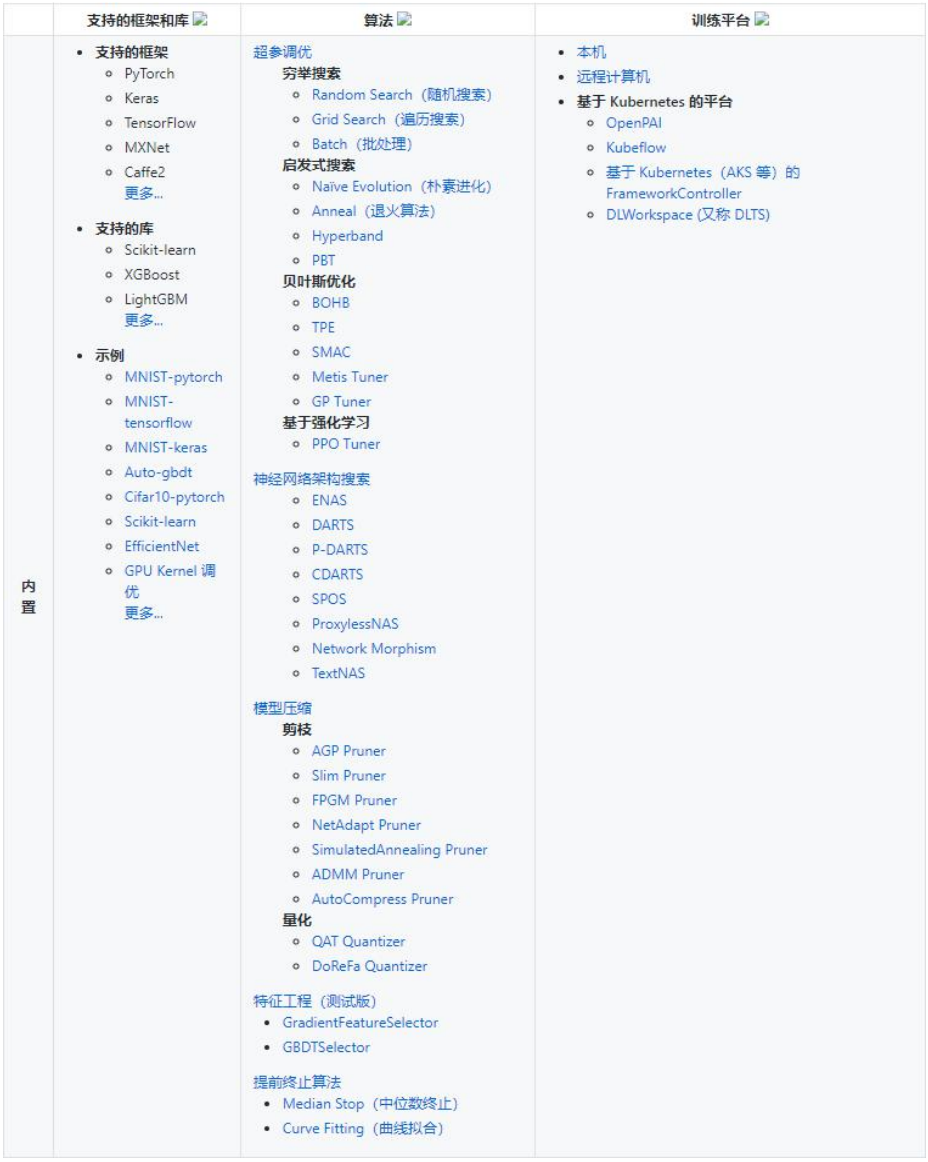
云脑自动机器学习-NNI 使用指南

云脑自动机器学习-NNI 使用指南

一、概述

NNI (Neural Network Intelligence) Github link 是微软开源的一个 autoML 工具包，帮助用户自动地进行特征工程，神经网络架构搜索，超参调优以及模型压缩（如图 1 所示）。NNI 通过多种调优的算法来搜索最好的神经网络结构和超参，拥有可视化界面并支持单机、本地多机、云等不同的运行环境。鹏城云脑现已集成 NNI 功能，在 dgx 和 debug 模式下支持任务使用自动机器学习。该功能需要用户了解 NNI 的基本使用方法，可参考文档。所有的示例代码可在官方 github 的 nni/examples 中查看。



	支持的框架和库	算法	训练平台
内置	• **支持的框架** ◦ PyTorch ◦ Keras ◦ TensorFlow ◦ MXNet ◦ Caffe2 更多... • **支持的库** ◦ Scikit-learn ◦ XGBoost ◦ LightGBM 更多... • **示例** ◦ MNIST-pytorch ◦ MNIST-tensorflow ◦ MNIST-keras ◦ Auto-gbdt ◦ Cifar10-pytorch ◦ Scikit-learn ◦ EfficientNet ◦ GPU Kernel 调优 更多...	超参调优 **穷举搜索** ◦ Random Search（随机搜索） ◦ Grid Search（遍历搜索） ◦ Batch（批处理） **启发式搜索** ◦ Naïve Evolution（朴素进化） ◦ Anneal（退火算法） ◦ Hyperband ◦ PBT **贝叶斯优化** ◦ BOHB ◦ TPE ◦ SMAC ◦ Metis Tuner ◦ GP Tuner **基于强化学习** ◦ PPO Tuner 神经网络架构搜索 ◦ ENAS ◦ DARTS ◦ P-DARTS ◦ CDARTS ◦ SPOS ◦ ProxylessNAS ◦ Network Morphism ◦ TextNAS 模型压缩 **剪枝** ◦ AGP Pruner ◦ Slim Pruner ◦ FPGM Pruner ◦ NetAdapt Pruner ◦ SimulatedAnnealing Pruner ◦ ADMM Pruner ◦ AutoCompress Pruner **量化** ◦ QAT Quantizer ◦ DoReFa Quantizer 特征工程（测试版） • GradientFeatureSelector • GBDTSelector 提前终止算法 • Median Stop（中位数终止） • Curve Fitting（曲线拟合）	• 本机 • 远程计算机 • **基于 Kubernetes 的平台** ◦ OpenPAI ◦ Kubeflow ◦ 基于 Kubernetes（AKS 等）的 FrameworkController ◦ DLWorkspace (又称 DLTS)

图 1

二、提交任务界面

NNI 功能支持在 DGX 和 DEBUG 模式下使用。

DEBUG：编写保存搜索空间文件 search_space.json；修改训练脚本从 NNI 获取超参，并返回 NNI 最终结果；编写保存配置文件 config.yml；在 Jupyterlab 代码编辑器调试程序。限时 2 小时。

DGX：在 DEBUG 模式下调试好程序后，即可切换至 DGX 模式正式运行程序。不限时长。

提交使用 NNI 功能的 DEBUG 任务详细介绍

1. 镜像选择

选镜像时输入 192.168.202.74:5000/msranni/nni-modified:v1.7，该镜像是从开源版本 v1.7 中有所修改适配到云脑平台的，镜像描述如下：

> CUDA 9.0, CuDNN 7.0
> python 3.5
> numpy 1.14.3,scipy 1.1.0
> TensorFlow-gpu 1.10.0
> Keras 2.1.6
> PyTorch 0.4.1
> scikit-learn 0.20.0
> pandas 0.23.4
> lightgbm 2.2.2
> NNI modified by yunnao from the official version v1.7

如果想用自己的镜像，也可以通过源码安装 nni。如果出现失败，原因可能是网络问题，请重试。

```
git clone -b v1.7 git@git.pcl.ac.cn:pengfang/nni.git
cd nni
bash install.sh
```

安装原版 nni 的命令如下，pip install nni，但是可能存在兼容问题报错。

自己制作镜像还需要：1.安装 ssh 服务端和客户端；2.echo "root:abc123" | chpasswd ，将 root 密码修改为 abc123（用户可自定义），定为初始密码；3.将/etc/ssh/sshd_config 文件中的 PermitRootLogin 改成 yes，允许 root 登录；4.子任务名和 ip 的匹配关系将失效，在 remote 模式下的 config.yml 中 machineList/ip 只能填写真实 ip，可通过在/etc/hosts 文件中自动抓取 ip 地址实现。

2. 在编辑子任务时选中 NNI 功能

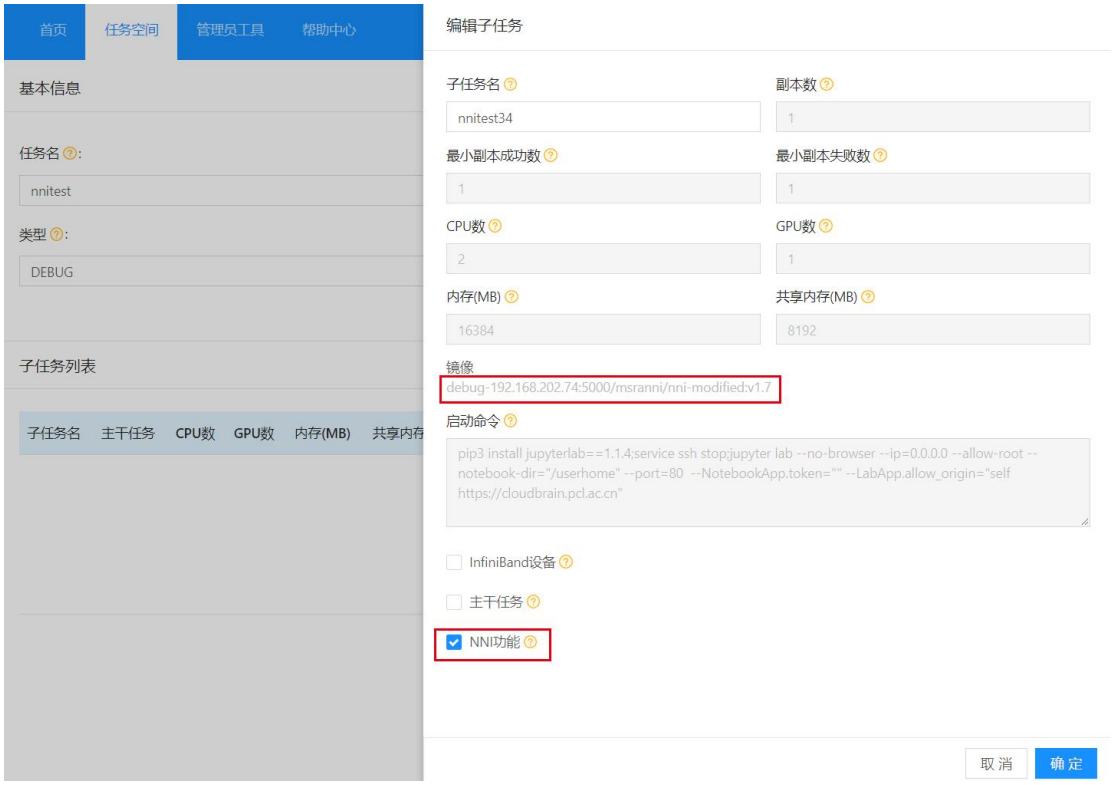


图 2

3. 进入调试界面

NNI 搜索超参的流程如下：

输入：搜索空间，Trial 代码，配置文件

输出：一组最佳的超参配置

```
For t = 0, 1, 2, ..., maxTrialNum,
    hyperparameter = 从搜索空间选择一组参数
    final result = run_trial_and_evaluate(hyperparameter)
    返回最终结果给 NNI
    If 时间达到上限，
        停止实验
    return 最好的实验结果
```

参考文档，编写保存搜索空间文件 search_space.json；编写保存配置文件 config.yml；修改训练脚本 mnist.py 从 NNI 获取超参，并返回 NNI 最终结果；在 Jupyterlab 代码编辑器调试程序。

示例如下：code link

search_space.json

```
{
    "dropout_rate":{"_type":"uniform","_value":[0.5, 0.9]},
    "conv_size":{"_type":"choice","_value":[3,5,7]},
    "channel_1_num":{"_type":"choice","_value":[16,32,64]},
    "channel_2_num":{"_type":"choice","_value":[32,64,128]},
    "hidden_size":{"_type":"choice","_value":[124, 512, 1024]},
    "learning_rate":{"_type":"choice","_value":[0.0001, 0.001, 0.01, 0.1]}
}
```

config.yml

```
authorName: default
experimentName: example_mnist
trialConcurrency: 1     #同时运行的 trial 数，一般与(GPU 总数/每个 trial 的 GPU 数)相同
maxExecDuration: 1h
maxTrialNum: 20
trainingServicePlatform: local        #云脑上支持的训练平台: local, remote
searchSpacePath: search_space.json
useAnnotation: false                  #若选择注释模式则选 true
logDir: '/userhome/nni-experiments'   #默认为'/root/nni/experiments'，不方便查看
tuner:
  #choice: TPE, Random, Anneal, Evolution, BatchTuner, MetisTuner, GPTuner
  #SMAC (SMAC should be installed through nnictl)
  builtinTunerName: TPE
  classArgs:
    optimize_mode: maximize           #choice: maximize, minimize
trial:
  command: python3 mnist.py
  codeDir: .
  gpuNum: 1                           #每个 trial 所用的 gpu 数目
```

mnist.py 伪代码(增加三至四行代码)

```
+  import nni
def main(params):
    # Import data ...
    # Build the model ...
    test_acc = 0.0
    with tf.Session() as sess:
        #train the model ...
        if i % 100 == 0:
            test_acc = mnist_network.accuracy.eval(feed_dict={...})
+           nni.report_intermediate_result(test_acc)
    test_acc = mnist_network.accuracy.eval(feed_dict={...})
```

```
+   nni.report_final_result(test_acc)

def get_params():
    parser = argparse.ArgumentParser()
    parser.add_argument("--dropout_rate", type=float, default=0.5, help="dropout
rate")
    parser.add_argument("--channel_1_num", type=int, default=32)
    ...
    args, _ = parser.parse_known_args()
    return args

if __name__ == '__main__':
    # get parameters form tuner
+   tuner_params = nni.get_next_parameter()
    logger.debug(tuner_params)
    params = vars(get_params())
+   params.update(tuner_params)
    main(params)
```

4. 启动实验

cd code_dir; nnictl create --config config.yml -f

或 cd code_dir; nnictl create -c config.yml -f

（-f 代表在前台运行）

ERROR 定位：

如果 trial 长时间 waiting，可能是示例代码 mnist.py 的数据集地址中没有数据集，从网络中下载经常会下载不到。

常用命令链接：

nnictl stop [id]，停止实验，如果只有一个实验在运行，就可以不加 id 默认停止该实验

nnictl view [id] --port [port] 查看一个已经停止的实验

nnictl log stdout [id] 打印日志

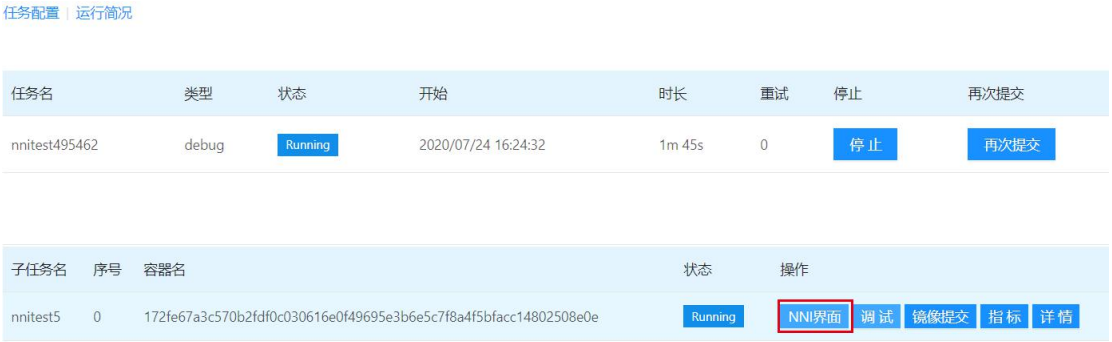
nnictl update searchspace [id] -f [filename] 更新搜索空间

nnictl update concurrency [id] -v [value] 更新 trial 同步数

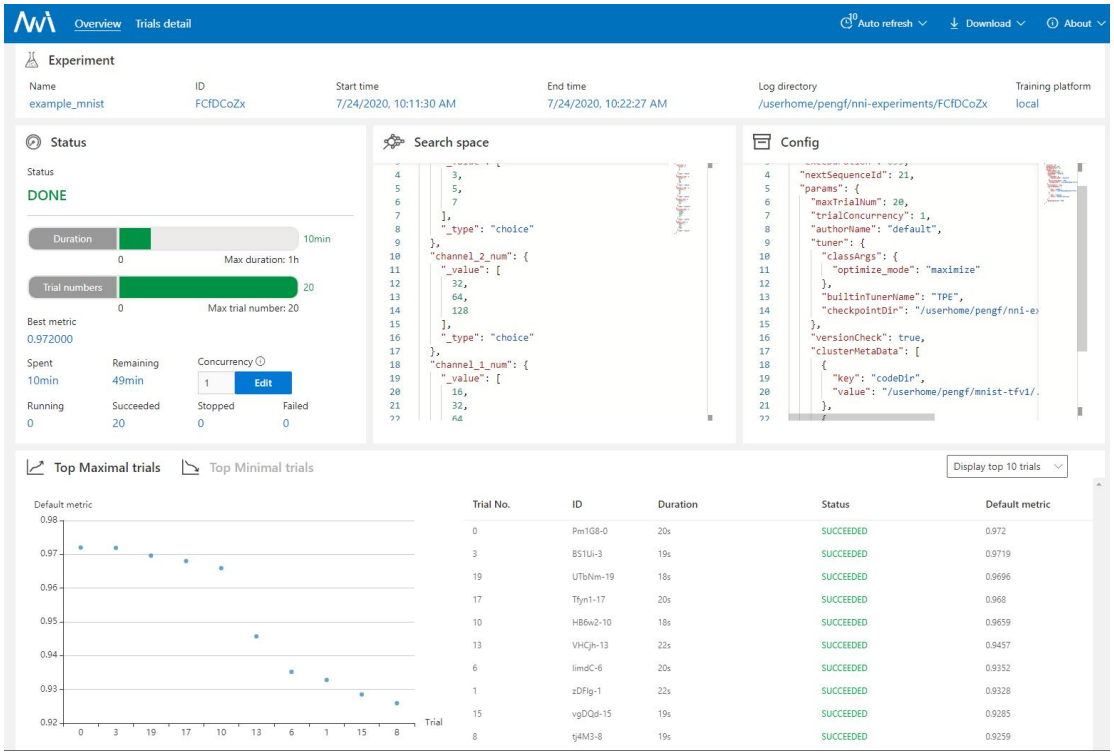
nnictl update duration [id] -v [value] 更新运行时长

5. 在 web 上查看实验

点击“NNI”界面



NNI 的界面详情



提交使用 NNI 功能的 DGX 任务详细介绍

界面操作与 DEBUG 模式大致相同，选择类型 DGX，输入镜像 192.168.202.74:5000/msranni/nni-modified:v1.7，点选“NNI 功能”，启动命令 cd code_dir; nnictl create -c config.yml -f，提交。（-f 代表在前台运行，若不加则会导致任务直接被终止。）注意：dgx 不会有超时限制，所以在完成任务后一定要手动停止任务。

使用多个本地 GPU 加快速度

如

```
trainingServicePlatform: local

trialConcurrency: 4
```

在任务详情提交处申请 4 块 NVIDIA GPUs，因为一个容器是在一台机器上申请资源，所以相当于多个本地 GPU，通过设置 4 个并发的 Trial 任务，每个 Trial 任务使用 1 块 GPU。

三、多机 remote 模式

涉及到多机多卡实验时，NNI 可以通过 SSH 在多个远程计算机上运行同一个实验，称为 remote 模式，就像一个轻量级的训练平台。在此模式下，每台计算机上的 NNI 版本一致，可以从一台主节点上启动 NNI，并将 Trial 并行调度到远程计算机。可参考文档。

示例：code link

1. 修改 config.yml，增加 machineList 的定义

```
trainingServicePlatform: remote
machineList:  #local 模式下没有
  - ip: major-0  #子任务名-$index
    username: root
    passwd: abc123
    #使用默认端口 22 时，该配置可跳过
    #port: 22
  - ip: woker-0  #子任务名-$index
    username: root
    passwd: abc123
```

这里有两个 pod 容器，ip 地址会在容器启动后从/etc/hosts 里抓取，/etc/hosts 中 ip 会和“子任务名-index”相对应。（注意：后端已自动化，不需用户操作。）使用默认 22 端口。密码默认 abc123，没有设置免密登录，如果需要免密登录，请参考鹏城云脑快速使用指南。ssh 服务在镜像中已经装好，而且配置了自启动，如果发现 ssh 连不通，尝试使用命令 service ssh start 开启服务，或者检查子任务名和 config.yml 中的 machineList/ip 配置是否正确对应。

2. 提交任务界面，选择一个主干任务，副本为 1；一个或多个非主干任务（只需使用 sleep 命令占住即可），副本数任意；所有任务都勾选 NNI 功能；主干任务

的启动命令前加一个等待时间，要等所有的非主干任务都启动起来后再执行 nnictl create 命令。实测跑相同的任务，remote 模式比单机多卡模式慢 28%。

3. 任务详情中，只有主任务的界面可以点开，其余任务的按钮为灰色。因为 NNI 没有在其他非主干远程计算机上设置界面。

四、注释模式（推荐使用）

不修改原先的代码逻辑，通过 Annotation，只需要在代码中加入一些注释字符串，就能启用 NNI。

参考文档。

示例：code link

1. train 代码中修改，在相关变量、函数、result 出现的上一行加注释。

> 一共四种类型的 Annotations：
>
> 1.变量：单选、随机数、均匀分布、正态分布等
>
> ```
> @nni.variable(nni.choice(option1,option2,...,optionN),name=variable)
> @nni.variable(nni.randint(lower, upper),name=variable)
> @nni.variable(nni.uniform(low, high),name=variable)
> ```

```
@nni.variable(nni.normal(mu, sigma),name=variable)（正态分布）等
例如：
'''@nni.variable(nni.choice(0.1, 0.01, 0.001), name=learning_rate)'''
learning_rate = 0.1
2.函数：池化、激活函数、损失函数、优化器等
例如：
'''@nni.function_choice(max_pool(hidden_layer, pool_size), avg_pool(hidden_layer,
 pool_size), name=max_pool)'''
h_pooling = max_pool(hidden_layer, pool_size)
3.中间结果（）
'''@nni.report_intermediate_result(metrics)'''
4.最终结果
'''@nni.report_final_result(metrics)'''
```

2. 修改 config.yml

useAnnotation: true

注释一行#searchSpacePath: search_space.json