NNI使用教程

一、阅读官方文档了解NNI基本的使用方法

NNI快速上手

二、使用示例——在tf2-gnn上搭建NNI框架

0、文件结构一览

- tf2_gnn
  - cli
    - train.py
    - search_space.json
    - config.yml
  - cli_utils
    - training_utils.py
    - default_hypers
      - GraphBinaryClassification_RGIN.json

1、安装

在 `python >= 3.5` 的环境中，运行 `pip install nni` 完成安装。

2、定义搜索空间

在工作目录下添加一个search_space.json文件，写入要配置的参数。

```
{
    "gnn_hidden_dim":{ "_type": "choice", "_value": [4,8,16]},
    "gnn_num_layers": { "_type": "choice", "_value": [2,4,8] },
    "graph_aggregation_num_heads":{ "_type": "choice", "_value": [4,8,16,32]
},
    "graph_aggregation_hidden_layers":{ "_type": "choice", "_value":
[32,64,128,256] },
    "graph_aggregation_dropout_rate":{ "_type": "choice", "_value": [
0.1,0.2,0.5] },
    "learning_rate": { "_type": "choice", "_value": [0.01,0.001,0.0001] }
}
```

3、在代码中添加NNI配置

1. 修改cli/train.py，添加如下代码。

```
import nni
import os
import json
# 使用nni获取参数
def nni_config():
    # 因为tf2-gnn的代码会从json文件中读取默认参数，所以我们只需要修改json文件即可完成超参的设置
    # 比如，我们的任务是GraphBinaryClassification，使用的网络是RGIN，那么我们要修改的文件就是GraphBinaryClassification_RGIN.json
    json_file = "/home/wangzheng/nisl/pdx/tf2-gnn-master/tf2-gnn-master/tf2_gnn/cli_utils/default_hypers/GraphBinaryClassification_RGIN.json"
    # 设置一些默认的参数（如果全部由NNI获取则不需要这一步）
    params = {"model_params":{\
    "gnn_aggregation_function": "sum",\
    "gnn_message_activation_function": "relu",\
    "gnn_hidden_dim": 16
    }}
    # 通过NNI接口获取一组由调优算法选择的参数
    try:
        model_params = nni.get_next_parameter()
        params["model_params"].update(model_params)
        # 因为"graph_aggregation_hidden_layers"的参数类型是一个数组，所以需要作特殊处理（如果在search_space.json中已经将其设为数组则不需要这一步）
        if "graph_aggregation_hidden_layers" in model_params.keys():
            params["model_params"]["graph_aggregation_hidden_layers"] = [model_params["graph_aggregation_hidden_layers"]]
    except Warning:
        pass
    # 将新的参数写入参数文件
    with open(json_file, 'w') as f:
        json.dump(params, f)
        print("参数配置", params)
if __name__ == "__main__":
    nni_config()
```

2. 修改cli_utils/training_utils.py，添加如下代码。

```
def train:
    # 找到这行代码，在下一行添加nni的代码
    valid_metric, valid_metric_string =
model.compute_epoch_metrics(valid_results)
    # 获取中间结果
    nni.report_intermediate_result(-valid_metric)
def run_train_from_args(...):
    # 找到这行代码，在下一行添加nni的代码
    test_metric, test_metric_string =
model.compute_epoch_metrics(test_results)
    # 获取最终结果
    nni.report_final_result(-test_metric)
```

4、写入配置文件

定义 YAML 格式的`配置`文件，其中声明了搜索空间和 Trial 文件的`路径`。 它还提供其他信息，例如调整算法，最大 Trial 运行次数和最大持续时间的参数。

```
authorName: NNI Example
experimentName: tf2-nn TF v2.x
trialConcurrency: 1
maxExecDuration: 3h  # 执行时间上限
maxTrialNum: 50  # 调参次数上限
trainingServicePlatform: local
searchSpacePath: search_space.json  # 搜索空间的存放路径
useAnnotation: false
tuner:
    builtinTunerName: TPE   # choices: TPE, Random, Anneal, Evolution,
BatchTuner, MetisTuner,
                            #          GPTuner, SMAC (SMAC should be installed
through nnictl)
    classArgs:
        optimize_mode: maximize  # choices: maximize, minimize
    gpuIndices: "4"  # 指定用于运行优化器的GPU
trial:
    command: python3 train.py RGIN GraphBinaryClassification
../../../../404/cdfg  # 指定要在框架中执行的命令
    codeDir: .
    gpuNum: 1
logDir: /home/wangzheng/nisl/pdx/nni/experiment  # 指定NNI日志输出路径
localConfig:
    gpuIndices: "4"  # 指定用于运行代码的GPU
    useActiveGpu: true
```

5、运行NNI

默认端口为8080，如被占用可手动指定。

```
nnictl create --config config.yml --port 8080
```

在命令行中等待输出 `INFO: Successfully started experiment!`。 此消息表明 Experiment 已成功启动。

6、nnictl常用命令

查看实验情况：`nnictl trial ls`

查看错误报告：`nnictl log stderr`

停止当前trial：`nnictl trial kill [experiment_id] -T [trial_id]`

停止实验：`nnictl stop`

导出结果：`nnictl experiment export [experiment_id] --filename [file_path] --type json`