagation: $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$, where $R_i^{l+1} = \frac{\partial f^{out}}{\partial f_i^{l+1}}$
backward 'deconvnet': $R_i^l = (R_i^{l+1} > 0) \cdot R_i^{l+1}$
guided backpropagation: $R_i^l = (f_i^l > 0) \cdot (R_i^{l+1} > 0) \cdot R_i^{l+1}$

Figure 2: How guided backpropaagation works

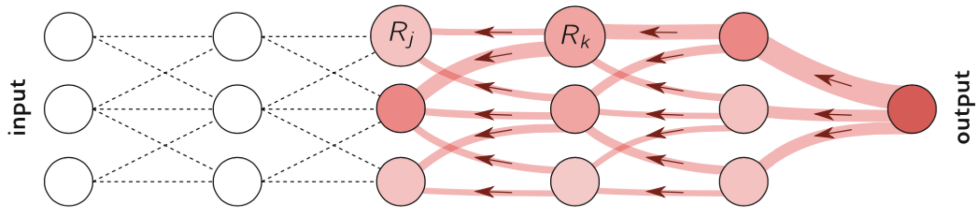


Fig. 10.2. Illustration of the LRP procedure. Each neuron redistributes to the lower layer as much as it has received from the higher layer.

Figure 3: How LRP backpropagates its score

where f(x) is the prediction of the fixed class, $n(l)$ is the number of neurons in the l-th layer, and R_k^l is the score received by k-th neuron in the l-th layer. This constraint can make the score backpropagation more stable than gradient backpropagation. Now, it is non-trivial to find a proper equation for R which not only satisfies the constraint but also leads to the identification of the actual impactful features at the end. The equation they used is:

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k$$

where two neurons j and k are connected in two consecutive layers (neuron j is in the previous layer which consists of neuron k), a_j is the activation of the neuron j, and w_{jk} is the weight of the edge between j and k. The equation is normalized to satisfy the conservation law. By backpropagating based on this equation from the last layer, one can find the important features.

Question 9

I have implemented Saliency Maps and Guided Backpropagation (got help from here).

The backpropagated gradient for the protein cannot backpropagate through the embedding layer; therefore, the gradient for the proteins is actually the gradient with respect to the embeddings (whose number is 128 per amino acid). Additionally, the gradient for a drug node would be a vector with a shape equal to the number of the features of the node (chemical features whose number is 78). Thus, in proteins, for each amino acid, we have a gradient vector with the shape $[128, 1]$ and a $[78, 1]$ vector for each node in the drug. Subsequently, we have to somehow aggregate these vectors independently to acquire a corresponding number for each amino acid and each node in the molecule.

Now, we have to somehow come up with an idea which determines how many of the inputs with the highest corresponding gradients should be selected as the most effective ones in the classification. The first idea that comes to mind is to choose the ones which are greater than the average. However, the first moment cannot be a reliable statistic since it does not care about the variance. Accordingly, I chose to select the features which are greater than $mean + std$.

Furthermore, by finding the desired features, one can manipulate them to see what happens to the prediction of the model. Therefore, we take the data points which are correctly classified by the model and zero the identified features, once for protein and once for the drug. The result is in table 3(note that the number of True Positives is 152 while the number of total positive cases in the test dataset is 410) , which shows that Guided BackProp was more successful to identify the more impactful features.

A key insight one can gain is that it seems that the model is not reasonably dependent on the proteins since the manipulation of their important features does not lead to misclassification in contrast to the drugs. Consequently, an

method	# of changes in classification (protein)	# of changes in classification (drug)
Saliency Maps	1	113
Guided BackProp	3	132

Table 3: Results of the changed classified data after the manipulation of the input

interesting question to think about for future works on this problem can be addressing this problem. Note that one can doubt this claim by mentioning that maybe the number of changed features in drugs is significantly higher than the similar number in the proteins. By plotting the distribution of the number of changed features for both drugs and proteins (both in Saliency Maps and Guided BackProp), one can see that it is not the case. This can be seen in table 4.

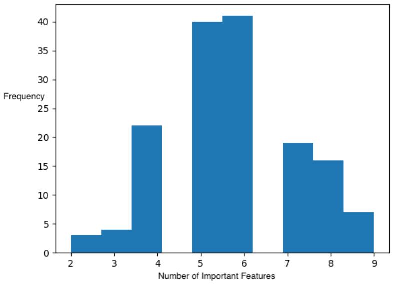
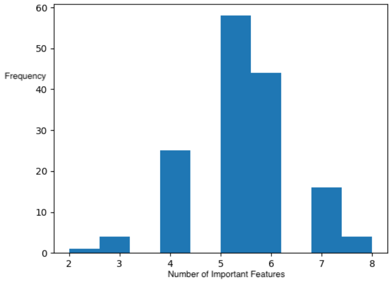
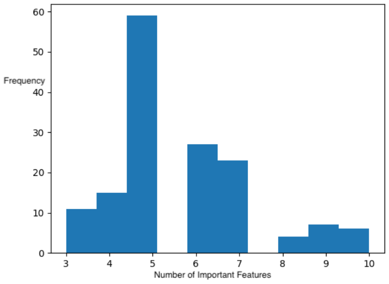
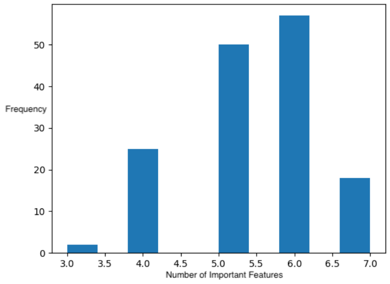
method	Important Features (Drug)	Important Features (Target)
Saliency Maps		
Guided BackProp		

Table 4: The distribution of detected features, because of their importance. It also shows that how many features are set to zero to find out whether the classification would change or not.