

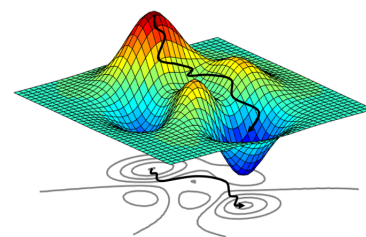
hyper-parameter optimization project report

Arash Sajjadi

Optimization in data science

Shahid Beheshti University

July 26, 2022



Keywords: Deep learning, Neural network, Hyper-parameter optimization, Bayesian optimization, Optuna search algorithm, Ray tune

ABSTRACT

The upcoming project is a hyper-parameter optimization of the neural network, which has been implemented on three datasets of Bbbp, HIV, and FreeSolv. Two datasets describe classification problems, and one dataset that I am working on is regression. Next, I will explain each data set. One of my most essential references during this project will be the ray library article. [1]

CONTENTS

Contents

1 Introduction

- 1.1 Distribution of labels
- 1.2 Type of the features

2 Preprocessing

- 2.1 Data partitioning
- 2.2 Feature selection
- 2.3 Missing values
- 2.4 Accuracy and error measurement criteria

3 Neural networks and introduction of hyper-parameters

- 3.1 Defining data ladders
- 3.2 Defining Neural Networks
- 3.3 Training loops
- 3.4 Applying the best model to the test set
- 3.5 Defining the search space
- 3.6 Main function

4 Final results

5 Conclusion

References

PRE-INTRODUCTION

The data set that I am going to work on during this project is the molecular data set. Since my focus in this project is only on optimizing the hyper-parameters of a neural network, I will refrain from additional explanations about the nature of the data and introduce them briefly.

	Dataset	Task Type	Tasks	Compunds	Category
1	Bbbp	Classification	1	2, 039	Physiology
2	HIV	Classification	1	41, 127	Biophysics
3	FreeSolv	Regression	1	642	Pysical Chemistry

1 INTRODUCTION

In this section, I would like to put a summary table of the features and labels of the data sets I am dealing with. It is necessary to explain that some features represent a constant value that I have deleted in all datasets after calling in Python. Therefore, I have specified each record's algebraic dimension of the features vector in this table. It should be mentioned that I have partitioned the data set to the training set, validation, and testing in the same proportion.

	Number of features	The algebraic dimension of the features	Number of records	Target variable	Training set Validation set Test set
Bbbp	200	185	2, 039	{0, 1}	70%-20%-10%
HIV	200	192	41, 127	{0, 1}	70%-20%-10%
FreeSolv	200	161	642	\mathbb{R}	70%-20%-10%

Also, since I'm presenting all three projects in one report, I've highlighted all the code for the **Bbbp** dataset with a **light red** background, the **HIV** dataset with a **light green** background, and the **FreeSolv** dataset with a **light blue** background for more straightforward diagnosis.

1.1 Distribution of labels

Examining the distribution of labels is one of the first things we should do when dealing with any data set. I have also plotted this distribution for all three data sets. ¹

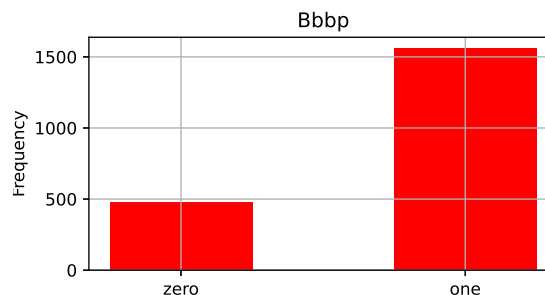


Figure 1: Distribution of Bbbp dataset labels

Since there is an imbalance between the class of one and zero labels in Bbbp's dataset (Fig 1), I will partition the data with the same proportion of distribution.²

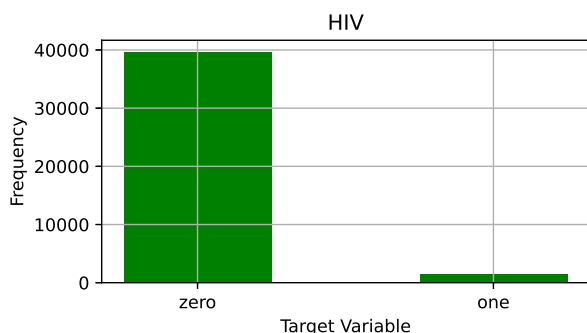


Figure 2: Distribution of HIV dataset labels

Since there is an imbalance between the class of one and zero labels in HIV's dataset (Fig 2), I will partition the data with the same proportion of distribution.

It is clear that this imbalance is significantly more apparent in the HIV dataset than in the Bbbp dataset.

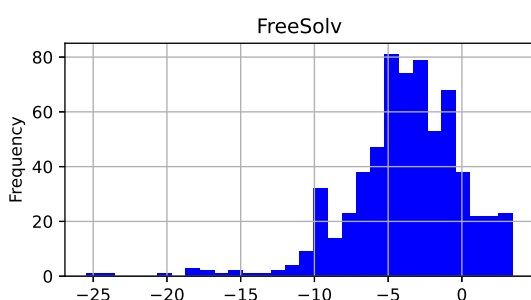


Figure 3: Distribution of FreeSolv dataset labels

The best way to interpret Figure 3 is to present a frequency table describing this graph. Therefore, I draw your attention to this table.

count	mean	std	min	25%	50%	75%	max
642	-3.803	3.847	-25.47	-5.727	-3.53	-1.215	3.43

1.2 Type of the features

Undoubtedly, all the features specify different properties of chemicals. It should be noted that all these data have been normalized by the CDF³ scaler method. According to my research, the standard and min-max scaler methods are not applicable for these types of data sets. [2] Also, I should point out that none of the features represent categorical features (or at least I have access to their numerically encoded version).

2 PREPROCESSING

Data preprocessing includes Data partitioning, Feature selection, and Missing values subsections that prepare the data for final processing.

2.1 Data partitioning

I have chosen an evaluation method for the best model for these three projects, which I will explain in detail. For a better visual understanding, I drew Figure 4 to make it easier for me to explain what is going to happen. I will use a semi-cross-validation system to train the neural network, which needs to partition the data like Figure 4.

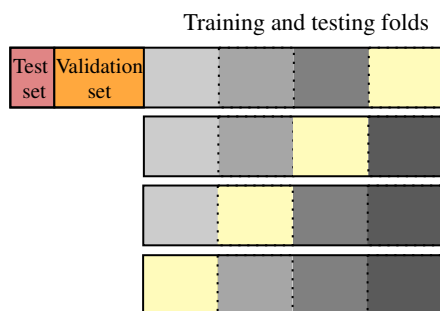


Figure 4: Data partitioning diagram in all three data sets

```
1 %%capture
2 try:
3     import ray
4 except:
5     %pip install ray
6     import ray
7
8 %pip install ray
9 try:
10     import optuna
11 except:
12     %pip install optuna
13     import optuna
14
15 try:
16     from torchmetrics import ConfusionMatrix
17 except:
18     %pip install torchmetrics
19     from torchmetrics import ConfusionMatrix
20 import numpy as np
21 import torch
22 import torch.optim as optim
23 import torch.nn as nn
24 from torchvision import datasets, transforms
25 from torch.utils.data import DataLoader,
26     TensorDataset, Dataset
27 import torch.nn.functional as F
28 from ray import tune
29 import os
30 import ray
31 from ray.tune.schedulers import ASHAScheduler
32 import warnings
33 warnings.filterwarnings('ignore')
34 import pandas as pd
35 from sklearn.model_selection import train_test_split
36 from sklearn.metrics import balanced_accuracy_score
37 import matplotlib as mpl
38 import matplotlib.pyplot as plt
39 import math
40 from sklearn.metrics import roc_auc_score
41 from sklearn.experimental import
42     enable_hist_gradient_boosting
```

```
41 from sklearn.ensemble import
42     HistGradientBoostingRegressor
43 confmat = ConfusionMatrix(num_classes=2)
44 %%capture
45 try:
46     from featurewiz import featurewiz
47 except:
48     !pip install featurewiz==0.1.70
49     from featurewiz import featurewiz
50
51 #read data
52 initial_targets=pd.read_csv("bbbp.csv")
53 initial_features=pd.read_csv("bbbp_global_cdf_rdkit.
54     csv")
55
56 initial_features=initial_features.loc[:, (
57     initial_features != initial_features.iloc[0]).any
58     ())
59
60 shuffled_targets=initial_targets.sample(frac=1,
61     random_state=1234).reset_index(drop=True)
62 shuffled_features=initial_features.sample(frac=1,
63     random_state=1234).reset_index(drop=True)
64 #shuffling the data to randomize the sequence
65
66 X_train, X_test, y_train, y_test = train_test_split(
67     shuffled_features,shuffled_targets["p_np"],
68     test_size=0.1, random_state=1234, stratify=
69     shuffled_targets["p_np"])
70
71 X_train, X_valid, y_train, y_valid = train_test_split(
72     X_train,y_train,test_size=0.22, random_state=1234,
73     stratify=y_train)
74
75 X_train12 ,X_train34 , y_train12, y_train34=
76     train_test_split(X_train,y_train,test_size=0.5,
77     random_state=1234,stratify=y_train)
78
79 X_train1 ,X_train2 , y_train1, y_train2=
80     train_test_split(X_train12,y_train12,test_size
81     =0.5, random_state=1234,stratify=y_train12)
82
83 X_train3 ,X_train4 , y_train3, y_train4=
84     train_test_split(X_train34,y_train34,test_size
85     =0.5, random_state=1234,stratify=y_train34)
86
87 del X_train12
88 del X_train34
89 del y_train12
90 del y_train34
91
92 k_fold_X_train=[X_train1 ,X_train2,X_train3 ,X_train4
93     ]
94
95 k_fold_y_train=[y_train1 ,y_train2,y_train3 ,y_train4
96     ]
97
98 del X_train1
99 del X_train2
100 del X_train3
101 del X_train4
102 del y_train1
103 del y_train2
104 del y_train3
105 del y_train4
106
107 Data=pd.concat([X_train, y_train], axis=1) #reshape
108 data for featurewiz feature selection
109 target = ['p_np']
110
111 from featurewiz import featurewiz
112 feature_selection = featurewiz(Data, target,
113     corr_limit=0.97, verbose=2,header=0, nrows=None,
114     outputs="features")
115
116 X_train=X_train[feature_selection[0]]
117 X_valid=X_valid[feature_selection[0]]
118 X_test=X_test[feature_selection[0]]
119 y_train=pd.DataFrame(data=y_train)
120 y_valid=pd.DataFrame(data=y_valid)
```

```

91 y_test=pd.DataFrame(data=y_test)
92
93 for i in np.arange(0,4):
94     k_fold_X_train[i]=k_fold_X_train[i][
95         feature_selection[0]]
96     k_fold_y_train[i]=pd.DataFrame(data=k_fold_y_train[i
97         ])

```

Listing 1: Import libraries, data reading, train test splitting, and feature selection

```

1 import ray
2 import optuna
3 from torchmetrics import ConfusionMatrix
4 import numpy as np
5 import torch
6 import torch.optim as optim
7 import torch.nn as nn
8 from torchvision import datasets, transforms
9 from torch.utils.data import DataLoader,
10     TensorDataset, Dataset
11 import torch.nn.functional as F
12 from ray import tune
13 import os
14 from ray.tune.schedulers import ASHAScheduler
15 import warnings
16 warnings.filterwarnings('ignore')
17 import pandas as pd
18 from sklearn.model_selection import train_test_split
19 from sklearn.metrics import balanced_accuracy_score
20 import matplotlib as mpl
21 import matplotlib.pyplot as plt
22 import math
23 from sklearn.metrics import roc_auc_score
24 from sklearn.experimental import
25     enable_hist_gradient_boosting
26 from sklearn.ensemble import
27     HistGradientBoostingRegressor
28 confmat = ConfusionMatrix(num_classes=2)
29 from featurewiz import featurewiz
30 from sklearn.impute import KNNImputer
31 #read data
32 initial_targets=pd.read_csv("Downloads/Hiv dataset/hiv
33     .csv")
34 initial_features=pd.read_csv("Downloads/Hiv dataset/
35     hiv_global_cdf_rdkit.csv")
36
37 initial_features=initial_features.loc[:, (
38     initial_features != initial_features.iloc[0]).any
39     ())
40
41 shuffled_targets=initial_targets.sample(frac=1,
42     random_state=1234).reset_index(drop=True).drop("
43     smiles",axis=1)
44 shuffled_features=initial_features.sample(frac=1,
45     random_state=1234).reset_index(drop=True)
46 X_train, X_test, y_train, y_test = train_test_split(
47     shuffled_features,shuffled_targets,test_size=0.1,
48     random_state=1234,stratify=shuffled_targets["
49     HIV_active"])
50 X_train, X_valid, y_train, y_valid = train_test_split(
51     X_train,y_train,test_size=0.22, random_state=1234,
52     stratify=y_train["HIV_active"])
53
54 #feature selection
55 Data=pd.concat([X_train, y_train], axis=1)
56 target = ['HIV_active']
57 feature_selection = featurewiz(Data.dropna(), target,
58     corr_limit=0.8, verbose=2,header=0, nrows=None,
59     outputs="Features")

```

```

45 #dealing with missings
46 X_train=X_train[feature_selection[0]]
47 X_valid=X_valid[feature_selection[0]]
48 X_test=X_test[feature_selection[0]]
49 y_train=pd.DataFrame(data=y_train)
50 y_valid=pd.DataFrame(data=y_valid)
51 y_test=pd.DataFrame(data=y_test)
52
53
54 imputer = KNNImputer(n_neighbors=3)
55 X_train=pd.DataFrame(data=imputer.fit_transform(
56     X_train))
57 X_valid=pd.DataFrame(data=imputer.transform(X_valid))
58 X_test=pd.DataFrame(data=imputer.transform(X_test))
59
60 X_train12,X_train34 , y_train12, y_train34=
61     train_test_split(X_train,y_train,test_size=0.5,
62         random_state=1234,stratify=y_train["HIV_active"])
63 X_train1,X_train2 , y_train1, y_train2=
64     train_test_split(X_train12,y_train12,test_size
65         =0.5, random_state=1234,stratify=y_train12["
66         HIV_active"])
67 X_train3,X_train4 , y_train3, y_train4=
68     train_test_split(X_train34,y_train34,test_size
69         =0.5, random_state=1234,stratify=y_train34["
70         HIV_active"])
71 del X_train12
72 del X_train34
73 del y_train12
74 del y_train34
75 k_fold_X_train=[X_train1,X_train2,X_train3,X_train4
76     ]
77 k_fold_y_train=[y_train1,y_train2,y_train3,y_train4
78     ]
79 del X_train1
80 del X_train2
81 del X_train3
82 del X_train4
83 del y_train1
84 del y_train2
85 del y_train3
86 del y_train4
87
88 k_fold_y_train[0]=pd.DataFrame(data=k_fold_y_train[0])
89 k_fold_y_train[1]=pd.DataFrame(data=k_fold_y_train[1])
90 k_fold_y_train[2]=pd.DataFrame(data=k_fold_y_train[2])
91 k_fold_y_train[3]=pd.DataFrame(data=k_fold_y_train[3])

```

Listing 2: Import libraries, data reading, train test splitting, feature selection, and deal with missing data

```

1 %%capture
2 try:
3     import ray
4 except:
5     %pip install ray
6     import ray
7
8
9 %pip install ray
10 try:
11     import optuna
12 except:
13     %pip install optuna
14     import optuna
15
16 import numpy as np
17 import torch
18 import torch.optim as optim
19 import torch.nn as nn
20 from torchvision import datasets, transforms

```

```

21 from torch.utils.data import DataLoader,
    TensorDataset, Dataset
22 import torch.nn.functional as F
23 from ray import tune
24 import os
25 import ray
26 from ray.tune.schedulers import ASHAScheduler
27 import warnings
28 warnings.filterwarnings('ignore')
29 import pandas as pd
30 from sklearn.model_selection import train_test_split
31 import matplotlib as mpl
32 import matplotlib.pyplot as plt
33 import math
34 from sklearn.experimental import
    enable_hist_gradient_boosting
35 from sklearn.ensemble import
    HistGradientBoostingRegressor
36 try:
37     from featurewiz import featurewiz
38 except:
39     !pip install featurewiz==0.1.70
40     from featurewiz import featurewiz
41 #read data
42 initial_targets=pd.read_csv("freesolv.csv")
43 initial_features=pd.read_csv("sampl(freesolv)
    _global_cdf_rdkit.csv")
44
45 initial_features=initial_features.loc[:, (
    initial_features != initial_features.iloc[0]).any
    ()]
46
47 shuffled_targets=initial_targets.sample(frac=1,
    random_state=1234).reset_index(drop=True)
48 shuffled_features=initial_features.sample(frac=1,
    random_state=1234).reset_index(drop=True)
49 #shuffling the data to randomize the sequence
50
51 X_train, X_test, y_train, y_test = train_test_split(
    shuffled_features,shuffled_targets["freesolv"],
    test_size=0.1, random_state=1234)
52 X_train, X_valid, y_train, y_valid = train_test_split(
    X_train,y_train,test_size=0.22, random_state=1234)
53 X_train12,X_train34 , y_train12, y_train34=
    train_test_split(X_train,y_train,test_size=0.5,
    random_state=1234)
54 X_train1, X_train2, y_train1, y_train2=
    train_test_split(X_train12,y_train12,test_size
    =0.5, random_state=1234)
55 X_train3, X_train4, y_train3, y_train4=
    train_test_split(X_train34,y_train34,test_size
    =0.5, random_state=1234)
56 del X_train12
57 del X_train34
58 del y_train12
59 del y_train34
60 k_fold_X_train=[X_train1 ,X_train2,X_train3 ,X_train4
    ]
61 k_fold_y_train=[y_train1 ,y_train2,y_train3 ,y_train4
    ]
62 del X_train1
63 del X_train2
64 del X_train3
65 del X_train4
66 del y_train1
67 del y_train2
68 del y_train3
69 del y_train4
70
71 Data=pd.concat([X_train, y_train], axis=1) #reshape
    data for featurewiz feature selection
72 target = ["freesolv"]

```

```

73 from featurewiz import featurewiz
74 feature_selection = featurewiz(Data, target,
    corr_limit=0.97, verbose=2,header=0, nrows=None,
    outputs="features")
75
76 X_train=X_train[feature_selection[0]]
77 X_valid=X_valid[feature_selection[0]]
78 X_test=X_test[feature_selection[0]]
79 y_train=pd.DataFrame(data=y_train)
80 y_valid=pd.DataFrame(data=y_valid)
81 y_test=pd.DataFrame(data=y_test)
82
83 for i in np.arange(0,4):
84     k_fold_X_train[i]=k_fold_X_train[i][
        feature_selection[0]]
85     k_fold_y_train[i]=pd.DataFrame(data=k_fold_y_train[i
    ])

```

Listing 3: Import libraries, data reading, train test splitting, and feature selection

In the continuation of the project, I will explain how to use our semi-cross-validation on the partitioned data.

2.2 Feature selection

The codes related to this section are also available in the previous listings. I have used the featurewiz library to select important features, which uses very up-to-date methods for feature selection. One of the main methods that featurewiz uses is exploiting the XGBoost function from the sklearn library.[3]

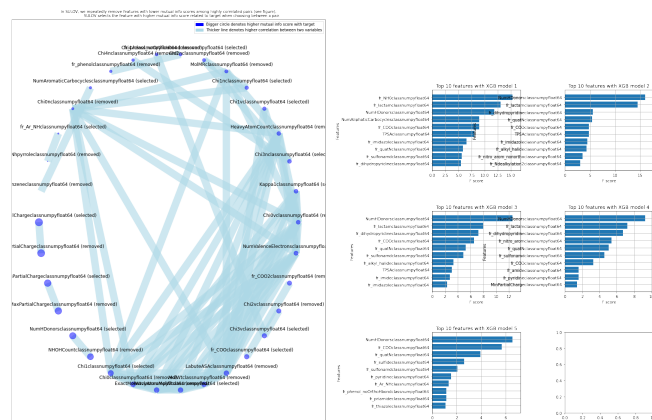


Figure 5: featurewiz output on Bbbp dataset

Out of **185** features, 20 features were removed due to being highly correlated with other features. Therefore, the remaining 165 features were processed by recursive feature selection XGBoost, and finally, **71** features were selected as important and influencing features on the target variable in the Bbbp dataset. (Fig.5)

The processing of this feature selection took 56 seconds on a Google colab server with two 2.30GHz core processors.

Out of **192** features, 51 features were removed due to being highly correlated with other features. Therefore, the remaining 141 features were processed by recursive feature selection XGBoost, and finally, **65** features were selected as important and influencing features on the target variable in the HIV dataset. (Fig.6)

The processing of this feature selection took 540 seconds on a private Ubuntu server with 24- 3.30GHz core processors.

Out of **161** features, 17 features were removed due to being highly correlated with other features. Therefore, the remaining 144 features

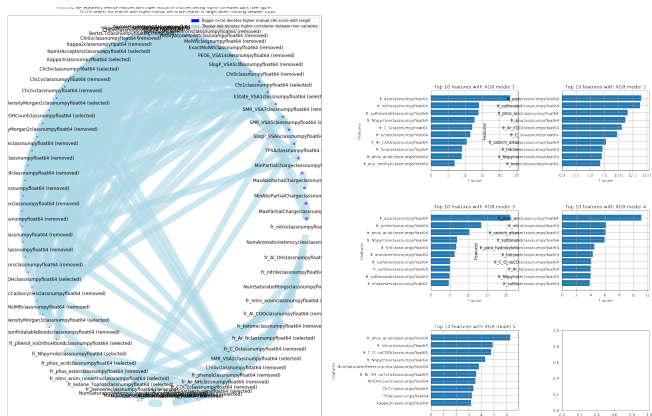


Figure 6: featurewiz output on HIV dataset

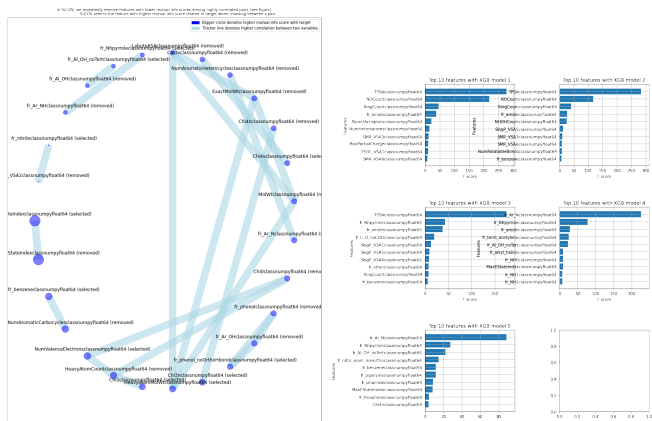


Figure 7: featurewiz output on FreeSolv dataset

were processed by recursive feature selection XGBoost, and finally, **65** features were selected as important and influencing features on the target variable in the FreeSolv dataset. (Fig.7)

The processing of this feature selection took 20 seconds on a Google colab server with two 2.30GHz core processors.

It should be noted that the feature selection section is processed **only** based on the information in the training set.

2.3 Missing values

Among these three data sets, the only one data set that has missing values. It is the HIV data set. I used knn imputer to deal with this problem. This method helps me use a predictive system to guess missing values. In fact, instead of using indicators such as median and mean, we estimate the amount of missing data more intelligently according to k close data.

Due to the large amount of data, this process was a bit time-consuming. But the result was ultimately satisfactory. Note that this step is also applied to the training, validation, and test sets based on the training set.

2.4 Accuracy and error measurement criteria

I use ROC⁴AUC⁵ score [4] for two classification projects (Bbbp,HIV) and the MSE⁶ loss meter for the regression project.

```
1 from sklearn.metrics import roc_auc_score
2
3 def compute_score(model, data_loader, device="cpu"):
4     model.eval()
5     metric = roc_auc_score
6     with torch.no_grad():
```

```
7         prediction_all= torch.empty(0, device=device)
8         labels_all= torch.empty(0, device=device)
9         for i, (feats, labels) in enumerate(
10             data_loader):
11             feats=feats.to(device)
12             labels=labels.to(device)
13             prediction = model(feats).to(device)
14             prediction = torch.sigmoid(prediction).to(
15                 device)
16             prediction_all = torch.cat((prediction_all
17                 , prediction), 0)
18             labels_all = torch.cat((labels_all, labels
19                 ), 0)
20         try:
21             t = metric(labels_all.int().cpu(),
22                 prediction_all.cpu()).item()
23         except ValueError:
24             t = 0
25         return t
```

Listing 4: ROC_AUC score

```
1 from sklearn.metrics import roc_auc_score
2
3 def compute_score(model, data_loader, device="cpu"):
4     model.eval()
5     metric = roc_auc_score
6     with torch.no_grad():
7         prediction_all= torch.empty(0, device=device)
8         labels_all= torch.empty(0, device=device)
9         for i, (feats, labels) in enumerate(
10             data_loader):
11             feats=feats.to(device)
12             labels=labels.to(device)
13             prediction = model(feats).to(device)
14             prediction = torch.sigmoid(prediction).to(
15                 device)
16             prediction_all = torch.cat((prediction_all
17                 , prediction), 0)
18             labels_all = torch.cat((labels_all, labels
19                 ), 0)
20         try:
21             t = metric(labels_all.int().cpu(),
22                 prediction_all.cpu()).item()
23         except ValueError:
24             t = 0
25         return t
```

Listing 5: ROC_AUC score

```
1 from sklearn.metrics import mean_squared_error
2
3 def compute_loss(model, data_loader, device="cpu"):
4     model.eval()
5     metric = mean_squared_error
6     with torch.no_grad():
7         prediction_all= torch.empty(0, device=device)
8         labels_all= torch.empty(0, device=device)
9         for i, (feats, labels) in enumerate(
10             data_loader):
11             feats=feats.to(device)
12             labels=labels.to(device)
13             prediction = model(feats).to(device)
14             prediction_all = torch.cat((prediction_all
15                 , prediction), 0)
16             labels_all = torch.cat((labels_all, labels
17                 ), 0)
18         t = metric(labels_all.int().cpu()
19             prediction_all.cpu()).item()
20         return t
```

Listing 6: MSE loss

3 NEURAL NETWORKS AND INTRODUCTION OF HYPER-PARAMETERS

In the optimization process of hyper-parameters, I do not decide to generally consider the number of nodes of each hidden layer as a hyper-parameter. Since I intuitively feel that the number of nodes should gradually decrease, I define the general shape of the neural network and add some nodes to each hidden layer at each step. These numbers can be defined as hyper-parameters. The figure below can give me a general idea of what I am looking for.

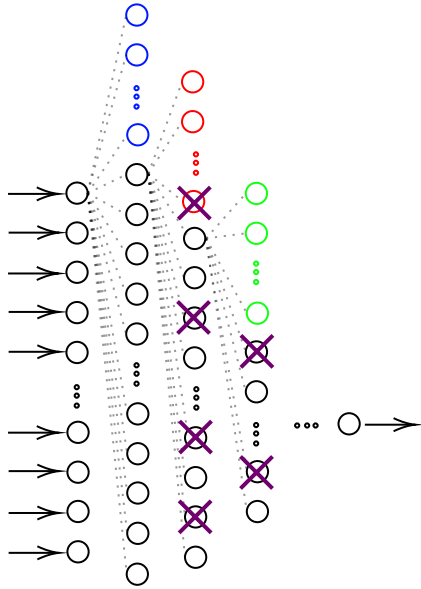


Figure 8: The general structure of the neural networks of this project

3.1 Defining data ladders

At first, according to what was said in the previous part and concerning figure 4, I will define different data loaders for different goals. Note that I separated the data during the last section completely. Here these data loaders help me to use them in the PyTorch[5] environment.

```

1 class BbbpDataset_train (Dataset):
2     def __init__(self,transform=None):
3         #data loading
4         self.x=X_train.to_numpy().astype("float32")
5         self.y=y_train.to_numpy().astype("float32")
6         self.n_samples=X_train.shape[0]
7         self.transform=transform
8     def __getitem__(self, index):
9         sample=self.x[index], self.y[index]
10        if self.transform:
11            sample=self.transform(sample)
12        return sample
13    def __len__(self):
14        return self.n_samples
15
16 class BbbpDataset_train (Dataset):
17     def __init__(self,transform=None):
18         #data loading
19         self.x=X_train.to_numpy().astype("float32")
20         self.y=y_train.to_numpy().astype("float32")
21         self.n_samples=X_train.shape[0]
22         self.transform=transform
23     def __getitem__(self, index):
24         sample=self.x[index], self.y[index]
25         if self.transform:

```

```

26         sample=self.transform(sample)
27         return sample
28     def __len__(self):
29         return self.n_samples
30
31 class kf_BbbpDataset_train (Dataset):
32     def __init__(self,k,transform=None):
33         #data loading np.delete(np.array([0,1,2,3]),k)
34         self.x=( k_fold_X_train[ np.delete(np.array(
35             ([0,1,2,3]),k)[0] ).append(k_fold_X_train[ np.
36             delete(np.array([0,1,2,3]),k)[1] ] ).append(
37             k_fold_X_train[ np.delete(np.array([0,1,2,3]),k)
38             [2] ) ] ).to_numpy().astype("float32")
39         self.y=( k_fold_y_train[ np.delete(np.array(
40             ([0,1,2,3]),k)[0] ).append(k_fold_y_train[ np.
41             delete(np.array([0,1,2,3]),k)[1] ] ).append(
42             k_fold_y_train[ np.delete(np.array([0,1,2,3]),k)
43             [2] ) ] ).to_numpy().astype("float32")
44         self.n_samples=(k_fold_X_train[ np.delete(np.
45             array([0,1,2,3]),k)[0] ].append(k_fold_X_train[
46             np.delete(np.array([0,1,2,3]),k)[1] ] ).append(
47             k_fold_X_train[ np.delete(np.array([0,1,2,3]),k)
48             [2] ])).shape[0]
49         self.transform=transform
50     def __getitem__(self, index):
51         sample=self.x[index], self.y[index]
52         if self.transform:
53             sample=self.transform(sample)
54         return sample
55     def __len__(self):
56         return self.n_samples
57
58 class kf_BbbpDataset_valid (Dataset):
59     def __init__(self,k,transform=None):
60         #data loading
61         self.x=(k_fold_X_train[k]).to_numpy().astype("
62         float32")
63         self.y=(k_fold_y_train[k]).to_numpy().astype("
64         float32")
65         self.n_samples=(k_fold_X_train[k]).shape[0]
66         self.transform=transform
67     def __getitem__(self, index):
68         sample=self.x[index], self.y[index]
69         if self.transform:
70             sample=self.transform(sample)
71         return sample
72     def __len__(self):
73         return self.n_samples
74
75 class BbbpDataset_valid (Dataset):
76     def __init__(self,transform=None):
77         #data loading
78         self.x=X_valid.to_numpy().astype("float32")
79         self.y=y_valid.to_numpy().astype("float32")
80         self.n_samples=X_valid.shape[0]
81         self.transform=transform
82     def __getitem__(self, index):
83         sample=self.x[index], self.y[index]
84         if self.transform:
85             sample=self.transform(sample)
86         return sample
87     def __len__(self):
88         return self.n_samples
89
90 class BbbpDataset_test (Dataset):
91     def __init__(self,transform=None):
92         #data loading
93         self.x=X_test.to_numpy().astype("float32")
94         self.y=y_test.to_numpy().astype("float32")
95         self.n_samples=X_test.shape[0]
96         self.transform=transform
97     def __getitem__(self, index):

```

```

84     sample=self.x[index], self.y[index]
85     if self.transform:
86         sample=self.transform(sample)
87     return sample
88     def __len__(self):
89         return self.n_samples
90
91 class ToTensor():
92     def __call__(self,sample):
93         inputs,targets=sample
94         inputs=torch.from_numpy(inputs.astype("float32
95     "))
96         targets=torch.tensor(targets.astype("float32")
97     )
98         #targets=targets.view(targets.shape[0],1)
99         return inputs,targets
100 #training set --
101 training_set = BbbpDataset_train(transform=ToTensor())
102 train_loader = DataLoader(dataset=training_set,
103                           batch_size=64,
104                           shuffle=True)
105 dataiter_train = iter(train_loader)
106 data_train = dataiter_train.next()
107
108
109 def trainloader(config):
110     return DataLoader(dataset=training_set,
111                      batch_size=config["
112                          batch_size"],
113                      shuffle=True,num_workers=2)
114 #kf-training set --
115 def kf_trainloader(config,k):
116     return DataLoader(dataset=kf_BbbpDataset_train(k,
117                      transform=ToTensor()),
118                      batch_size=config["
119                          batch_size"],
120                      shuffle=True,num_workers=2)
121 #kf-validation set --
122 def kf_validloader(config,k):
123     return DataLoader(dataset=kf_BbbpDataset_valid(k,
124                      transform=ToTensor()),
125                      batch_size=config["
126                          batch_size"],
127                      shuffle=True,num_workers=2)
128 #validation set --
129 validation_set = BbbpDataset_valid(transform=ToTensor
130 ())
131 valid_loader = DataLoader(dataset=validation_set,
132                           batch_size=64,
133                           shuffle=True)
134 dataiter_valid = iter(valid_loader)
135 data_valid = dataiter_valid.next()
136
137 def validloader(config):
138     return DataLoader(dataset=validation_set,
139                      batch_size=config["
140                          batch_size"],
141                      shuffle=True,num_workers=2)
142 #test set --
143 test_set = BbbpDataset_test(transform=ToTensor())
144 test_loader = DataLoader(dataset=test_set,
145                           batch_size=8,
146                           shuffle=False)
147 dataiter_test = iter(test_loader)
148 data_test = dataiter_test.next()
149
150 def testloader(config):

```

```

147     return DataLoader(dataset=test_set,
148                      batch_size=4,
149                      shuffle=False,num_workers=2)

```

Listing 7: Defining data loaders

```

1 class HIVDataset_train (Dataset):
2     def __init__(self,transform=None):
3         #data loading
4         self.x=X_train.to_numpy().astype("float32")
5         self.y=y_train.to_numpy().astype("float32")
6         self.n_samples=X_train.shape[0]
7         self.transform=transform
8     def __getitem__(self, index):
9         sample=self.x[index], self.y[index]
10        if self.transform:
11            sample=self.transform(sample)
12        return sample
13    def __len__(self):
14        return self.n_samples
15
16 class HIVDataset_train (Dataset):
17     def __init__(self,transform=None):
18        #data loading
19        self.x=X_train.to_numpy().astype("float32")
20        self.y=y_train.to_numpy().astype("float32")
21        self.n_samples=X_train.shape[0]
22        self.transform=transform
23    def __getitem__(self, index):
24        sample=self.x[index], self.y[index]
25        if self.transform:
26            sample=self.transform(sample)
27        return sample
28    def __len__(self):
29        return self.n_samples
30
31 class kf_HIVDataset_train (Dataset):
32     def __init__(self,k,transform=None):
33        #data loading np.delete(np.array([0,1,2,3]),k)
34        self.x=( k_fold_X_train[ np.delete(np.array
35        ([0,1,2,3]),k)[0] ].append(k_fold_X_train[ np.
36        delete(np.array([0,1,2,3]),k)[1] ] ).append(
37        k_fold_X_train[ np.delete(np.array([0,1,2,3]),k)
38        [2] ] ) ).to_numpy().astype("float32")
39        self.y=( k_fold_y_train[ np.delete(np.array
40        ([0,1,2,3]),k)[0] ].append(k_fold_y_train[ np.
41        delete(np.array([0,1,2,3]),k)[1] ] ).append(
42        k_fold_y_train[ np.delete(np.array([0,1,2,3]),k)
43        [2] ] ) ).to_numpy().astype("float32")
44        self.n_samples=(k_fold_X_train[ np.delete(np.
45        array([0,1,2,3]),k)[0] ].append(k_fold_X_train[
46        np.delete(np.array([0,1,2,3]),k)[1] ] ).append(
47        k_fold_X_train[ np.delete(np.array([0,1,2,3]),k)
48        [2] ] ) ).shape[0]
49        self.transform=transform
50    def __getitem__(self, index):
51        sample=self.x[index], self.y[index]
52        if self.transform:
53            sample=self.transform(sample)
54        return sample
55    def __len__(self):
56        return self.n_samples
57
58 class kf_HIVDataset_valid (Dataset):
59     def __init__(self,k,transform=None):
60        #data loading
61        self.x=(k_fold_X_train[k]).to_numpy().astype("
62        float32")
63        self.y=(k_fold_y_train[k]).to_numpy().astype("
64        float32")
65        self.n_samples=(k_fold_X_train[k]).shape[0]
66        self.transform=transform

```



```

53 def __getitem__(self, index):
54     sample=self.x[index], self.y[index]
55     if self.transform:
56         sample=self.transform(sample)
57     return sample
58 def __len__(self):
59     return self.n_samples
60
61 class HIVDataset_valid (Dataset):
62     def __init__(self,transform=None):
63         #data loading
64         self.x=X_valid.to_numpy().astype("float32")
65         self.y=y_valid.to_numpy().astype("float32")
66         self.n_samples=X_valid.shape[0]
67         self.transform=transform
68     def __getitem__(self, index):
69         sample=self.x[index], self.y[index]
70         if self.transform:
71             sample=self.transform(sample)
72         return sample
73     def __len__(self):
74         return self.n_samples
75
76 class HIVDataset_test (Dataset):
77     def __init__(self,transform=None):
78         #data loading
79         self.x=X_test.to_numpy().astype("float32")
80         self.y=y_test.to_numpy().astype("float32")
81         self.n_samples=X_test.shape[0]
82         self.transform=transform
83     def __getitem__(self, index):
84         sample=self.x[index], self.y[index]
85         if self.transform:
86             sample=self.transform(sample)
87         return sample
88     def __len__(self):
89         return self.n_samples
90
91 class ToTensor():
92     def __call__(self,sample):
93         inputs,targets=sample
94         inputs=torch.from_numpy(inputs.astype("float32"))
95         targets=torch.tensor(targets.astype("float32"))
96         #targets=targets.view(targets.shape[0],1)
97         return inputs,targets
98
99 #training set --
100 training_set = HIVDataset_train(transform=ToTensor())
101 train_loader = DataLoader(dataset=training_set,
102                             batch_size=64,
103                             shuffle=True)
104
105 dataiter_train = iter(train_loader)
106 data_train = dataiter_train.next()
107
108
109 def trainloader(config):
110     return DataLoader(dataset=training_set,
111                       batch_size=config["
112 batch_size"],
113                       shuffle=True,num_workers=2)
114
115 #kf-training set --
116 def kf_trainloader(config,k):
117     return DataLoader(dataset=kf_HIVDataset_train(k,
118 transform=ToTensor()),
119                       batch_size=config["
120 batch_size"],
121                       shuffle=True,num_workers=2)
122
123 #kf-training set --
124
125 def kf_validloader(config,k):
126     return DataLoader(dataset=kf_HIVDataset_valid(k,
127 transform=ToTensor()),
128                       batch_size=config["
129 batch_size"],
130                       shuffle=True,num_workers=2)
131
132 #validation set --
133 validation_set = HIVDataset_valid(transform=ToTensor())
134 valid_loader = DataLoader(dataset=validation_set,
135                           batch_size=64,
136                           shuffle=True)
137
138 dataiter_valid = iter(valid_loader)
139 data_valid = dataiter_valid.next()
140
141
142 def validloader(config):
143     return DataLoader(dataset=validation_set,
144                       batch_size=config["
145 batch_size"],
146                       shuffle=True,num_workers=2)
147
148 #test set --
149 test_set = HIVDataset_test(transform=ToTensor())
150 test_loader = DataLoader(dataset=test_set,
151                           batch_size=8,
152                           shuffle=False)
153
154 dataiter_test = iter(test_loader)
155 data_test = dataiter_test.next()
156
157
158 def testloader(config):
159     return DataLoader(dataset=test_set,
160                       batch_size=4,
161                       shuffle=False,num_workers=2)

```

Listing 8: Defining data ladders

```

1 class freesolvDataset_train (Dataset):
2     def __init__(self,transform=None):
3         #data loading
4         self.x=X_train.to_numpy().astype("float32")
5         self.y=y_train.to_numpy().astype("float32")
6         self.n_samples=X_train.shape[0]
7         self.transform=transform
8     def __getitem__(self, index):
9         sample=self.x[index], self.y[index]
10        if self.transform:
11            sample=self.transform(sample)
12        return sample
13    def __len__(self):
14        return self.n_samples
15
16 class freesolvDataset_train (Dataset):
17     def __init__(self,transform=None):
18         #data loading
19         self.x=X_train.to_numpy().astype("float32")
20         self.y=y_train.to_numpy().astype("float32")
21         self.n_samples=X_train.shape[0]
22         self.transform=transform
23     def __getitem__(self, index):
24         sample=self.x[index], self.y[index]
25         if self.transform:
26             sample=self.transform(sample)
27         return sample
28     def __len__(self):
29         return self.n_samples
30
31 class kf_freesolvDataset_train (Dataset):
32     def __init__(self,k,transform=None):
33         #data loading np.delete(np.array([0,1,2,3]),k)

```

```

34     self.x=( k_fold_X_train[ np.delete(np.array
35 ([0,1,2,3]),k)[0] ].append(k_fold_X_train[ np.
36 delete(np.array([0,1,2,3]),k)[1] ] ).append(
37 k_fold_X_train[ np.delete(np.array([0,1,2,3]),k)
38 [2] ] ) ).to_numpy().astype("float32")
39     self.y=( k_fold_y_train[ np.delete(np.array
40 ([0,1,2,3]),k)[0] ].append(k_fold_y_train[ np.
41 delete(np.array([0,1,2,3]),k)[1] ] ).append(
42 k_fold_y_train[ np.delete(np.array([0,1,2,3]),k)
43 [2] ] ) ).to_numpy().astype("float32")
44     self.n_samples=(k_fold_X_train[ np.delete(np.
45 array([0,1,2,3]),k)[0] ].append(k_fold_X_train[
46 np.delete(np.array([0,1,2,3]),k)[1] ] ).append(
47 k_fold_X_train[ np.delete(np.array([0,1,2,3]),k)
48 [2] ] ) ).shape[0]
49     self.transform=transform
50 def __getitem__(self, index):
51     sample=self.x[index], self.y[index]
52     if self.transform:
53         sample=self.transform(sample)
54     return sample
55 def __len__(self):
56     return self.n_samples
57
58 class kf_freesolvDataset_valid (Dataset):
59     def __init__(self,k,transform=None):
60         #data loading
61         self.x=(k_fold_X_train[k]).to_numpy().astype("
62 float32")
63         self.y=(k_fold_y_train[k]).to_numpy().astype("
64 float32")
65         self.n_samples=(k_fold_X_train[k]).shape[0]
66         self.transform=transform
67     def __getitem__(self, index):
68         sample=self.x[index], self.y[index]
69         if self.transform:
70             sample=self.transform(sample)
71         return sample
72     def __len__(self):
73         return self.n_samples
74
75 class freesolvDataset_valid (Dataset):
76     def __init__(self,transform=None):
77         #data loading
78         self.x=X_valid.to_numpy().astype("float32")
79         self.y=y_valid.to_numpy().astype("float32")
80         self.n_samples=X_valid.shape[0]
81         self.transform=transform
82     def __getitem__(self, index):
83         sample=self.x[index], self.y[index]
84         if self.transform:
85             sample=self.transform(sample)
86         return sample
87     def __len__(self):
88         return self.n_samples
89
90 class freesolvDataset_test (Dataset):
91     def __init__(self,transform=None):
92         #data loading
93         self.x=X_test.to_numpy().astype("float32")
94         self.y=y_test.to_numpy().astype("float32")
95         self.n_samples=X_test.shape[0]
96         self.transform=transform
97     def __getitem__(self, index):
98         sample=self.x[index], self.y[index]
99         if self.transform:
100             sample=self.transform(sample)
101         return sample
102     def __len__(self):
103         return self.n_samples
104
105 class ToTensor():
106     def __call__(self, sample):
107         inputs,targets=sample
108         inputs=torch.from_numpy(inputs.astype("float32
109 "))
110         targets=torch.tensor(targets.astype("float32")
111 )
112         #targets=targets.view(targets.shape[0],1)
113         return inputs,targets
114
115 #training set --
116 training_set = freesolvDataset_train(transform=
117 ToTensor())
118 train_loader = DataLoader(dataset=training_set,
119                             batch_size=64,
120                             shuffle=True)
121
122 dataiter_train = iter(train_loader)
123 data_train = dataiter_train.next()
124
125 def trainloader(config):
126     return DataLoader(dataset=training_set,
127                         batch_size=config["
128 batch_size"],
129                         shuffle=True,num_workers=2)
130
131 #kf-training set --
132 def kf_trainloader(config,k):
133     return DataLoader(dataset=
134 kf_freesolvDataset_train(k,transform=ToTensor())
135 ,
136                         batch_size=config["
137 batch_size"],
138                         shuffle=True,num_workers=2)
139
140 #kf-training set --
141 def kf_validloader(config,k):
142     return DataLoader(dataset=
143 kf_freesolvDataset_valid(k,transform=ToTensor())
144 ,
145                         batch_size=config["
146 batch_size"],
147                         shuffle=True,num_workers=2)
148
149 #validation set --
150 validation_set = freesolvDataset_valid(transform=
151 ToTensor())
152 valid_loader = DataLoader(dataset=validation_set,
153                             batch_size=64,
154                             shuffle=True)
155
156 dataiter_valid = iter(valid_loader)
157 data_valid = dataiter_valid.next()
158
159 def validloader(config):
160     return DataLoader(dataset=validation_set,
161                         batch_size=config["
162 batch_size"],
163                         shuffle=True,num_workers=2)
164
165 #test set --
166 test_set = freesolvDataset_test(transform=ToTensor())
167 test_loader = DataLoader(dataset=test_set,
168                             batch_size=8,
169                             shuffle=False)
170
171 dataiter_test = iter(test_loader)
172 data_test = dataiter_test.next()
173
174 def testloader(config):
175     return DataLoader(dataset=test_set,
176                         batch_size=4,
177                         shuffle=False,num_workers=2)

```

Listing 9: Defining data ladders

3.2 Defining Neural Networks

According to what I explained in Figure 5, I define neural networks for all three data sets. Of course, slight and sometimes fundamental differences exist between the neural networks of these three models. For example, it can be clearly seen that the last layer of the regression dataset's neural network does not have an activation function. Of course, there are other differences.

But in general, in all these neural networks, it is clear that at least some activation functions, three natural numbers that regulate the number of nodes in some layers, are defined as hyper-parameters. Of course, the learning rate is also a hyper-parameter that is defined, and it is better to introduce it in the following sections

```

1 n_samples,n_features=X_train.shape
2 class NeuralNetwork (nn.Module):
3     def __init__(self,n_input_features,l1, l2,l3,
4         config):
5         super (NeuralNetwork, self).__init__()
6         self.config = config
7         self.linear1=nn.Linear(n_input_features,4*math
8         .floor(n_input_features/2)+l1)
9         self.linear2=nn.Linear(l1+4*math.floor(
10        n_input_features/2),math.floor(n_input_features/2)
11        +l2)
12         self.D1=torch.nn.Dropout (config.get ("
13        drop_out_ratio1"))
14         self.linear3=nn.Linear(math.floor(
15        n_input_features/2)+l2,math.floor(n_input_features
16        /4)+l3)
17         self.D2=torch.nn.Dropout (config.get ("
18        drop_out_ratio2"))
19         self.linear5=nn.Linear(math.floor(
20        n_input_features/4)+l3,1)
21
22         self.a1 = self.config.get("a1")
23         self.a2 = self.config.get("a2")
24         self.a3 = self.config.get("a3")
25
26     @staticmethod
27     def activation_func(act_str):
28         if act_str=="tanh" or act_str=="sigmoid":
29             return eval("torch."+act_str)
30         elif act_str=="silu" or act_str=="relu" or
31         act_str=="leaky_relu" or act_str=="gelu":
32             return eval("torch.nn.functional."+act_str
33             )
34
35     def forward(self,x):
36         out=self.linear1(x)
37         out=self.activation_func(self.a1) (out.float())
38         out=self.linear2(out)
39         out=self.D1(out)
40         out=self.activation_func(self.a2) (out.float())
41         out=self.linear3(out)
42         out=self.activation_func(self.a3) (out.float())
43         out=self.D1(out)
44         out=self.linear5(out)
45         out=torch.sigmoid(out)
46         y_predicted=out
47         return y_predicted

```

Listing 10: Neural network for the Bbbp dataset

```

1 n_samples,n_features=X_train.shape
2 class NeuralNetwork (nn.Module):
3     def __init__(self,n_input_features,l1, l2,l3,
4         config):
5         super (NeuralNetwork, self).__init__()
6         self.config = config

```

```

6         self.linear1=nn.Linear(n_input_features,
7         n_input_features+l1)
8         self.linear2=nn.Linear(n_input_features+l1,
9         math.floor(n_input_features/2)+l2)
10        self.D1=torch.nn.Dropout (config.get ("
11        drop_out_ratio1"))
12        self.linear3=nn.Linear(math.floor(
13        n_input_features/2)+l2,math.floor(n_input_features
14        /4)+l3)
15        self.D2=torch.nn.Dropout (config.get ("
16        drop_out_ratio2"))
17        self.linear5=nn.Linear(math.floor(
18        n_input_features/4)+l3,1)
19
20        self.a1 = self.config.get("a1")
21        self.a2 = self.config.get("a2")
22        self.a3 = self.config.get("a3")
23
24    @staticmethod
25    def activation_func(act_str):
26        if act_str=="tanh" or act_str=="sigmoid":
27            return eval("torch."+act_str)
28        elif act_str=="silu" or act_str=="relu" or
29        act_str=="leaky_relu" or act_str=="gelu":
30            return eval("torch.nn.functional."+act_str
31            )
32
33    def forward(self,x):
34        out=self.linear1(x)
35        out=self.activation_func(self.a1) (out.float())
36        out=self.linear2(out)
37        out=self.D1(out)
38        out=self.activation_func(self.a2) (out.float())
39        out=self.linear3(out)
40        out=self.activation_func(self.a3) (out.float())
41        out=self.D1(out)
42        out=self.linear5(out)
43        out=torch.sigmoid(out)
44        y_predicted=out
45        return y_predicted

```

Listing 11: Neural network for the HIV dataset

```

1 n_samples,n_features=X_train.shape
2 class NeuralNetwork (nn.Module):
3     def __init__(self,n_input_features,l1, l2,l3,
4         config):
5         super (NeuralNetwork, self).__init__()
6         self.config = config
7         self.linear1=nn.Linear(n_input_features,4*math
8         .floor(n_input_features/2)+l1)
9         self.linear2=nn.Linear(l1+4*math.floor(
10        n_input_features/2),math.floor(n_input_features/2)
11        +l2)
12         self.D1=torch.nn.Dropout (config.get ("
13        drop_out_ratio1"))
14         self.linear3=nn.Linear(math.floor(
15        n_input_features/2)+l2,math.floor(n_input_features
16        /4)+l3)
17         self.D2=torch.nn.Dropout (config.get ("
18        drop_out_ratio2"))
19         self.linear5=nn.Linear(math.floor(
20        n_input_features/4)+l3,1)
21
22         self.a1 = self.config.get("a1")
23         self.a2 = self.config.get("a2")
24         self.a3 = self.config.get("a3")
25
26     @staticmethod
27     def activation_func(act_str):

```

```

20         if act_str=="tanh" or act_str=="sigmoid":
21             return eval("torch."+act_str)
22         elif act_str=="silu" or act_str=="relu" or
act_str=="leaky_relu" or act_str=="gelu":
23             return eval("torch.nn.functional."+act_str
24         )
25     def forward(self,x):
26         out=self.linear1(x)
27         out=self.activation_func(self.a1)(out.float())
28         out=self.linear2(out)
29         out=self.D1(out)
30         out=self.activation_func(self.a2)(out.float())
31         out=self.linear3(out)
32         out=self.activation_func(self.a3)(out.float())
33         out=self.D1(out)
34         out=self.linear5(out)
35         y_predicted=out
36         return y_predicted

```

Listing 12: Neural network for the FreeSolv dataset

3.3 Training loops

Perhaps the most critical part of this project is this part. In this section, two neural network training processes are done separately. It would help if you remembered that I discussed a semi-cross-validation in figure 4. My goal in this project was not to get high accuracy. I am looking for a mindset to optimize the hyper-parameters of a neural network.

Let's talk a little more about this semi-cross-validation. Why did I call it semi-cross-validation? The truth is that I do not stop the process of training the neural network weights when the local validation set is changed, so I look for an overfit model on the entire training set. After that, by resetting the neural network weights with the same hyper-parameters, I train the neural network as is customary in all projects. But what is the application of this mindset? I'm looking for hyper-parameters that have good ultimate accuracy and don't overfit easily compared to other hyper-parameters. Having the overfitted models' results helps me choose hyper-parameters that do not easily lead to overfitting on the training set.

A criticism may be raised that says that this process cannot be justified because the failure of a neural network may be largely dependent on the choice of learning rate and not due to the hyper-parameters themselves. I answer that my optimizer in these three projects is Adam, and the learning rate changes during the learning process. Therefore, our learning rate specifies only an initial value, which of course, I do not deny its effect in any way, but I have given up its direct impact on this project. On the other hand, I am looking for a package of hyper-parameters, including the learning rate itself. So, I do not ignore the learning rate's effect in model training.

```

1 def train_Bbbp(config,checkpoint_dir=None,max_iter=11)
2 :
3     net = NeuralNetwork(np.shape(feature_selection[0])
4     [0],config["l1"],config["l2"],config["l3"],config)
5
6     device = "cpu"
7     if torch.cuda.is_available():
8         device = "cuda:0"
9         if torch.cuda.device_count() > 1:
10             net = nn.DataParallel(net)
11         net.to(device)
12
13     #Define my loss function and optimizer
14     criterion=nn.BCELoss()
15     optimizer=torch.optim.Adam(net.parameters(), lr=
config["lr"])

```

```

16
17 # The 'checkpoint_dir' parameter gets passed by
Ray Tune when a checkpoint
18 # should be restored.
19 if checkpoint_dir:
20     checkpoint = os.path.join(checkpoint_dir, "
checkpoint")
21     model_state, optimizer_state = torch.load(
checkpoint)
22     net.load_state_dict(model_state)
23     optimizer.load_state_dict(optimizer_state)
24
25 localiter=0
26 for epoch in range(max_iter): # loop over the
dataset multiple times
27     running_loss1 = 0.0
28     epoch_steps1 = 0
29     for i, data in enumerate(kf_trainloader(config
,1), 0):
30         # get the inputs; data is a list of [
inputs, labels]
31         inputs, labels = data
32         inputs, labels = inputs.to(device), labels
.to(device)
33
34         # zero the parameter gradients
35         optimizer.zero_grad()
36
37         # forward + backward + optimize
38         outputs = net(inputs)
39         loss = criterion(outputs, labels)
40         loss.backward()
41         optimizer.step()
42
43         # print statistics
44         running_loss1 += loss.item()
45         epoch_steps1 += 1
46         if i % 2000 == 1999: # print every 2000
mini-batches
47             print("[%d, %5d] loss: %.3f" % (epoch
+ 1, i + 1, running_loss1 / epoch_steps1))
48             running_loss1 = 0.0
49
50         # Validation score
51         score1 = compute_score(net, kf_validloader(
config,1), device="cpu")
52
53 #second loop -
54
55     running_loss2 = 0.0
56     epoch_steps2 = 0
57     for i, data in enumerate(kf_trainloader(config
,2), 0):
58         # get the inputs; data is a list of [
inputs, labels]
59         inputs, labels = data
60         inputs, labels = inputs.to(device), labels
.to(device)
61
62         # zero the parameter gradients
63         optimizer.zero_grad()
64
65         # forward + backward + optimize
66         outputs = net(inputs)
67         loss = criterion(outputs, labels)
68         loss.backward()
69         optimizer.step()
70
71         # print statistics
72
73
74

```

```

75     running_loss2 += loss.item()
76     epoch_steps2 += 1
77     if i % 2000 == 1999: # print every 2000
mini-batches
78         print("[%d, %5d] loss: %.3f" % (epoch
+ 1, i + 1, running_loss2 / epoch_steps2))
79         running_loss2 = 0.0
80
81
82     # Validation score
83
84     score2 = compute_score(net, kf_validloader(
config,2), device="cpu")
85
86 #third loop -
87
88     running_loss3 = 0.0
89     epoch_steps3 = 0
90     for i, data in enumerate(kf_trainloader(config
,3), 0):
91         # get the inputs; data is a list of [
inputs, labels]
92         inputs, labels = data
93         inputs, labels = inputs.to(device), labels
.to(device)
94
95         # zero the parameter gradients
96         optimizer.zero_grad()
97
98         # forward + backward + optimize
99         outputs = net(inputs)
100         loss = criterion(outputs, labels)
101         loss.backward()
102         optimizer.step()
103
104         # print statistics
105         running_loss3 += loss.item()
106         epoch_steps3 += 1
107         if i % 2000 == 1999: # print every 2000
mini-batches
108             print("[%d, %5d] loss: %.3f" % (epoch
+ 1, i + 1, running_loss3 / epoch_steps3))
109             running_loss3 = 0.0
110
111
112     # Validation score
113
114     score3 = compute_score(net, kf_validloader(
config,3), device="cpu")
115
116 #forth loop -
117 for epoch in range(max_iter): # loop over the
dataset multiple times
118     running_loss4 = 0.0
119     epoch_steps4 = 0
120     for i, data in enumerate(kf_trainloader(config
,0), 0):
121         # get the inputs; data is a list of [
inputs, labels]
122         inputs, labels = data
123         inputs, labels = inputs.to(device), labels
.to(device)
124
125         # zero the parameter gradients
126         optimizer.zero_grad()
127
128         # forward + backward + optimize
129         outputs = net(inputs)
130         loss = criterion(outputs, labels)
131         loss.backward()
132         optimizer.step()
133

```

```

134     # print statistics
135     running_loss4 += loss.item()
136     epoch_steps4 += 1
137     if i % 2000 == 1999: # print every 2000
mini-batches
138         print("[%d, %5d] loss: %.3f" % (epoch
+ 1, i + 1, running_loss4 / epoch_steps4))
139         running_loss4 = 0.0
140
141
142     # Validation score
143
144     score4 = compute_score(net, kf_validloader(
config,0), device="cpu")
145
146
147     # Validation score
148
149     kf_score=np.min([score1,score2,score3,score4])
150     #print(f"score1: {score1:.4f}, score2: {score2
:.4f}, score3: {score3:.4f}. score4: {score4:.4f
}--->kf_score: {kf_score:.5f}")
151
152     localiter=localiter+1
153     val_score = compute_score(net, validloader(
config), device="cpu")
154
155     score=np.mean([val_score,val_score,kf_score])
156     -((localiter/max_iter)**2)*0.033-(kf_score-
val_score)/4
157
158
159     with tune.checkpoint_dir(epoch) as
checkpoint_dir:
160         path = os.path.join(checkpoint_dir, "
checkpoint")
161         torch.save((net.state_dict(), optimizer.
state_dict()), path)
162         tune.report(score=score,kf_score=kf_score,
val_score=val_score)
163
164     print("Finished Training")
165
166

```

Listing 13: Training loops

```

1 def train_HIV(config,checkpoint_dir=None,max_iter=11):
2     net = NeuralNetwork(np.shape(feature_selection[0])
[0],config["l1"],config["l2"],config["l3"],config)
3
4
5     device = "cpu"
6     if torch.cuda.is_available():
7         device = "cuda:0"
8         if torch.cuda.device_count() > 1:
9             net = nn.DataParallel(net)
10    net.to(device)
11
12    #Define my loss function and optimizer
13    criterion=nn.BCELoss()
14    optimizer=torch.optim.Adam(net.parameters(), lr=
config["lr"])
15
16
17    # The 'checkpoint_dir' parameter gets passed by
Ray Tune when a checkpoint
18    # should be restored.
19    if checkpoint_dir:
20        checkpoint = os.path.join(checkpoint_dir, "
checkpoint")

```

```

21     model_state, optimizer_state = torch.load(
22         checkpoint)
23     net.load_state_dict(model_state)
24     optimizer.load_state_dict(optimizer_state)
25
26
27     localiter=0
28     for epoch in range(max_iter): # loop over the
29         dataset multiple times
30         running_loss1 = 0.0
31         epoch_steps1 = 0
32         for i, data in enumerate(kf_trainloader(config
33             ,1), 0):
34             # get the inputs; data is a list of [
35             inputs, labels]
36             inputs, labels = data
37             inputs, labels = inputs.to(device), labels
38             .to(device)
39
40             # zero the parameter gradients
41             optimizer.zero_grad()
42
43             # forward + backward + optimize
44             outputs = net(inputs)
45             loss = criterion(outputs, labels)
46             loss.backward()
47             optimizer.step()
48
49             # print statistics
50             running_loss1 += loss.item()
51             epoch_steps1 += 1
52             if i % 2000 == 1999: # print every 2000
53                 mini-batches
54                 print("[%d, %5d] loss: %.3f" % (epoch
55                     + 1, i + 1, running_loss1 / epoch_steps1))
56                 running_loss1 = 0.0
57
58             # Validation score
59             score1 = compute_score(net, kf_validloader(
60                 config,1), device="cpu")
61
62     #second loop -
63
64     running_loss2 = 0.0
65     epoch_steps2 = 0
66     for i, data in enumerate(kf_trainloader(config
67         ,2), 0):
68         # get the inputs; data is a list of [
69         inputs, labels]
70         inputs, labels = data
71         inputs, labels = inputs.to(device), labels
72         .to(device)
73
74         # zero the parameter gradients
75         optimizer.zero_grad()
76
77         # forward + backward + optimize
78         outputs = net(inputs)
79         loss = criterion(outputs, labels)
80         loss.backward()
81         optimizer.step()
82
83         # print statistics
84         running_loss2 += loss.item()
85         epoch_steps2 += 1
86         if i % 2000 == 1999: # print every 2000
87             mini-batches
88             print("[%d, %5d] loss: %.3f" % (epoch
89                 + 1, i + 1, running_loss2 / epoch_steps2))
90             running_loss2 = 0.0

```

```

80
81
82     # Validation score
83
84     score2 = compute_score(net, kf_validloader(
85         config,2), device="cpu")
86
87     #third loop -
88
89     running_loss3 = 0.0
90     epoch_steps3 = 0
91     for i, data in enumerate(kf_trainloader(config
92         ,3), 0):
93         # get the inputs; data is a list of [
94         inputs, labels]
95         inputs, labels = data
96         inputs, labels = inputs.to(device), labels
97         .to(device)
98
99         # zero the parameter gradients
100         optimizer.zero_grad()
101
102         # forward + backward + optimize
103         outputs = net(inputs)
104         loss = criterion(outputs, labels)
105         loss.backward()
106         optimizer.step()
107
108         # print statistics
109         running_loss3 += loss.item()
110         epoch_steps3 += 1
111         if i % 2000 == 1999: # print every 2000
112             mini-batches
113             print("[%d, %5d] loss: %.3f" % (epoch
114                 + 1, i + 1, running_loss3 / epoch_steps3))
115             running_loss3 = 0.0
116
117     # Validation score
118
119     score3 = compute_score(net, kf_validloader(
120         config,3), device="cpu")
121
122     #forth loop -
123     # loop over the dataset multiple times
124     running_loss4 = 0.0
125     epoch_steps4 = 0
126     for i, data in enumerate(kf_trainloader(config
127         ,0), 0):
128         # get the inputs; data is a list of [
129         inputs, labels]
130         inputs, labels = data
131         inputs, labels = inputs.to(device), labels
132         .to(device)
133
134         # zero the parameter gradients
135         optimizer.zero_grad()
136
137         # forward + backward + optimize
138         outputs = net(inputs)
139         loss = criterion(outputs, labels)
140         loss.backward()
141         optimizer.step()
142
143         # print statistics
144         running_loss4 += loss.item()
145         epoch_steps4 += 1
146         if i % 2000 == 1999: # print every 2000
147             mini-batches
148             print("[%d, %5d] loss: %.3f" % (epoch
149                 + 1, i + 1, running_loss4 / epoch_steps4))
150             running_loss4 = 0.0

```



```

140
141
142     # Validation score
143
144     score4 = compute_score(net, kf_validloader(
145         config,0), device="cpu")
146
147 #global loop -
148 for layer in net.children():
149     if hasattr(layer, 'reset_parameters'):
150         layer.reset_parameters()
151 for epoch in range(max_iter):
152     running_loss = 0.0
153     epoch_steps = 0
154     for i, data in enumerate(trainloader(config),
155         0):
156         # get the inputs; data is a list of [
157         inputs, labels]
158         inputs, labels = data
159         inputs, labels = inputs.to(device), labels
160         .to(device)
161
162         # zero the parameter gradients
163         optimizer.zero_grad()
164
165         # forward + backward + optimize
166         outputs = net(inputs)
167         loss = criterion(outputs, labels)
168         loss.backward()
169         optimizer.step()
170
171         # print statistics
172         running_loss += loss.item()
173         epoch_steps += 1
174         if i % 2000 == 1999: # print every 2000
175             mini-batches
176             print("[%d, %5d] loss: %.3f" % (epoch
177 + 1, i + 1, running_loss / epoch_steps))
178             running_loss = 0.0
179
180     # Validation score
181
182     kf_score=np.min([score1,score2,score3,score4])
183
184     localiter=localiter+1
185     val_score = compute_score(net, validloader(
186         config), device="cpu")
187
188     score=np.mean([val_score,val_score,kf_score])
189     -((localiter/max_iter)**2)*0.04-(kf_score-
190     val_score)/4
191
192     with tune.checkpoint_dir(epoch) as
193     checkpoint_dir:
194         path = os.path.join(checkpoint_dir, "
195         checkpoint")
196         torch.save((net.state_dict(), optimizer.
197         state_dict()), path)
198         tune.report(score=score,kf_score=kf_score,
199         val_score=val_score)
200
201 print("Finished Training")

```

Listing 14: Training loops

```

1 def train_freesolv(config,checkpoint_dir=None,max_iter
2 =11):

```

```

3     net = NeuralNetwork(np.shape(feature_selection[0])
4     [0],config["11"],config["12"],config["13"],config)
5
6     device = "cpu"
7     if torch.cuda.is_available():
8         device = "cuda:0"
9         if torch.cuda.device_count() > 1:
10             net = nn.DataParallel(net)
11     net.to(device)
12
13     #Define my loss function and optimizer
14     criterion=nn.MSELoss()
15     optimizer=torch.optim.Adam(net.parameters(), lr=
16     config["1r"])
17
18     # The 'checkpoint_dir' parameter gets passed by
19     Ray Tune when a checkpoint
20     # should be restored.
21     if checkpoint_dir:
22         checkpoint = os.path.join(checkpoint_dir, "
23         checkpoint")
24         model_state, optimizer_state = torch.load(
25         checkpoint)
26         net.load_state_dict(model_state)
27         optimizer.load_state_dict(optimizer_state)
28
29     localiter=0
30     for epoch in range(max_iter): # loop over the
31     dataset multiple times
32         running_loss1 = 0.0
33         epoch_steps1 = 0
34         for i, data in enumerate(kf_trainloader(config
35         ,1), 0):
36             # get the inputs; data is a list of [
37             inputs, labels]
38             inputs, labels = data
39             inputs, labels = inputs.to(device), labels
40             .to(device)
41
42             # zero the parameter gradients
43             optimizer.zero_grad()
44
45             # forward + backward + optimize
46             outputs = net(inputs)
47             loss = criterion(outputs, labels)
48             loss.backward()
49             optimizer.step()
50
51             # print statistics
52             running_loss1 += loss.item()
53             epoch_steps1 += 1
54             if i % 2000 == 1999: # print every 2000
55                 mini-batches
56                 print("[%d, %5d] loss: %.3f" % (epoch
57 + 1, i + 1, running_loss1 / epoch_steps1))
58                 running_loss1 = 0.0
59
60             # Validation loss
61             loss1 = compute_loss(net, kf_validloader(
62             config,1), device="cpu")
63
64     #second loop -
65
66     running_loss2 = 0.0
67     epoch_steps2 = 0

```

```

61     for i, data in enumerate(kf_trainloader(config
62     ,2), 0):
63         # get the inputs; data is a list of [
64         inputs, labels]
65         inputs, labels = data
66         inputs, labels = inputs.to(device), labels
67         .to(device)
68
69         # zero the parameter gradients
70         optimizer.zero_grad()
71
72         # forward + backward + optimize
73         outputs = net(inputs)
74         loss = criterion(outputs, labels)
75         loss.backward()
76         optimizer.step()
77
78         # print statistics
79         running_loss2 += loss.item()
80         epoch_steps2 += 1
81         if i % 2000 == 1999: # print every 2000
82             mini-batches
83             print("[%d, %5d] loss: %.3f" % (epoch
84             + 1, i + 1, running_loss2 / epoch_steps2))
85             running_loss2 = 0.0
86
87         # Validation loss
88
89         loss2 = compute_loss(net, kf_validloader(
90         config,2), device="cpu")
91
92     #third loop -
93
94     running_loss3 = 0.0
95     epoch_steps3 = 0
96     for i, data in enumerate(kf_trainloader(config
97     ,3), 0):
98         # get the inputs; data is a list of [
99         inputs, labels]
100         inputs, labels = data
101         inputs, labels = inputs.to(device), labels
102         .to(device)
103
104         # zero the parameter gradients
105         optimizer.zero_grad()
106
107         # forward + backward + optimize
108         outputs = net(inputs)
109         loss = criterion(outputs, labels)
110         loss.backward()
111         optimizer.step()
112
113         # print statistics
114         running_loss3 += loss.item()
115         epoch_steps3 += 1
116         if i % 2000 == 1999: # print every 2000
117             mini-batches
118             print("[%d, %5d] loss: %.3f" % (epoch
119             + 1, i + 1, running_loss3 / epoch_steps3))
120             running_loss3 = 0.0
121
122         # Validation loss
123
124         loss3 = compute_loss(net, kf_validloader(
125         config,3), device="cpu")
126
127     #forth loop -
128     # loop over the dataset multiple times

```

```

121     running_loss4 = 0.0
122     epoch_steps4 = 0
123     for i, data in enumerate(kf_trainloader(config
124     ,0), 0):
125         # get the inputs; data is a list of [
126         inputs, labels]
127         inputs, labels = data
128         inputs, labels = inputs.to(device), labels
129         .to(device)
130
131         # zero the parameter gradients
132         optimizer.zero_grad()
133
134         # forward + backward + optimize
135         outputs = net(inputs)
136         loss = criterion(outputs, labels)
137         loss.backward()
138         optimizer.step()
139
140         # print statistics
141         running_loss4 += loss.item()
142         epoch_steps4 += 1
143         if i % 2000 == 1999: # print every 2000
144             mini-batches
145             print("[%d, %5d] loss: %.3f" % (epoch
146             + 1, i + 1, running_loss4 / epoch_steps4))
147             running_loss4 = 0.0
148
149         # Validation loss
150
151         loss4 = compute_loss(net, kf_validloader(
152         config,0), device="cpu")
153
154     # Validation loss
155
156     kf_loss=np.max([loss1,loss2,loss3,loss4])
157
158     #global loop -
159     for layer in net.children():
160         if hasattr(layer, 'reset_parameters'):
161             layer.reset_parameters()
162     for epoch in range(max_iter):
163         running_loss = 0.0
164         epoch_steps = 0
165         for i, data in enumerate(trainloader(config),
166         0):
167             # get the inputs; data is a list of [
168             inputs, labels]
169             inputs, labels = data
170             inputs, labels = inputs.to(device), labels
171             .to(device)
172
173             # zero the parameter gradients
174             optimizer.zero_grad()
175
176             # forward + backward + optimize
177             outputs = net(inputs)
178             loss = criterion(outputs, labels)
179             loss.backward()
180             optimizer.step()
181
182             # print statistics
183             running_loss += loss.item()
184             epoch_steps += 1
185             if i % 2000 == 1999: # print every 2000
186                 mini-batches
187                 print("[%d, %5d] loss: %.3f" % (epoch
188                 + 1, i + 1, running_loss / epoch_steps))

```

```

182         running_loss4 = 0.0
183
184         localiter=localiter+1
185         val_loss = compute_loss(net, validloader(
186             config), device="cpu")
187
188         loss=np.mean([val_loss,val_loss,kf_loss])+((
189             localiter/max_iter)**2)*0.2+(val_loss-kf_loss)/2
190
191
192         with tune.checkpoint_dir(epoch) as
193             checkpoint_dir:
194                 path = os.path.join(checkpoint_dir, "
195                     checkpoint")
196                 torch.save((net.state_dict(), optimizer.
197                     state_dict()), path)
198                 tune.report(loss=loss,kf_loss=kf_loss,val_loss
199                     =val_loss)
200
201         print("Finished Training")

```

Listing 15: Training loops

I have to explain the leading indicator I chose to choose the best model. As I said before, I have penalized the model for overfitting the neural networks in more iterations. In addition, the sooner Ray concludes that the neural network does not need more training; in other words, the training process stops in fewer iterations, the less likely the final model will be overfit. (I first observed this experimentally while working with the data and was inspired to choose the final criterion) Therefore, I also penalized the models that use the maximum possible iteration for training.

Ultimately, I expect my chosen metric to be an estimate of the result of the test set. In the following, we will examine how successful I have been.

3.4 Applying the best model to the test set

After the process of training and generating the best model, it is apparent that we have to apply the final algorithm to the test set. This will help us to understand whether we are caught in the Biased Optimization trap or not.

```

1 def test_best_model(best_trial):
2     best_trained_model = NeuralNetwork(np.shape(
3         feature_selection[0])[0],best_trial.config["11"],
4         best_trial.config["12"],best_trial.config["13"],
5         best_trial.config)
6     device = "cuda:0" if torch.cuda.is_available()
7     else "cpu"
8     best_trained_model.to(device)
9
10    checkpoint_path = os.path.join(best_trial.
11        checkpoint.value, "checkpoint")
12
13    model_state, optimizer_state = torch.load(
14        checkpoint_path)
15    best_trained_model.load_state_dict(model_state)
16
17    test_score = compute_score(best_trained_model,
18        testloader(best_trial.config), device)
19    print("Best trial test set score:         - {}
20        ".format(test_score))
21    return best_trial.config, best_trained_model

```

Listing 16: Applying the best model to the test set

```

1 def test_best_model(best_trial):

```

```

2     best_trained_model = NeuralNetwork(np.shape(
3         feature_selection[0])[0],best_trial.config["11"],
4         best_trial.config["12"],best_trial.config["13"],
5         best_trial.config)
6     device = "cuda:0" if torch.cuda.is_available()
7     else "cpu"
8     best_trained_model.to(device)
9
10    checkpoint_path = os.path.join(best_trial.
11        checkpoint.value, "checkpoint")
12
13    model_state, optimizer_state = torch.load(
14        checkpoint_path)
15    best_trained_model.load_state_dict(model_state)
16
17    test_score = compute_score(best_trained_model,
18        testloader(best_trial.config), device)
19    print("Best trial test set score:         - {}
20        ".format(test_score))
21    return best_trial.config, best_trained_model

```

Listing 17: Applying the best model to the test set

```

1 def test_best_model(best_trial):
2     best_trained_model = NeuralNetwork(np.shape(
3         feature_selection[0])[0],best_trial.config["11"],
4         best_trial.config["12"],best_trial.config["13"],
5         best_trial.config)
6     device = "cuda:0" if torch.cuda.is_available()
7     else "cpu"
8     best_trained_model.to(device)
9
10    checkpoint_path = os.path.join(best_trial.
11        checkpoint.value, "checkpoint")
12
13    model_state, optimizer_state = torch.load(
14        checkpoint_path)
15    best_trained_model.load_state_dict(model_state)
16
17    test_loss = compute_loss(best_trained_model,
18        testloader(best_trial.config), device)
19    print("Best trial test set loss:         - {}
20        ".format(test_loss))
21    return best_trial.config, best_trained_model

```

Listing 18: Applying the best model to the test set

3.5 Defining the search space

In this part, I specify the space in which we are going to find the optimal hyper-parameters. In this part, all three data sets are similar.

```

1 config = {
2     "11": tune.choice([2**6,2**7,2**8]),
3     "12": tune.choice([2**6,2**7,2**8]),
4     "13": tune.choice([2**6,2**7,2**8]),
5     "lr": tune.quniform(0.0001, 0.1,0.0001),
6     "drop_out_ratio1": tune.quniform(0.3, 0.65,0.01)
7     ,
8     "drop_out_ratio2": tune.quniform(0.01, 1,0.01),
9     "a1":tune.choice(["relu","leaky_relu","silu"]),
10    "a2":tune.choice(["gelu","leaky_relu"]),
11    "a3":tune.choice(["relu","gelu"]),
12    "batch_size": tune.choice([ 32, 64, 128]),
13 }

```

Listing 19: Config

```

1 config = {
2     "11": tune.choice([2**6,2**7,2**8]),
3     "12": tune.choice([2**6,2**7,2**8]),
4     "13": tune.choice([2**6,2**7,2**8]),

```

```

5  "lr": tune.quniform(0.0001, 0.1, 0.0001),
6  "drop_out_ratio1": tune.quniform(0.3, 0.65, 0.01)
7
8  ,
9  "drop_out_ratio2": tune.quniform(0.01, 1, 0.01),
10 "a1": tune.choice(["relu", "leaky_relu", "silu"]),
11 "a2": tune.choice(["gelu", "leaky_relu"]),
12 "a3": tune.choice(["relu", "gelu"]),
    "batch_size": tune.choice([ 32, 64, 128]),
    }

```

Listing 20: Config

```

1 config = {
2     "l1": tune.choice([2**6, 2**7, 2**8]),
3     "l2": tune.choice([2**6, 2**7, 2**8]),
4     "l3": tune.choice([2**6, 2**7, 2**8]),
5     "lr": tune.quniform(0.0001, 0.1, 0.0001),
6     "drop_out_ratio1": tune.quniform(0.3, 0.65, 0.01)
7
8     ,
9     "drop_out_ratio2": tune.quniform(0.01, 1, 0.01),
10    "a1": tune.choice(["relu", "leaky_relu", "silu"]),
11    "a2": tune.choice(["gelu", "leaky_relu"]),
12    "a3": tune.choice(["relu", "gelu"]),
    "batch_size": tune.choice([ 32, 64, 128]),
    }

```

Listing 21: Config

3.6 Main function

Almost everything happens in this function. Note that this is where the functions are finally one by one called, and the optimization of the hyper-parameters begins. In this section, I use the search algorithm of OptunaSearch[6], which is based on Bayesian optimization. In addition, ASHAScheduler⁷ as a scheduler plays an active role in reducing computational costs for me.

```

1 from ray.tune import CLIReporter
2 from functools import partial
3 def main(num_samples=10, max_num_epochs=100,
4         gpus_per_trial=2):
5
6     config = {
7         "l1": tune.choice([2**6, 2**7, 2**8]),
8         "l2": tune.choice([2**6, 2**7, 2**8]),
9         "l3": tune.choice([2**6, 2**7, 2**8]),
10        "lr": tune.quniform(0.0001, 0.1, 0.0001),
11        "drop_out_ratio1": tune.quniform(0.3, 0.65, 0.01)
12
13        ,
14        "drop_out_ratio2": tune.quniform(0.01, 1, 0.01),
15        "a1": tune.choice(["relu", "leaky_relu", "silu"]),
16        "a2": tune.choice(["gelu", "leaky_relu"]),
17        "a3": tune.choice(["relu", "gelu"]),
18        "batch_size": tune.choice([ 32, 64, 128]),
19    }
20
21    from ray.tune.suggest.optuna import OptunaSearch
22    from ray.tune.suggest import ConcurrencyLimiter
23    search_alg = OptunaSearch(
24        metric="score", #or accuracy, etc.
25        mode="max", #or max
26        seed = 42,
27    )
28    search_alg = ConcurrencyLimiter(search_alg,
29        max_concurrent=10)
30
31    scheduler = ASHAScheduler(
32        metric="score",
33        mode="max",
34        max_t=max_num_epochs,
35        reduction_factor=2,

```

```

33        grace_period=4,
34        brackets=5
35    )
36
37    reporter = CLIReporter(
38        metric_columns=["score", "val_score", "kf_score"
39        , "training_iteration"]
40    )
41
42    # wrap data loading and training for tuning using
43    # 'partial'
44    # (note that there exist other methods for this
45    # purpose)
46    max_iter=max_num_epochs
47    result = tune.run(
48        partial(train_Bbbp, max_iter=max_iter),
49        scheduler=scheduler,
50        search_alg=search_alg,
51        num_samples=num_samples,
52        config=config,
53        verbose=3,
54        checkpoint_score_attr="score",
55        checkpoint_freq=0,
56        keep_checkpoints_num=1,
57        progress_reporter=reporter,
58        resources_per_trial={"cpu": 1, "gpu":
59        gpus_per_trial},
60        stop={"training_iteration": max_iter},
61    )
62
63    best_trial = result.get_best_trial("score", "max",
64        "last")
65    print("Best trial config: {}".format(best_trial.
66        config))
67    print("Average ROC_AUC score of the chosen model
68    for different validation sets in 4-fold cross
69    validation: {}".format(best_trial.last_result["
70    kf_score"]))
71    print("Best trial final validation ROC_AUC:
72    - - {}".format(best_trial.last_result["
73    val_score"]))
74    print("Best trial final score: - - {}".
75        format(best_trial.last_result["score"]))
76
77    if ray.util.client ray.is_connected():
78        # If using Ray Client, we want to make sure
79        # checkpoint access
80        # happens on the server. So we wrap `
81        test_best_model` in a Ray task.
82        # We have to make sure it gets executed on the
83        # same node that
84        # ``tune.run`` is called on.
85        from ray.util.ml_utils.node import
86        force_on_current_node
87        remote_fn = force_on_current_node(ray.remote(
88        test_best_model))
89        ray.get(remote_fn.remote(best_trial))
90    else:
91        best_trial.config, best_trained_model=
92        test_best_model(best_trial)
93    return best_trial.config, best_trained_model
94
95 configuration, Bneuralnetwork=main(num_samples=100,
96    max_num_epochs=14, gpus_per_trial=0)

```

Listing 22: Main function

```

1 Best trial config: {'l1': 256, 'l2': 256, 'l3': 256, '
    lr': 0.00023721028804998172, 'drop_out_ratio1':
    0.5269833252256684, 'drop_out_ratio2':
    0.012888067179906312, 'a1': 'relu', 'a2': '
    leaky_relu', 'a3': 'relu', 'batch_size': 128}

```

```

2 Average ROC_AUC score of the chosen model for
  different validation sets in 4-fold cross
  validation: 0.9841849148418491
3 Best trial final validation ROC_AUC:0.9066445239312796
4 Best trial final score:0.9001562145698741
5 Best trial test set score:0.9153247029986053

```

Listing 23: Minimized output

```

1 from ray.tune import CLIReporter
2 from functools import partial
3 def main(num_samples=10, max_num_epochs=100,
  gpus_per_trial=2):
4
5     config = {
6         "l1": tune.choice([2**6, 2**7, 2**8]),
7         "l2": tune.choice([2**6, 2**7, 2**8]),
8         "l3": tune.choice([2**6, 2**7, 2**8]),
9         "lr": tune.quniform(0.0001, 0.1, 0.0001),
10        "drop_out_ratio1": tune.quniform(0.3, 0.65, 0.01)
11    ,
12        "drop_out_ratio2": tune.quniform(0.01, 1, 0.01),
13        "a1":tune.choice(["relu", "leaky_relu", "silu"]),
14        "a2":tune.choice(["gelu", "leaky_relu"]),
15        "a3":tune.choice(["relu", "gelu"]),
16        "batch_size": tune.choice([ 32, 64, 128]),
17    }
18
19    from ray.tune.suggest.optuna import OptunaSearch
20
21    from ray.tune.suggest import ConcurrencyLimiter
22    search_alg = OptunaSearch(
23        metric="score", #or accuracy, etc.
24        mode="max", #or max
25        seed = 42,
26    )
27    search_alg = ConcurrencyLimiter(search_alg,
28        max_concurrent=10)
29
30    scheduler = ASHAScheduler(
31        metric = "score",
32        mode="max",
33        max_t=max_num_epochs,
34        reduction_factor=2,
35        grace_period=4,
36        brackets=5
37    )
38
39    reporter = CLIReporter(
40        metric_columns=["score", "val_score", "kf_score"
41        , "training_iteration"]
42    )
43
44    # wrap data loading and training for tuning using
45    # 'partial'
46    # (note that there exist other methods for this
47    # purpose)
48    max_iter=max_num_epochs
49    result = tune.run(
50        partial(train_HIV, max_iter=max_iter),
51        scheduler=scheduler,
52        search_alg=search_alg,
53        num_samples=num_samples,
54        config=config,
55        verbose=3,
56        checkpoint_score_attr="score",
57        checkpoint_freq=0,
58        keep_checkpoints_num=1,
59        progress_reporter=reporter,
60        resources_per_trial={"cpu": 2, "gpu":
61        gpus_per_trial},
62        stop={"training_iteration": max_iter},

```

```

57    )
58
59    best_trial = result.get_best_trial("score", "max",
60        "last")
61    print("Best trial config: {}".format(best_trial.
62        config))
63    print("Average ROC_AUC score of the chosen model
64    for different validation sets in 4-fold cross
65    validation: {}".format(best_trial.last_result["
66    kf_score"]))
67    print("Best trial final validation ROC_AUC:
68    - - {}".format(best_trial.last_result["
69    val_score"]))
70    print("Best trial final score: - - {}".
71        .format(best_trial.last_result["score"]))
72
73    if ray.util.client.ray.is_connected():
74        # If using Ray Client, we want to make sure
75        checkpoint access
76        # happens on the server. So we wrap `
77        test_best_model` in a Ray task.
78        # We have to make sure it gets executed on the
79        same node that
80        # `tune.run` is called on.
81        from ray.util.ml_utils.node import
82        force_on_current_node
83        remote_fn = force_on_current_node(ray.remote(
84        test_best_model))
85        ray.get(remote_fn.remote(best_trial))
86    else:
87        best_trial.config, best_trained_model=
88        test_best_model(best_trial)
89    return best_trial.config, best_trained_model
90
91 configuration, Bneuralnetwork=main(num_samples=10,
92    max_num_epochs=8, gpus_per_trial=0)

```

Listing 24: Main function

```

1 Best trial config: {'l1': 256, 'l2': 64, 'l3': 256, '
  lr': 0.00032476735706274504, 'drop_out_ratio1':
  0.32254266574935124, 'drop_out_ratio2':
  0.7912456325487695, 'a1': 'relu', 'a2': '
  leaky_relu', 'a3': 'relu', 'batch_size': 128}
2 Average ROC_AUC score of the chosen model for
  different validation sets in 4-fold cross
  validation:0.92654115475632112
3 Best trial final validation ROC_AUC:0.810365456975426
4 Best trial final score:0.799459873651485
5 Best trial test set score:0.81422115885625425

```

Listing 25: Minimized output

```

1 from ray.tune import CLIReporter
2 from functools import partial
3 def main(num_samples=10, max_num_epochs=100,
  gpus_per_trial=2):
4
5     config = {
6         "l1": tune.choice([2**6, 2**7, 2**8]),
7         "l2": tune.choice([2**6, 2**7, 2**8]),
8         "l3": tune.choice([2**6, 2**7, 2**8]),
9         "lr": tune.quniform(0.0001, 0.1, 0.0001),
10        "drop_out_ratio1": tune.quniform(0.3, 0.65, 0.01)
11    ,
12        "drop_out_ratio2": tune.quniform(0.01, 1, 0.01),
13        "a1":tune.choice(["relu", "leaky_relu", "silu"]),
14        "a2":tune.choice(["gelu", "leaky_relu"]),
15        "a3":tune.choice(["relu", "gelu"]),
16        "batch_size": tune.choice([ 32, 64, 128]),
17    }

```

```

18 from ray.tune.suggest.optuna import OptunaSearch
19
20 from ray.tune.suggest import ConcurrencyLimiter
21 search_alg = OptunaSearch(
22     metric="loss", #or accuracy, etc.
23     mode="min", #or max
24     seed = 42,
25 )
26
27 search_alg = ConcurrencyLimiter(search_alg,
28                                 max_concurrent=10)
29
30 scheduler = ASHAScheduler(
31     metric="loss",
32     mode="min",
33     max_t=max_num_epochs,
34     reduction_factor=2,
35     grace_period=4,
36     brackets=5
37 )
38
39 reporter = CLIReporter(
40     metric_columns=["loss", "val_loss", "kf_loss", "
41                     training_iteration"]
42 )
43
44 # wrap data loading and training for tuning using
45 # 'partial'
46 # (note that there exist other methods for this
47 # purpose)
48 max_iter=max_num_epochs
49 result = tune.run(
50     partial(train_freesolv, max_iter=max_iter),
51     scheduler=scheduler,
52     search_alg=search_alg,
53     num_samples=num_samples,
54     config=config,
55     verbose=3,
56     checkpoint_score_attr="loss",
57     checkpoint_freq=0,
58     keep_checkpoints_num=1,
59     progress_reporter=reporter,
60     resources_per_trial={"cpu": 1, "gpu":
61                           gpus_per_trial},
62     stop={"training_iteration": max_iter},
63 )
64
65 best_trial = result.get_best_trial("loss", "min",
66                                    "last")
67 print("Best trial config: {}".format(best_trial.
68                                     config))
69 print("Average MSE loss of the chosen model for
70       different validation sets in 4-fold cross
71       validation: {}".format(best_trial.last_result["
72                               kf_loss"]))
73 print("Best trial final validation MSEloss:
74       - - {} ".format(best_trial.last_result["
75                               val_loss"]))
76 print("Best trial final loss: {} ".
77       format(best_trial.last_result["loss"]))
78
79 if ray.util.client ray.is_connected():
80     # If using Ray Client, we want to make sure
81     # checkpoint access
82     # happens on the server. So we wrap `
83     test_best_model` in a Ray task.
84     # We have to make sure it gets executed on the
85     # same node that
86     # ``tune.run`` is called on.

```

```

73 from ray.util.ml_utils.node import
74 force_on_current_node
75 remote_fn = force_on_current_node(ray.remote(
76     test_best_model))
77 ray.get(remote_fn.remote(best_trial))
78 else:
79     best_trial.config, best_trained_model=
80     test_best_model(best_trial)
81 return best_trial.config, best_trained_model
82
83 configuration, Bneuralnetwork=main(num_samples=15,
84                                    max_num_epochs=30,
85                                    gpus_per_trial=0)

```

Listing 26: Main function

```

1 Best trial config: {'l1': 64, 'l2': 64, 'l3': 256, 'lr
': 0.004048778818153414, 'drop_out_ratio1':
0.3383871708118728, 'drop_out_ratio2':
0.4616014195190429, 'a1': 'relu', 'a2': '
leaky_relu', 'a3': 'gelu', 'batch_size': 64}
2 Average MSE loss of the chosen model for different
validation sets in 4-fold cross validation
:0.9645641363221736
3 Best trial final validation MSEloss:1.13998745615874
4 Best trial final loss:1.3503651421559874
5 Best trial test set loss:1.190654159875324

```

Listing 27: Main function

4 FINAL RESULTS

Now we have the optimal hyper-parameters. But have only the initial weights of the neural network caused acceptable results? To answer this question, we again train the neural network from the beginning and check the results on all three datasets.

```

1 def final_traing(config,max_iter=14):
2     net = NeuralNetwork(np.shape(feature_selection[0])
3       [0],config["l1"],config["l2"],config["l3"],config)
4
5     device = "cpu"
6     if torch.cuda.is_available():
7         device = "cuda:0"
8         if torch.cuda.device_count() > 1:
9             net = nn.DataParallel(net)
10    net.to(device)
11
12    #Define my loss function and optimizer
13    criterion=nn.BCELoss()
14    optimizer=torch.optim.Adam(net.parameters(), lr=
15      config["lr"])
16
17    for epoch in range(max_iter):
18        running_loss = 0.0
19        epoch_steps = 0
20        for i, data in enumerate(trainloader(config), 0)
21          :
22            # get the inputs; data is a list of [inputs,
23              labels]
24            inputs, labels = data
25            inputs, labels = inputs.to(device), labels.to(
26              device)
27            # zero the parameter gradients
28            optimizer.zero_grad()
29            # forward + backward + optimize
30            outputs = net(inputs)
31            loss = criterion(outputs, labels)
32            loss.backward()
33            optimizer.step()
34            # print statistics
35            running_loss += loss.item()
36            epoch_steps += 1
37            if i % 2000 == 1999: # print every 2000 mini-
38              batches
39              print("[%d, %5d] loss: %.3f" % (epoch + 1, i
40                + 1,running_loss / epoch_steps))
41              running_loss = 0.0
42
43            # Validation score
44            return net
45
46    neurlnet=final_traing(configuration,max_iter=14)
47    with torch.no_grad():
48        y_predicted=neurlnet(X_test_)
49        y_predicted_cls=y_predicted.round()
50        acc= y_predicted_cls.eq(y_test_).sum()/float(
51          y_test_.shape[0])
52
53    #print(f'accuracy={acc:.4f}')
54    a=confmat( y_predicted_cls.int(),y_test_.int())
55    print(f'accuracy={acc:.19f}
56          balanced_accuracy_score={balanced_accuracy_score(
57            y_predicted_cls.int(),y_test_.int()):.19f}
58          ROC_AUC={compute_score(neurlnet, testloader(
59            configuration), device="cpu") :.19f}')
60    print(a)

```

Listing 28: Retraining the neural network

```

1 accuracy=0.8970588235294118
2 balanced_accuracy_score=0.8622641509433963

```

```

3 ROC_AUC=0.92352994855429571248
4 tensor([[ 36  12],
5         [ 9 147]])

```

Listing 29: Final results

```

1 def final_traing(config,max_iter=14):
2     net = NeuralNetwork(np.shape(feature_selection[0])
3       [0],config["l1"],config["l2"],config["l3"],config)
4
5     device = "cpu"
6     if torch.cuda.is_available():
7         device = "cuda:0"
8         if torch.cuda.device_count() > 1:
9             net = nn.DataParallel(net)
10    net.to(device)
11
12    #Define my loss function and optimizer
13    criterion=nn.BCELoss()
14    optimizer=torch.optim.Adam(net.parameters(), lr=
15      config["lr"])
16
17    for epoch in range(max_iter):
18        running_loss = 0.0
19        epoch_steps = 0
20        for i, data in enumerate(trainloader(config), 0)
21          :
22            # get the inputs; data is a list of [inputs,
23              labels]
24            inputs, labels = data
25            inputs, labels = inputs.to(device), labels.to(
26              device)
27            # zero the parameter gradients
28            optimizer.zero_grad()
29            # forward + backward + optimize
30            outputs = net(inputs)
31            loss = criterion(outputs, labels)
32            loss.backward()
33            optimizer.step()
34            # print statistics
35            running_loss += loss.item()
36            epoch_steps += 1
37            if i % 2000 == 1999: # print every 2000 mini-
38              batches
39              print("[%d, %5d] loss: %.3f" % (epoch + 1, i
40                + 1,running_loss / epoch_steps))
41              running_loss = 0.0
42
43            # Validation score
44            return net
45
46    neurlnet=final_traing(configuration,max_iter=9)
47    with torch.no_grad():
48        y_predicted=neurlnet(X_test_)
49        y_predicted_cls=y_predicted.round()
50        acc= y_predicted_cls.eq(y_test_).sum()/float(
51          y_test_.shape[0])
52
53    #print(f'accuracy={acc:.4f}')
54    a=confmat( y_predicted_cls.int(),y_test_.int())
55    print(f'accuracy={acc:.19f}
56          balanced_accuracy_score={balanced_accuracy_score(
57            y_predicted_cls.int(),y_test_.int()):.19f}
58          ROC_AUC={compute_score(neurlnet, testloader(
59            configuration), device="cpu") :.19f}')
60    print(a)

```

Listing 30: Retraining the neural network

```

1 accuracy=0.9720398735716023
2 balanced_accuracy_score=0.8807050684974516

```

```

3 ROC_AUC=0.83159499309512508965
4 tensor([[3961, 10],
5        [ 105, 37]])

```

Listing 31: Final results

```

1 def final_traing(config,max_iter=14):
2     net = NeuralNetwork(np.shape(feature_selection[0])
3       [0],config["l1"],config["l2"],config["l3"],config)
4
5     device = "cpu"
6     if torch.cuda.is_available():
7         device = "cuda:0"
8         if torch.cuda.device_count() > 1:
9             net = nn.DataParallel(net)
10    net.to(device)
11
12    #Define my loss function and optimizer
13    criterion=nn.MSELoss()
14    optimizer=torch.optim.Adam(net.parameters(), lr=
15      config["lr"])
16
17    for epoch in range(max_iter):
18        running_loss = 0.0
19        epoch_steps = 0
20        for i, data in enumerate(trainloader(config), 0)
21        :
22            # get the inputs; data is a list of [inputs,
23            labels]
24            inputs, labels = data
25            inputs, labels = inputs.to(device), labels.to(
26            device)
27            # zero the parameter gradients
28            optimizer.zero_grad()
29            # forward + backward + optimize
30            outputs = net(inputs)
31            loss = criterion(outputs, labels)
32            loss.backward()
33            optimizer.step()
34            # print statistics
35            running_loss += loss.item()
36            epoch_steps += 1
37            if i % 2000 == 1999: # print every 2000 mini-
38            batches
39                print("[%d, %5d] loss: %.3f" % (epoch + 1, i
40                + 1,running_loss / epoch_steps))
41                running_loss = 0.0
42        return net
43
44    neurlnet=final_traing(configuration,max_iter=30)
45    loss=compute_loss(neurlnet, testloader(config), device
46      ="cpu")
47    print(f'loss={loss:.10f}')

```

Listing 32: Retraining the neural network

```

1 loss=1.24510621941294373

```

Listing 33: Final results

5 CONCLUSION

Among these three datasets, I have examined the results of the Bbbp dataset more than the other datasets. Therefore, I thought it would be good to put the results of working with this data in the form of a table.

	ROC_AUC	Search Algorithm	Changes compared to the previous version	Optimizer
1 Primary neural network	0.8920	-	-	SGD
2 hyper-parameter tuning with ray tune	0.9066	Random search	Using Ray tune	SGD
3 hyper-parameter tuning with ray tune	0.9110	Random search	Introducing the activation functions as hyper-parameters	SGD
4 hyper-parameter tuning with ray tune	0.9155	Random search	Change the optimizer	Adam
5 hyper-parameter tuning with ray tune	0.9222	Optuna	Change the search algorithm	Adam
6 hyper-parameter tuning with ray tune	0.9054	Optuna	add semi.CV -Changing the score criterion	Adam
7 hyper-parameter tuning with ray tune	0.91052	Optuna	Changing the max possible iterations	Adam
8 hyper-parameter tuning with ray tune	0.9074	Optuna	Changing the score criterion	Adam
9 hyper-parameter tuning with ray tune	0.9235	Optuna	Limit the search space & New neural network model training with the optimized hyper-parameters	Adam

In addition, referring to the report I wrote earlier in the applied machine learning course about the HIV dataset is not harmful. My meter in that project was balanced accuracy. In the following table, you can have a better comparison of the final accuracy of these two data sets. [7]

	Best algorithm	Dimension reduction	Search Algorithm	Balanced accuracy	Sensitivity	Specificity
1	KNN	PCA	Gridsearchcv	0.7326	0.4861	0.9791
				Accuracy= 0.96131		
2	Neural Network	featurewiz	Optuna	0.8807	0.7872	0.9741
				Accuracy= 0.97203		

I would have loved to partition the data set like the article FunQG: Molecular Representation Learning Via Quotient Graphs using Scaffold Split. Still, the little time I had on this project prevented me. I hope to have this opportunity in the future.[8]

NOTES

¹Matplotlib library has been used in most visualizations of this report, either directly or indirectly. [9]

²I am not sure that using `stratify` in `train_test_split` [10] does not cause any information leakage. However, this is a standard method, and I will use this method in this project.

³Cumulative Distribution Function

⁴Receiver Operating characteristic curve

⁵Area under the (ROC) Curve

⁶Mean Squared Error

⁷In Tune, some hyper-parameter optimization algorithms are written as “scheduling algorithms”. These Trial Schedulers can early terminate bad trials, pause trials, clone trials, and alter hyper-parameters of a running trial.[11]

ACKNOWLEDGEMENTS

I would like to express my special gratitude to my dear professors, **Dr.Zahra Taheri** and **Dr.Bijan Ahmadi**. Both of these dignitaries gave me the golden opportunity to do this wonderful project on the topic of hyper-parameter optimization, which also helped me do a lot of research that has led to learning fascinating and valuable information in this field.

It is also appropriate to mention **Professor Andrew Ng**, who introduced me to the structure of neural networks through his online training courses. In addition, let me say that I learned to work with the PyTorch library through Mr. *Patrick Loeber*’s YouTube channel.

I would like to mention that I consulted with my good friends *Dr. Behrad Taghi Beiglo* and *Kian Adib* in this project. I should also thank my good teammate, Mr. *Mahmoud Hashempour*.

REFERENCES

- [1] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.
- [2] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, et al. Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling*, 59(8):3370–3388, 2019.
- [3] AutoViML. Autoviml/featurewiz: Use advanced feature engineering strategies and select best features from your data set with a single line of code.
- [4] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. ROC Analysis in Pattern Recognition.
- [5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [6] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.
- [7] Arash Sajjadi. *HIV project report*. 2022.
- [8] Hossein Hajiabolhassan, Zahra Taheri, Ali Hojatnia, and Yavar Taheri Yeganeh. Funqg: Molecular representation learning via quotient graphs, 2022.
- [9] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
- [10] 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*, 12:2825–2830, 2011.
- [11] trial schedulers (tune.schedulers) - ray 1.13.0.