Retrosynthetic Planning

Kewei Zhao, Mabiao Long and Yichuan Peng

June 16, 2023

Abstract

As one of the fundamental problems in organic chemistry, retrosynthetic is aimed to identify latent reactants from given products, so that scientists can discover new paths to generate chemistry molecules. Recently, this retrosynthetic process can be decided by reaction templates and Morgan Fingerprints. In this project, we explore the methods of single-step retrosynthesis prediction and molecule evaluation. Then we try multi-step retrosynthetic process on Retro* to view the performance of combining single-step retrosynthetic and molecule evaluation.

1 Introduction

Retrosynthesis planning is one of the key issues in organic chemistry. Due to the thousands of possible reactions at every step of the organic theory, the search space for design is enormous, which makes the problem challenging even for experienced chemists.

In this report, we take three step to explore, analysis and solve this problem to some extent.

In Task 1, we focused on single-step retrosynthesis prediction using a Multi-Layer Perceptron (MLP) neural network. We constructed our dataset by using products as inputs and reactants as labels. Two strategies were explored for molecule representation: fingerprints and molecular formulas. Our evaluation revealed that using fingerprints resulted in higher prediction accuracy compared to molecular formulas. This highlights the importance of extracting meaningful molecular features for successful retrosynthesis prediction. Our findings contribute to the field of chemical informatics and have implications for automated retrosynthesis planning in drug discovery and organic synthesis.

Task2 focuses on the evaluation of the synthetic cost of molecules. For a specific reaction, we need to know the reaction conditions, which can help us better decide the steps. Cost is one of the main dimension to evaluate the feasibility of reactions. In this project, we will explore two parts of molecule evaluation: single molecule and multiple molecules. Single molecule evaluation focus on the synthetic cost of one molecule, regardless of other conditions. But for multiple molecule evaluation, relationship between different molecules (e.g. structures, atoms or mutual reactions) also needs to be considered, which bring more challenge to our task.

Task3 uses Retro* algorithm [CLDS20] to solve the Multi-Step retrosynthesis planning problem. It is a neural-based A* like algorithm and gets highest efficiency compared to other existing algorithms. In the given test data, the success rate of finding synthesis path reached 52% in 100 epochs.

2 Methods

2.1 Task1: Single-step retrosynthesis prediction

In this part we introduce Multi-Layer Perception(MLP) to help solve this classification problem.
To begin with, we performed data preprocessing on the raw data, leveraging the rdchiral library. This allowed us to break down the chemical reaction formulas into separate molecule formulas for both products and reactants. Additionally, we utilized the rdkit library to generate fingerprints for each product. Subsequently, we used the molecule formulas to construct our dataset.

For the input data, we translated the molecule formulas into arrays of ASCII codes for each character. These arrays served as our inputs. To facilitate classification, we created a dictionary to track each template, which represented the categories we aimed to classify. Using the PyTorch framework,

we developed our Multi-Layer Perceptron (MLP) network. We employed the ReLU activation function in our network.

After training the network for 100 epochs, we achieved a classification accuracy of 22.78%. We also experimented with other activation functions, but they yielded similar results.

Subsequently, we attempted using the fingerprints of the molecules to build our dataset. However, the final results were disappointing. Despite the high-dimensionality (2048 dimensions) of the fingerprints, we observed an accuracy of only 3.17%. This outcome was unexpected, as we anticipated that the rich feature representation offered by high-dimensional fingerprints would significantly improve performance.

2.2 Task2: Molecule evaluation

In this part we apply several methods to do single and multiple molecule evaluation, including linear regression, cosine similarity and MLP. In section 2.2.1, we will introduce all three methods mentioned to evaluate single molecule evaluation; and in section 2.2.2, we will introduce MLP model used in both ways of multi cost evaluation.

2.2.1 Single molecule evaluation

Single molecule evaluation is a regression task to predict the synthetic cost of given molecule. Each molecule is represented by a Morgan Fingerprint, which suggests the molecule's atoms and structures. Therefore, we can regard the fingerprints as sample data and corresponding costs as sample label, and construct our datasets for training and testing.

Here, each fingerprint is a $[1, 2048]$ tensor and its elements are assigned to 0 or 1, and each cost is an non-negative real value. Thus molecule evaluation task is a multivariate regression task, which predicts the dependent variable from multiple arguments.

(1) Linear regression

Linear regression is a statistical analysis method that uses regression analysis in mathematical statistics to determine the interdependent linear relationship between two or more variables. Here we use linear regression to analyse the relationships from fingerprint to cost. The main steps are shown as below:

1. Create a linear regression model.
2. Fit the model by training data(fingerprints) and label(costs).
3. Predict testing data costs on the model, and calculate the square error $||\hat{y} - y||^2$.

To further evaluate the performance of our model, we calculate the "accuracy" of regression result by loss. If $loss_i < 0.01$, we believe the model well predicts the cost of sample i. By this method, we can also get an accuracy result, which will be shown in section 3. However, although we introduce "accuracy" to help to evaluate, loss is still the main principle to measure the performance.

(2) Cosine similarity

Cosine similarity is one of the algorithms in comparing the difference between two vectors. In this task, cosine similarity of the fingerprints can be used to describe the relationship between molecules. Generally, molecules with larger cosine similarity have similar fingerprints, and therefore have similar synthetic costs as well.

In this task, we use cosine similarity directly to assign train data i's cost to test data j if j has the largest similarity with i among all training data. By this method, some costs can be accurately predicted (e.g. those molecules with $cost = 0$), but it's also hard to catch crucial information for molecules with large costs. Detailed experimental results and analysis will be introduced in 3.

(3) MLP

MLP can be designed to solve regression tasks as well. In this task, we construct a simple MLP with two hidden layers. The structure of MLP is:

- $layer1 = nn.Linear(input\_dim, 512)$
- $layer2 = nn.Linear(512, 32)$
- $output\_layer = nn.Linear(32, 1)$

Here, we use $nn.ReLU()$ as the activation function for all layers. In general $sigmoid()$ or $tanh()$ are used in the last layer; however in this task, the predicted cost should be non-negative, therefore $ReLU()$ is more suitable. MLP performs better than linear regression as well as cosine similarity.

2.2.2 Multiple molecule evaluation

To get multi-molecule products, we can randomly sample from different fingerprints. Here we generate 10000 samples, each consists of 2 to 5 different molecules. A paired sample is represented as:

$$[data, label] = [[fp_1, fp_2, ..., fp_N], total\_cost] \tag{1}$$

where fp_i is the fingerprint of the i-th molecule, and $total\_cost = \sum_{i=1}^{N} cost_i$ is the sum of all molecules.

For multi-molecule evaluation, one feasible method is to predict the cost of each molecule separately and sum them up. By this way, we can use the model mentioned in section 2.2.1 to train on single molecule's cost. To deal with multiple molecules, we can just split data into single molecules and then predict the cost for each molecule.

$$cost = f(m_1) + f(m_2) + ... + f(m_n) \tag{2}$$

Another way is to design a transform function $f : [fp_1, fp_2, ..., fp_N] \mapsto [Fp]$. This function should preserve origin information of each fingerprint, and imply more information which is easy to train and distinguish. In this task, we pile and sum all fingerprints up to get Fp. After this step, the input data is still a $[1, 2048]$ tensor and can be trained by our MLP model.

$$cost = f(m_1, m_2, ..., m_n) \tag{3}$$

In general, equation 2 only sums the costs of single fingerprints up, but ignores the relationship between different fingerprints. By equation 3, more related information can be considered, thus the model trained on the latter data may perform better.

2.3 Task3: Multi-step retrosynthesis planning

In this part, we introduce the Retro* algorithm proposed by Binghong Chen et al. in 2020. As a neural-based A* like algorithm, it maintains the search as an AND-OR tree, which we introduced in section 2.3.1, and builds a neural network to learn the cost of specific molecule and guide the A* like algorithm, which we introduced in section 2.3.2. Shown in Fig.2, The algorithm can be divided into three steps: Selection, Expansion and Update.

2.3.1 AND-OR Tree

Firstly, in the AND-OR tree, node in "AND" type corresponds to a specific reaction and node in "OR" type corresponds to a specific molecule. Edge only connects "AND" type node and "OR" type node to denote the relation between candidate reactions and reactant molecules, which allows us to spilt the problem into sub-problems that we discuss in Task1.

Different from existing searching algorithm like Monte Caro Search or Proof Number Search. We use AND-OR tree only to utilize the global value estimation, which served as a single player game. It conforms to the circumstance and gets lower time complexity.

Algorithm 1: Retro*(t)

Initialize $T = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} \leftarrow \{t\}, \mathcal{E} \leftarrow \emptyset$;
while *route not found* **do**
  $m_{next} \leftarrow \text{argmin}_{m \in \mathcal{F}(T)} V_t(m)$;
  $\{R_i, \mathcal{S}_i, c(R_i)\}_{i=1}^{k} \leftarrow B(m_{next})$;
  for $i \leftarrow 1$ **to** k **do**
    Add R_i to T under m_{next};
    for $j \leftarrow 1$ **to** $|\mathcal{S}_i|$ **do**
      Add $\mathcal{S}_{ij}$ to T under R_i;
  Update $V_t(m)$ for m in $\mathcal{F}(T)$;
return route;

Figure 1: Retro* algorithm pseudo code.

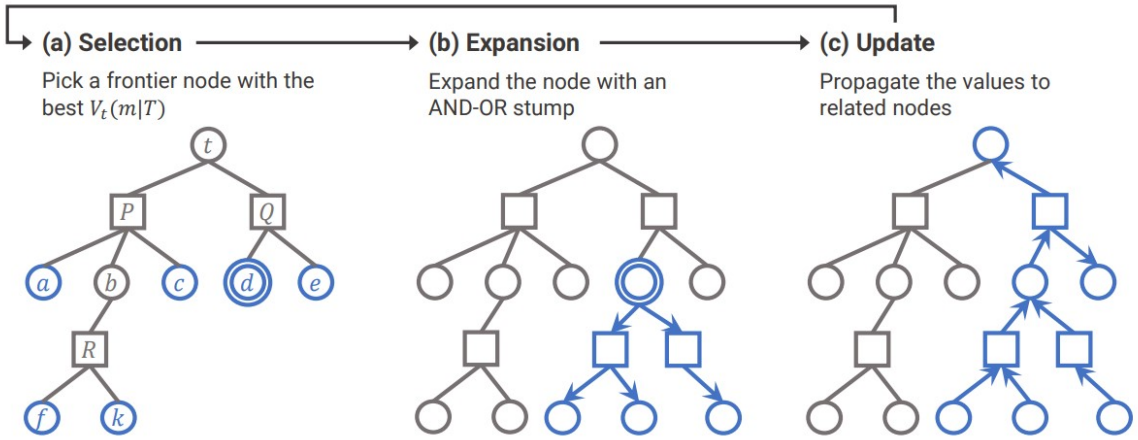

Figure 2: Retro* algorithm framework.

2.3.2 A* like algorithm and Neural Network

A* algorithm is a best-first search algorithm which uses the cost from start $g()$ together with the estimation of future cost $h()$ to select move.

Assume that we build the AND-OR tree T and the target molecule t is the root node. m are the molecule nodes("AND" nodes) space in tree T. Our goal is to minimize the cost $V_t(m|T) = g_t(m|T) + h_t(m|T)$, where $g_t(m|T)$ is the cost of current reactions that have happened in T, and $h_t(m|T)$ is the estimated cost for future reactions needed to complete such planning.

In Selection step, to calculate the value of $V_t(m|T)$, the cost to synthesize specific molecule m is needed. Sometime we need to predict the cost, where the MLP discussed in Task2 is brought in. After Selection, we Expanded the graph by adding reactions related and propagate the values to related nodes back.

3 Experiments

Our code is uploaded into Github repo. Click to download and see the result.

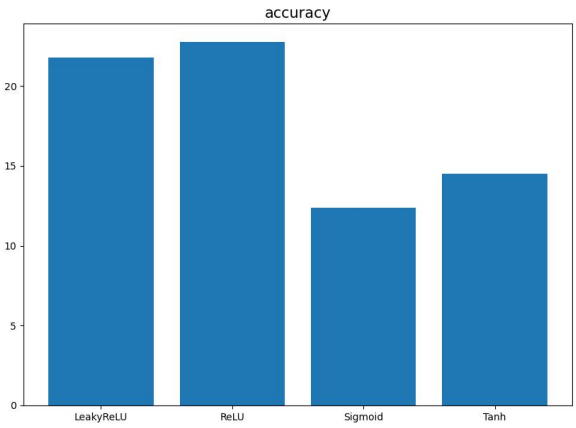

Figure 3: Results of different activation function

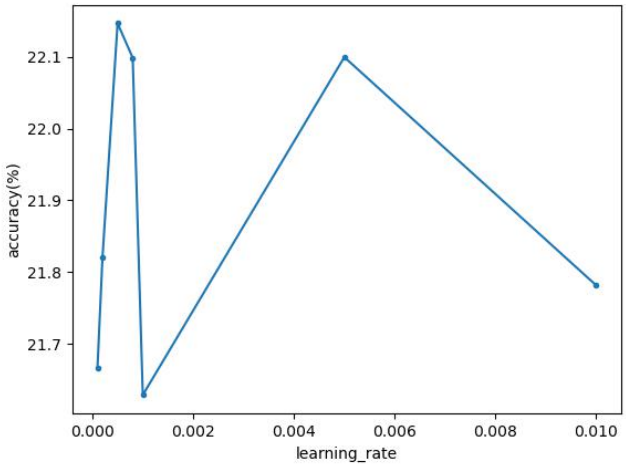

Figure 4: Results of different learning rate

3.1 Task1: Single-step retrosynthesis prediction

To begin with, we set the learning rate to 0.0005 and utilize the molecule formula as our dataset. Subsequently, we employ various activation functions and conduct experiments to evaluate the performance of the models. These experiments reveal distinct differences in performance among the models when different activation functions are applied. Listed below is our results in Figure 3. After careful evaluation, it becomes evident that when ReLU is employed as the activation function, our model demonstrates the highest level of performance among all the tested activation functions. This finding underscores the effectiveness of ReLU in capturing the underlying patterns and representations in the molecular data, ultimately leading to superior model performance.

We opted for the Rectified Linear Unit (ReLU) activation function and proceeded to meticulously tune the learning rate. The outcomes of our efforts are visually represented in Figure 4, which provides a clear illustration of the impact of different learning rates on the performance of our model.Through our analysis, we discovered that the model achieves its optimal performance when the learning rate is set to 0.005. This finding highlights the importance of fine-tuning hyperparameters, specifically the learning rate, to maximize the model's capability to learn and generalize patterns from the molecular data.

3.2 Task2: Molecule evaluation

3.2.1 Single molecule evaluation

In this part, we will test the performance for all three methods: linear regression, cosine similarity and MLP. Since there are almost 400k training data and 125k testing data, it's hard for cosine similarity method to deal with so many data. (Cosine similarity takes $O(mn)$ time for m training data and n testing data because each testing data needs to be compared to all training data). Thus we choose $training_size = 100000$ when dealing with cosine similarity. Other experimental settings are shown as below:

- Linear regression: use $sklearn.linear_model.LinearRegression$ with default settings.
- Cosine similarity: no extra settings.
- MLP: use $MSELoss$ as loss function and $Adam(lr = 1e-3)$ as optimizer. Training epochs $epoch = 10$, and batch size $batch_size = 16$

By experiments, we get the average loss and "accuracy" recorded in Table 1

Method	Loss	Accuracy
Linear regression	8.582	0.176
Cosine similarity	9.028	0.673
MLP	0.664	0.593

Table 1: Average loss and approximate accuracy for different methods.

According to the result, MLP has the best average loss, which is about 13x lower than other methods, implying that non-linear deep neural network is one of the strongest model in complex molecule evaluation task. It's also notable that though linear regression and cosine similarity have similar average loss (the gap is less than 10%), they are very different in accuracy. By analysing the loss for every molecule, maybe we can explain the reason:

Cosine similarity method directly assign the cost of training data with largest similarity to testing data, thus the cost can be very precise for some samples, especially for those with $cost = 0$ (shown in Fig 5). On the other hand, linear regression method predicts costs for testing data. Predictions fluctuate around 0, but it's hard to calculate an accurate 0 (shown in Fig 6).

That's why cosine similarity and linear regression looks so different in accuracy. However, most of the loss comes from those data with large costs (e.g. test sample 299), therefore the average loss is closer for two methods.

```
Test sample: 297, Best fit sample: 11961, Predicted cost: 0.000000, Actual cost: 0.000000, Square loss: 0.000000
Test sample: 298, Best fit sample: 41180, Predicted cost: 0.064043, Actual cost: 0.000003, Square loss: 0.004101
Test sample: 299, Best fit sample: 30057, Predicted cost: 0.000027, Actual cost: 5.349688, Square loss: 28.618878
Test sample: 300, Best fit sample: 971, Predicted cost: 0.000000, Actual cost: 0.000000, Square loss: 0.000000
Test sample: 301, Best fit sample: 23741, Predicted cost: 0.000000, Actual cost: 0.000000, Square loss: 0.000000
Test sample: 302, Best fit sample: 23742, Predicted cost: 0.000000, Actual cost: 0.000000, Square loss: 0.000000
Test sample: 303, Best fit sample: 23743, Predicted cost: 0.004552, Actual cost: 0.004552, Square loss: 0.000000
Test sample: 304, Best fit sample: 23744, Predicted cost: 2.716793, Actual cost: 2.716793, Square loss: 0.000000
Test sample: 305, Best fit sample: 84610, Predicted cost: 2.717466, Actual cost: 2.736389, Square loss: 0.000358
Test sample: 306, Best fit sample: 12811, Predicted cost: 0.000000, Actual cost: 0.000000, Square loss: 0.000000
Test sample: 307, Best fit sample: 11173, Predicted cost: 0.000000, Actual cost: 0.000000, Square loss: 0.000000
```

Figure 5: Cost and loss for cosine similarity method.

To have an intuitive comparison, we draw a scatter plot to show the distribution of testing samples for different methods which is shown in Fig 7.

Above all, we can conclude the advantages and disadvantages for linear regression, cosine similarity and MLP:

- Linear regression

```
Test sample: 297, Predicted cost: 0.051033, Actual cost: 0.000000, Square loss: 0.002604
Test sample: 298, Predicted cost: -0.192197, Actual cost: 0.000003, Square loss: 0.036941
Test sample: 299, Predicted cost: -0.202659, Actual cost: 5.349688, Square loss: 30.828563
Test sample: 300, Predicted cost: -0.007290, Actual cost: 0.000000, Square loss: 0.000053
Test sample: 301, Predicted cost: 0.680787, Actual cost: 0.000000, Square loss: 0.463472
Test sample: 302, Predicted cost: -0.019695, Actual cost: 0.000000, Square loss: 0.000388
Test sample: 303, Predicted cost: 1.147317, Actual cost: 0.004552, Square loss: 1.305913
Test sample: 304, Predicted cost: 1.338514, Actual cost: 2.716793, Square loss: 1.899652
Test sample: 305, Predicted cost: 1.668795, Actual cost: 2.736389, Square loss: 1.139757
Test sample: 306, Predicted cost: -0.036535, Actual cost: 0.000000, Square loss: 0.001335
Test sample: 307, Predicted cost: 0.467819, Actual cost: 0.000000, Square loss: 0.218855
```

Figure 6: Cost and loss for linear regression method.

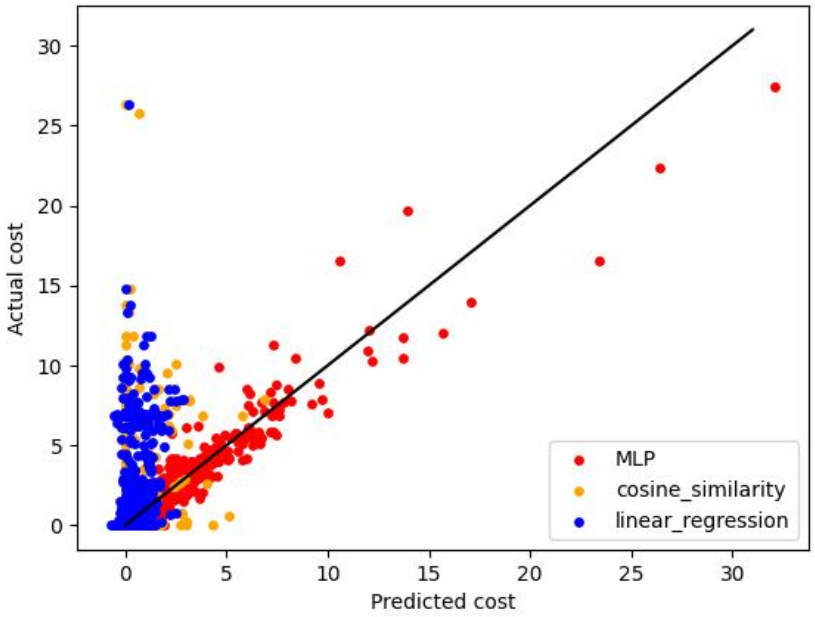

Figure 7: Testing sample distribution. Here lim_x is predicted cost and lim_y is actual cost. The black line $y = x$ represents points with $prediction\_cost = actual\_cost$. Closer the point to the line, better the prediction.

  - Pros: Easy to build; The model is simple, and can be trained in short time.
  - Cons: Not so accurate; Can't predict a very precise value, and can't predict large costs as well.

- Cosine similarity
  - Pros: Can directly predict the cost of a given molecule without training a model; Some predictions are very precise.
  - Cons: Takes too much time when the number of testing data grows; Can't predict large costs well.

- MLP
  - Pros: Most precise; Be able to deal with large scale data.
  - Cons: Takes a lot of time to train; May suffer over fitting after several epochs.

As MLP performs well in single molecule evaluation, we also apply it to solve multiple molecule evaluation task.

3.2.2 Multiple molecule evaluation

In this part, we test the performance of MLP for two different ways of calculating the total cost of several molecules.

For equation 2, the training process is similar with single molecule evaluation. Training & testing loss as well as accuracy for each epoch is shown in Fig 8

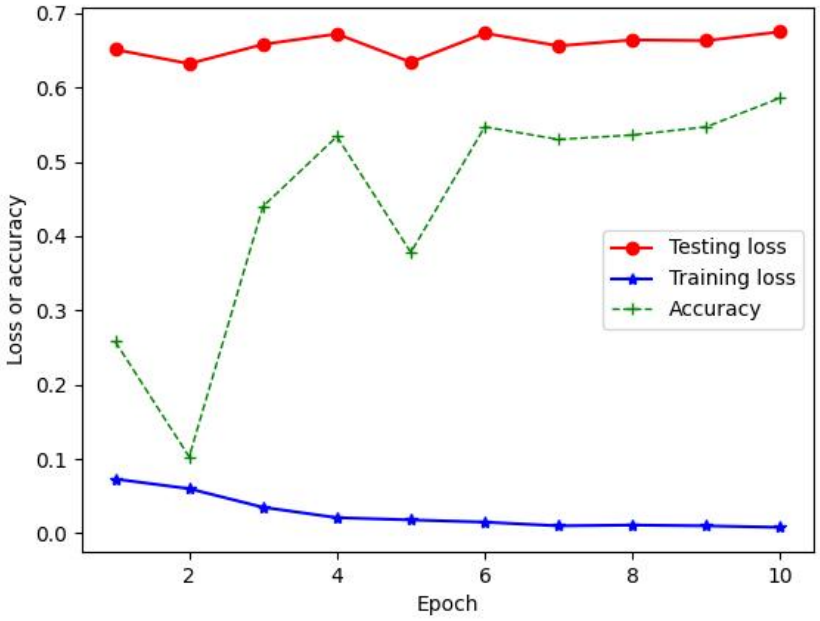

Figure 8: Loss and accuracy for training (equation 3.1).

According to the result, training loss decreases (from 0.073 to 0.008), but testing loss doesn't change a lot. However, the accuracy gradually rises, implying that the regression performance is getting better. After training, we test the multiple molecule sample on the model. For each testing sample, the cost is the sum of it's all molecules. Table 2 shows the loss for parts of testing samples.

Sample	109	110	111	112	113	114	115	116	117	118	119	120
Predicted cost	2.70	1.24	2.31	0.00	0.00	0.00	0.00	1.41	0.00	0.00	4.91	0.98
Actual cost	2.47	2.05	2.15	0.01	0.00	0.00	0.05	1.48	0.19	0.00	4.39	1.20
Square loss	0.05	0.65	0.03	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.28	0.05
Average loss	0.505											

Table 2: Predicted & actual cost and loss for some testing samples (equation 3.1).

Here average loss is for all testing samples. According to the result, MLP can predict large costs (e.g sample 119) better. And in multiple evaluation task, the error is still acceptable.

For equation 3, we pile molecules up to get another $2048 - d$ "aggregated finger", thus the origin MLP is still work. Training & testing loss as well as accuracy for each epoch is shown in Fig 9.

Also we can check detail loss for each sample:

In total, separately predicting each fingerprint has 3.84x better loss than piling them up, which is violated to our previous analysis in section 2.2.2. The reasons may be:

1. Bad function. We simply pile fingerprints up, which is just a rough linear reflection. By this method, mutual information for different molecules may not be well obtained. One considerable method is to use a graph to describe a multiple molecule input. The nodes are fingerprints of molecules, and the edges are cosine similarities of different molecules.

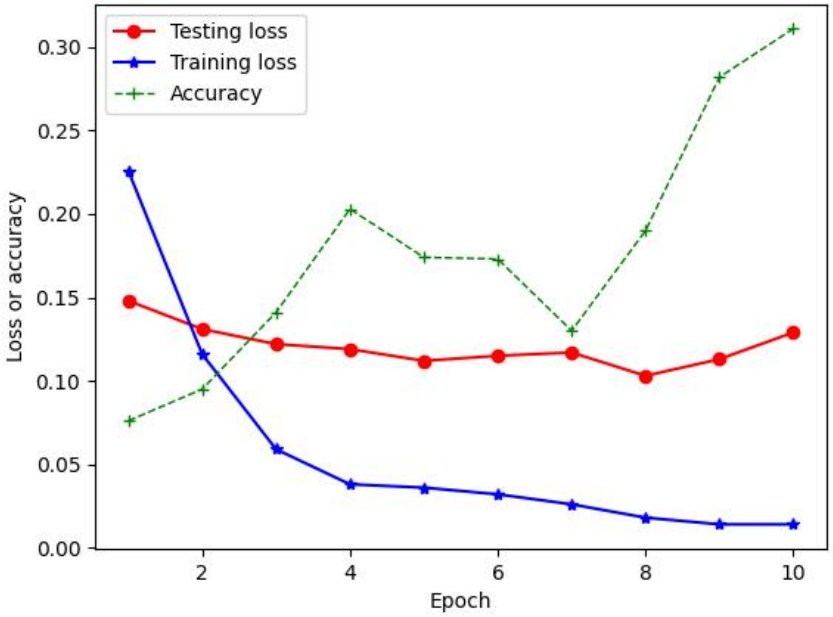

Figure 9: Loss and accuracy for training (equation 3.2).

Sample	378	379	380	381	382	383	384	385	386	387	388	389
Predicted cost	1.32	0.00	0.15	1.01	1.18	1.63	0.29	0.31	1.32	2.69	0.68	0.82
Actual cost	1.20	0.00	0.00	0.00	0.01	1.03	0.00	0.00	0.01	2.41	0.01	0.02
Square loss	0.01	0.00	0.02	1.02	1.37	0.37	0.08	0.09	1.73	0.08	0.45	0.64
Average loss	1.941											

Table 3: Predicted & actual cost and loss for some testing samples (equation 3.2).

2. Bad model selection. MLP may not be strong enough. Graph neural network (GNN) can grab more effective information among molecules, thus have a better performance.

3. Inaccurate sampling. During sampling, we just sum the costs of molecules as the labels for samples, thus the sampled cost is not so accurate. If precise data can be accessed, the training result may be better.

3.3 Task3: Multi-step retrosynthesis planning

The detailed result is updated as task3.log file. Iteration is 100 for each test data. For all 190 test molecules, there are 99 molecules that found their synthesis path successfully. However there are 12 molecules failed in Expansion part and 79 molecules didn't find a path in 100 iterations. So the success rate is $99/190 = 52.11\%$, identical with the result in the original paper. Limited by compute resource and time, there is only one experiment on this topic, maybe we can get a higher success rate when we increase the epochs according to the paper.

Finally we discuss the pros and cons of the Retro* algorithm.

- Pros

  1. The Retro* algorithm is an A* like algorithm, similar with A*, Retro can get the best result if it exists.
  2. The Retro* algorithm performs better both on efficiency(running time) and success rate compared to existing algorithm according to the paper. But we do not conduct comparative experiments.

3. The Retro* algorithm brings in the Value MLP to predict the molecules synthesis cost that doesn't occur in the dataset.

- Cons

1. The Retro* algorithm can not deal with the molecule that doesn't occur in given routes and starting molecules. Maybe we can bring in the Zero-Shot Learning method to predict possible synthesis path.
2. The Retro* algorithm needs the pre-trained model, and if we need to use the own trained model, maybe it takes lots of time.

4 Conclusion

In this project, we implement many algorithms and methods to finish all three tasks (in section 2), and collect experimental results to analyse (in section 3).

We have attempted to solve the single-step retrosynthesis prediction problem using a multi-layer perceptron (MLP) network and also experimented with the xgboost algorithm. However, due to the high dimensionality of the data, both approaches have proven ineffective in solving the problem. Therefore, we need to further explore alternative methods and strategies to overcome this challenge."

We also implement 3 different methods to solve molecule evaluation task: linear regression, cosine similarity and MLP. Using MLP, we can deal with both single and multiple molecule evaluation. We are glad to see our algorithm work on this complex task. We do a lot of experiments, and discuss the details of evaluation task. However, more methods like GNN are not applied yet, which is waiting for our further exploration.

In addition, we imply the Retro* algorithm in Task3, which combines neural network and traditional A* algorithm. Retro* is a relatively mature algorithm, which can find the minimized cost value synthesis path in fewer time, gets higher success rate meanwhile. But we need to pretrain a model or import a big pretrained model first, and without the Zero-Shot Learning method, it cannot give possible synthesis reactions for unknown products. In the future, we can bring in the ZSL to help chemists and chemical engineers find possible reactions and develop related industries.

The contribution of all three teammates is shown in Table.4. Huge thanks for teacher and TA's support.

Name	Student ID	Contribution	Work
Kewei Zhao	520021910674	50	Task2's code, experiment and report
Mabiao Long	520030910308	25	Task3's code, experiment and report
Yichuan Peng	520030910296	25	Task1's code, experiment and report

Table 4: Contribution Distribution.

References

[CLDS20] Binghong Chen, Chengtao Li, Hanjun Dai, and Le Song. Retro*: learning retrosynthetic planning with neural guided a* search. In *International Conference on Machine Learning*, pages 1608–1616. PMLR, 2020.