机器学习平台漫谈

gaocegege 23个月前 机器学习

随着深度学习的兴起，机器学习在最近几年以星火燎原之势席卷了整个科技行业。而在整个机器学习的工作流中，模型训练的代码只是其中的一小部分。除此之外，训练任务的监控，日志的回收，超参数的选择与优化，模型的发布与集成，数据清洗，特征提取等等，都是流程中不可或缺的部分。因此，有一些工具和公司的产品，致力于为机器学习从业者提供一个统一的平台，帮助用户更好地完成其机器学习业务的落地。这篇文章是关于机器学习平台产品的分析对比，由于利益相关性只放出国外的产品，如有遗漏或错误还请指出。

RiseML - Machine Learning Platform for Kubernetes

RiseML 是一个基于 Kubernetes 的机器学习平台，是一家在德国的创业公司的产品。RiseML 的产品特点在于其简约的 API，以及简易的安装方式。RiseML 支持通过 helm chart 进行安装，相比于 Kubernetes 原生的 API，RiseML 提供了更 high level 的 API 定义，有着更高一级的抽象。

```
project: ai-toaster
train:
  framework: tensorflow
  resources:
    cpus: 2
    mem: 16384
    gpus: 4
  tensorflow:
    version: 1.2.1-gpu
    distributed:
      worker: 3
      ps:
        count: 2
        resources:
          cpus: 2
          mem: 16384
          gpus: 0
  run: >-
```

```
    python train_model.py --num-layers {{ num-layers }}
                          --learning-rate {{ learning-rate }}
                          --training-data /data/my-project
```

以分布式的 TensorFlow 训练任务为例，RiseML 只需要指定参数服务器和 worker 的资源情况以及实例个数即可。这样的设计对于平台的初见者非常友好，但同时也会导致一些比较特化的需求难以得到很好地支持。比如 ML 工程师想控制任务的重启逻辑，以及为容器开放特定的端口，绑定数据卷时，就难以用这一套高级的 API 来完成。易用性与灵活性之间的 trade-off 在机器学习平台系统的设计中也值得考虑。

除此之外，RiseML 支持超参数训练。其 API 同样十分简洁：

```
project: ai-toaster
train:
  resources:
    cpus: 2
    mem: 4096
    gpus: 2
  parameters:
    lr:
      - 0.0001
      - 0.001
    lr-decay:
      - 1e-6
      - 1e-7
    epochs:
      - 20
      - 30
  concurrency: 2
  image:
    name: nvidia/cuda:8.0-cudnn7-runtime
  run: >-
    python train_model.py --num-layers {{num-layers}}
                          --learning-rate {{learning-rate}}
                          --training-data /data/ai-toaster
```

在示例中，RiseML 并没有指定超参数的搜索算法，所以对于其的支持相对而言也并不是十分完善的。RiseML 目前的工作主要在训练方面。其提供了一个 CLI，可以发起训练/超参数训练任务以及查看日志和状态等等。而对于推理服务的支持目前还未可见。

FloydHub - Deep Learning Platform for Productive Data Science Teams

FloydHub 是同类产品里比较有名的存在了，因为它收到了来自 Y Combinator，真格基金等知名的孵化器和投资机构的投资。FloydHub 有几个核心概念：项目，工作空间，任务，数据集，环境和输出。其中项目是一群目标一样的工作空间和任务的集合。工作空间是为了模型开发而存在的交互式的交互式开发环境，是基于 JupiterHub 实现的。而任务就是一次模型训练代码的运行，在运行的过程中，任务会拉取代码以及数据集，并且配置相应的环境。数据集是用户上传的数据集合，在 FloydHub 中支持版本化的数据集。之所以分离数据和代码，是因为数据不是经常变动的而代码则是会经常因为调参等等原因发生变化的。这样的分离可以节约数据上传和准备的时间，也利于数据的共享。环境是指代码运行的环境，比如需要的机器学习框架的版本，以及是否需要 GPU 等；值得一提的是 FloydHub 也是基于 Docker 的。最后是输出，输出是在一次任务中用户希望保留的日志，文件或者数据等内容。一个典型的用例是保存 checkpoint。因为是基于 Docker 进行的训练，最后训练任务的容器会要被清理的，这时需要通过输出这一概念来持久化对应的内容。

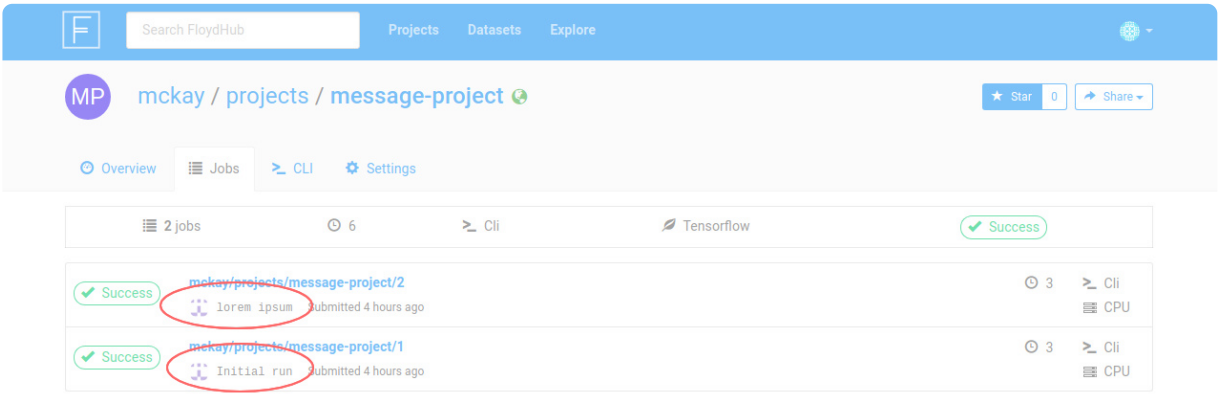
FloydHub 已经实现了较好的流水线支持，以下面的例子来说明：

```
env: pytorch-0.4

task:
  train:
    machine: gpu2
    description: train with lstm
    input:
      - source: foo/datasets/wine-reviews/1
        destination: input
    command: train.py /floyd/input/input

  test:
    machine: gpu
    description: evaluate model1
    input:
      - source: foo/projects/nlp/output
        destination: model
      - source: foo/datasets/wine-reviews-test/1
        destination: test
    command: test.py --model /floyd/input/model --data /floyd/input/test

  serve:
    machine: cpu
    mode: serve
    input:
```

```
      - source: foo/projects/nlp/output
        destination: model
```

与 RiseML 类似，FloydHub 也设计了自己的 high level API，同时有流水线的概念。关于 API 的易用性以及灵活性的问题之前已经讨论过，这里不再多说。而流水线的支持应该是所有的机器学习平台都需要支持的功能，FloydHub 相对是要完善一些。

除此之外，FloydHub 有一个简洁而且与 GitHub 类似的 UI，这在易用性上是非常巨大的优势。



Comet - Supercharge Machine Learning

Comet 是与上面的两个产品的做法不太一样的产品，其对用户的训练代码是有侵入性的。举例说明：

```
from comet_ml import Experiment
experiment = Experiment(api_key="YOUR_API_KEY",
                        project_name="my project name",
                        auto_param_logging=False)
batch_size = 128
num_classes = 10
epochs = 20

params={
    "batch_size":batch_size,
    "epochs":epochs,
    "num_classes":num_classes}

experiment.log_multiple_params(params)
```

Comet 目前主要关注在训练和超参数训练这边，它现在主要的使用方式是 Python SDK。我觉得这样有侵入性的使用方式，是不让人喜欢的。其 Python SDK 最大的抽象是 Experiment，对应一次训练。在某些框架下，需要用户将参数告诉 Comet，以便在 Comet 的 Web 端显示对应的参数值。其对超参数训练的支持是引入了一个新的抽象，Optimizer，可以将其看做是一个基于 cloud 的参数搜索的策略。至于参数搜索的策略到底是什么，文档中并未表述。通过这一抽象可以得到下一次尝试的超参数，并且训练模型记录模型的准确性指标。随后在 Comet 的 Web 端可以得到一个不同超参数下准确性的报表。

mlflow - Open Source Framework for the Complete Machine Learning Lifecycle

mlflow 可以被当做是加强版和开源版的 Comet，它们两个的实现思路是一致的，就是通过 Python API 侵入性地获得一些信息。其核心的概念一共有三个：Project，Tracking，Model。其中 Project 可以当成是类似 Dockerfile 的一种简化的替代产物，其存在的意思是定义了一种项目打包的方式，使得工作可以被完整复现。它支持如下的定义：

```
name: My Project

conda_env: my_env.yaml

entry_points:
  main:
    parameters:
      data_file: path
      regularization: {type: float, default: 0.1}
    command: "python train.py -r {regularization} {data_file}"
  validate:
    parameters:
      data_file: path
    command: "python validate.py {data_file}"
```

其中 conda_env 是配置代码运行环境时的配置文件，entry_points 与 Dockerfile 中的 entrypoint 概念类似。窃以为这种重新造轮子的方式并不可取，而且一定解决地不如 Docker 好。

Model 对应一个训练好的模型。其支持方式与 Project 类似，也存在一个描述文件，如下所示：

```
time_created: 2018-05-25T17:28:53.35

flavors:
  tensorflow:
```

```
    saved_model_dir: estimator
    signature_def_key: predict
```

描述文件中最主要的字段是 `flavors`，可以把它当做转换器的概念，通过 flavor，系统才能知道模型的相关信息，以及需要如何去部署它。以 TensorFlow flavor 为例，它共有两个参数，分别是 `saved_model_dir`，也就是 saved model 所在的目录，和 `signature_def_key`，也就是进行预测的函数签名。在 mlflow 的 TFWrapper 中我们可以看到其对应的处理逻辑：

```
class _TFWrapper(object):
    """
    Wrapper class that creates a predict function such that
    predict(data: pandas.DataFrame) -> pandas.DataFrame
    """
    def __init__(self, saved_model_dir):
        model = Model.load(os.path.join(saved_model_dir, "MLmodel"))
        assert "tensorflow" in model.flavors
        if "signature_def_key" not in model.flavors["tensorflow"]:
            self._signature_def_key = tf.saved_model.signature_constants \
                .DEFAULT_SERVING_SIGNATURE_DEF_KEY
        else:
            self._signature_def_key = model.flavors["tensorflow"]["signature_de
            self._saved_model_dir = model.flavors["tensorflow"]["saved_model_di
     def predict(self, df):
        graph = tf.Graph()
        with tf.Session(graph=graph) as sess:
            meta_graph_def = tf.saved_model.loader.load(sess,
                                                        [tf.saved_model.tag_con
                                                        self._saved_model_dir)
            sig_def = tf.contrib.saved_model.get_signature_def_by_key(meta_grap
            self._signature_def_key)
        ...
```

也就是将对应的参数取出来，然后按照参数的设定去寻找对应的模型目录，然后运行相应的 serving 逻辑。其最大的特色在于定义了一套标准化的方案，使得对模型部署的支持扩展相对比较方便。

最后的概念就是 Tracking，这个与 Comet 的 Experiment 类似，但是支持更多的保存内容，也支持保存到不同的介质中（本地，HTTP server，Databricks Workspace），但核心概念并无二致，因此不再赘述。

Kubeflow - ML stack on Kubernetes

Kubeflow 是在 2017 年底由 Google 开源，Caicloud，Intel，思科，阿里巴巴等等公司共同参与的一个项目，其旨在简化在 Kubernetes 上运维机器学习工作负载的流程。目前其相比于其他的项目，独特之处在于整个生态都是构建在 Kubernetes 之上的，是完完全全 Kubernetes Native 的。Kubeflow 希望能够实现一套流水线，支持从准备数据到模型发布的一整套机器学习端到端的过程。目前支持较好的是训练过程，其实现可参考 Kubeflow 安利：在 Kubernetes 上进行机器学习一文。

其缺点有两个方面，一方面是目前缺乏完整的端到端的系统支持，另一方面是需要用户了解 Kubernetes 相关的知识。针对第一个问题，社区背靠谷歌正在积极地探索完善的解决方案。第二个问题，目前也有一些探索，比如阿里云容器服务组开源的 arena，它实现了一个 CLI，屏蔽了底层的 Kubernetes，Kubeflow 等系统的细节，是与 FloydCLI 类似的思路。

Seldon - Machine Learning Deployment Platform for Enterprise

Seldon 主打的也是基于 Kubernetes 的开源机器学习系统，它在 KubeCon 上有过一个 Talk：Serving ML Models at Scale with Seldon and Kubeflow。它对 Serving 解决地比较好，是通过定义而了一个 CRD，也就是 SeldonDeployment 来实现的，除此之外目前还看不到太多独特之处。值得一提的是，它是用 Java 来实现的，这也是目前我看到的唯一使用 Java 作为主要开发语言的开源机器学习平台系统。

SageMaker - Build, train, and deploy machine learning models at scale

SageMaker 是由传统云计算厂商 AWS 推出的机器学习平台，其对传统的机器学习有内置的算法支持，是一个非常完整的系统，但是对于 TensorFlow 等等的支持并不是很好，有很多额外的限制。相比于之前提到的各个平台，SageMaker 先把脏活累活做好了（对传统机器学习各种模型的支持），这部分工作是需要人力但标准化较简单的，对于 AWS 这样不缺人的大公司而言是很好的选择。

关于作者

高策，上海交通大学软件学院研究生，预计 2019 年毕业，Kubeflow Maintainer。如有问题还请不吝赐教。

欢迎关注我们的 GitHub 以及博客 :)

许可协议

gaocegege

MOS 组的小哥哥

上篇文章

当我们在谈论机器学习平台时，我们在谈论什么

下篇文章

Kubernetes CRD Operator 实现指南