Graph Neural Network training

Álvaro Román Gómez

5/30/23

Table of contents

1 MODEL HYPERPARAMETER OPTIMIZATION USING OPTUNA 2

```
import numpy as np
import pandas as pd

import deepchem as dc
import tensorflow as tf
from deepchem.models.graph_models import GraphConvModel

import optuna
from optuna.samplers import TPESampler
from optuna.visualization import plot_optimization_history, plot_param_importances
```

```
input_path = "../data/processed/smiles/"

traing_path = input_path + "train_data/"
test_path = input_path + "test_data/"
output_path = ""
```

```
# FILES
molecules_train_file = "smiles_activity_training.csv"
molecules_test_file = "smiles_activity_test.csv"
```

```
molecules_train = pd.read_csv(input_path + molecules_train_file)
molecules_test = pd.read_csv(input_path + molecules_test_file)
```

```
smiles_train = molecules_train["canonical_smiles"]
smiles_test = molecules_test["canonical_smiles"]
```

```
activity_train = molecules_train["activity"]
activity_test = molecules_test["activity"]
```

```
featurizer = dc.feat.ConvMolFeaturizer()
```

```
molecules_graphs_train = featurizer.featurize(smiles_train)
molecules_graphs_test = featurizer.featurize(smiles_test)
```

```
train_dataset = dc.data.NumpyDataset(X=molecules_graphs_train, y=activity_train)
test_dataset = dc.data.NumpyDataset(X=molecules_graphs_test, y=activity_test)
```

1 MODEL HYPERPARAMETER OPTIMIZATION USING OPTUNA

```
def objective(trial):
    dropout = trial.suggest_float("dropout", 0.2, 0.5, step=0.1)
    graph_conv_layers = trial.suggest_categorical(
        "graph_conv_layers",
        [
            [128, 128, 64, 32, 32, 16, 16],
            [128, 128, 64, 32, 32, 16, 8, 8, 4],
            [256, 128, 64, 32, 16],
            [64, 64, 64, 64, 64, 64, 64],
        ],
    )
    dense_layer_size = trial.suggest_categorical(
        "dense_layer_size", [200, 128, 64, 32, 16]
    )
    batch_size = trial.suggest_categorical("batch_size", [64, 32, 16, 8])
    learning_rate = trial.suggest_categorical(
        "learning_rate", [0.2, 0.1, 0.01, 0.001, 0.0001, 0.00001]
    )
    use_queue = trial.suggest_categorical("use_queue", [False, True])
    epochs = trial.suggest_categorical("epochs", [10, 50, 100, 200])
```

```
    model = GraphConvModel(
        n_tasks=1,
        mode="classification",
        dropout=dropout,
        graph_conv_layers=graph_conv_layers,
        dense_layer_size=dense_layer_size,
        batch_size=batch_size,
        learning_rate=learning_rate,
        use_queue=use_queue,
    )

    eval_results = model.evaluate(
        test_dataset, [dc.metrics.Metric(dc.metrics.roc_auc_score)]
    )
    auc_score = eval_results["roc_auc_score"]
    return auc_score
```

```
sampler = TPESampler(seed=10)
study = optuna.create_study(direction="maximize", sampler=sampler)
study.optimize(objective, n_trials=100)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function KerasModel._compute_model at 0x159
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function KerasModel._compute_model at 0x159
```

```
plot_optimization_history(study)
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

```
plot_param_importances(study)
```

```
Unable to display output for mime type(s): text/html
```

```
best_model = GraphConvModel(
    n_tasks=1,
    mode="classification",
    dropout=study.best_params["dropout"],
    graph_conv_layers=study.best_params["graph_conv_layers"],
    dense_layer_size=study.best_params["dense_layer_size"],
    batch_size=study.best_params["batch_size"],
    learning_rate=study.best_params["learning_rate"],
    use_queue=study.best_params["use_queue"],
)
```

```
best_model.fit(train_dataset, nb_epoch=study.best_params["epochs"])
```

```
0.9526234436035156
```

```
best_model.evaluate(test_dataset, [dc.metrics.Metric(dc.metrics.roc_auc_score)])
```

```
{'roc_auc_score': 0.13523573200992556}
```