Forge：如何管理你的机器学习实验

专知 前天

【导读】在开始机器学习实验时，大多数人都会经历几个步骤。首先快速写出模型原型和训练脚本。然而几天之后，代码库变得不规则，并且由于代码间混乱的依赖性和没有固定结构，任何修改都需要花费很多时间。因此，我们队模型进行重构，模型的各个部分被包装成单独的，有意义的对象。更进一步，我们经常面临支持多个数据集和模型的变体，其中模型变体之间的差异不仅仅是超参数 - 它们通常在结构上不同并且具有不同的输入或输出。此时，我们开始复制训练脚本以支持模型变体。设置很简单，但维护变成了一场噩梦：代码在不同的文件中，一旦需要对代码修改，所有的文件都要改一遍。

典型的实验结构

而机器学习实验的共性让我们思考，有没有一个通用的工具帮助我们管理实验？ 机器学习实验有以下几个共性：

- 指定数据和相应的超参数。
- 自定义定义模型及其超参数。
- 运行训练脚本并在训练期间保存模型checkpoints和日志。
- 一旦训练收敛，可在另一个脚本或笔记本中加载模型checkpoints以进行全面评估或部署模型。

如果实验确实存在一般结构，那么应该管理实验的工具。. Sacred and artemis非常适合管理配置文件和实验结果;您可以检索实验的配置，但是如果要在笔记本中加载已保存的模型，则需要知道如何使用配置实例化模型。当谈到TensorFlow时，有keras和其他api简化模型构建，拟合和评估。虽然通常很有用，但它们相当繁重，难以访问低级模型功能。

所有这些都表明我们可以从用于管理ML实验的轻量级实验框架中受益。对我来说，它需要满足以下几个要求：

- 它应该需要最少的设置。
- 它必须与tensorflow兼容（这些天我是ML的主要工具）。
- 理想情况下，它应该可用于非张量流模型 - 软件发展迅速，我的下一个项目可能在pytorch中。
- 应单独指定和配置数据集和模型，以便以后可以混合和匹配它们。
- 应该为每个实验存储超参数参数和配置文件，我们可以快速浏览它们而不使用非标准应用程序。
- 加载模开销应该很少，理想情况下不需要触及原始的模型构建代码。指出一个特定的实验就足够了。

Forge

配置

配置可能是Forge中最有用的组件。我们的想法是，我们可以将任意复杂的配置文件指定为python函数，然后我们可以使用forge.load（config_file，* args，* *kwargs*）加载它，*其中config_file是文件系统上的路径。惯例是配置文件应该使用以下签名定义load函数：load（config，* args，** kwargs）。传递给forge.load的参数和kw-args会自动转发到配置文件中的load函数。为什么要通过提供文件路径来加载配置？使代码维护更容易！在训练/实验脚本中编写配置加载代码后，最好不再改写它。但是如何交换配置文件？不触及训练脚本：如果我们将文件路径指定为命令行参数，那么我们可以轻松完成。这是一个例子。假设我们的数据配置文件data_config.py如下：

```
from tensorflow.examples.tutorials.mnist import input_data

def load(config):
    # The `config` argument is here unused, but you can treat it
    # as a dict of keys and values accessible as attributes – it acts
    # like an AttrDict
    dataset = input_data.read_data_sets('.')  # download MNIST
    # to the current working dir and load it
    return dataset
```

模型文件定义了一个简单的单层全连接神经网络，分类损失和model_config.py中的一些指标。 如下所示。

```
import sonnet as snt
import tensorflow as tf

from forge import flags

flags.DEFINE_integer('n_hidden', 128, 'Number of hidden units.')

def process_dataset(x):
    pass
    # this function should return a minibatch, somehow
def load(config, dataset):
    imgs, labels = process_dataset(dataset)

    imgs = snt.BatchFlatten()(imgs)
    mlp = snt.nets.MLP([config.n_hidden, 10])
    logits = mlp(imgs)
    labels = tf.cast(labels, tf.int32)

    # softmax cross-entropy
    loss = tf.reduce_mean(
    tf.nn.sparse_softmax_cross_entropy_with_logits(
    logits=logits, labels=labels))

    # predicted class and accuracy
    pred_class = tf.argmax(logits, -1)
    acc = tf.reduce_mean(tf.to_float(
    tf.equal(tf.to_int32(pred_class), labels)))

    # put here everything that you might want to use later
    # for example when you load the model in a jupyter notebook
    artefacts = {
        'mlp': mlp,
        'logits': logits,
        'loss': loss,
        'pred_class': pred_class,
```

```
        'accuracy': acc
    }

    # put here everything that you'd like to be reported every N training iterations
    # as tensorboard logs AND on the command line
    stats = {'crossentropy': loss, 'accuracy': acc}

    # loss will be minimised with respect to the model parameters
    return loss, stats, artefacts
```

现在我们可以编写一个名为experiment.py的简单脚本来加载一些数据和模型配置文件，并做一些有意思的事。

```
from os import path as osp

import tensorflow as tf

import forge
from forge import flags


# job config
flags.DEFINE_string('data_config',
 'data_config.py', 'Path to a data config file.')
flags.DEFINE_string('model_config',
 'model_config.py', 'Path to a model config file.')
flags.DEFINE_integer('batch_size', 32,
 'Minibatch size used for training.')


config = forge.config()  # parse command-line flags
dataset = forge.load(config.data_config, config)
loss, stats, stuff = forge.load(config.model_config,
 config, dataset)

# ...
# do useful stuff
```

这是最好的部分。你可以运行python experiment.py来运行上面给出的配置文件的脚本。但是如果你想运行一个不同的配置，你可以执行python experiment.py --data_config some / config / file / path.py而不需要触及实验代码。所有这些都非常轻量级，因为配置文件可以返回任何内容并获取您认为必要的任何参数。

智能checkpoints

鉴于我们有非常通用和灵活的配置文件，应该可以抽象出模型加载。例如，如果我们可以加载模型快照而不指向用于训练模型的配置文件（一般来说是在模型代码中），那将是很棒的。我们可以通过使用模型快照存储配置文件来实现。它可以显着简化模型评估和部署，并提高我们实验的可重复性。我们该怎么做呢?

智能checkpoints框架取决于以下文件夹结构。

```
results_dir
|run_name
|1
|2
|...
|<integer>  # number of the current run
```

results_dir是包含潜在许多特定于实验的文件夹的顶级目录，其中每个实验都有一个由run_name表示的单独文件夹。 我们可能想重新运行一个特定的实验，因此，每次运行它时，forge都会创建一个文件夹，其名称是一个整数 - 这个运行的编号。 它从一开始，每次开始同一个实验的新运行时都会递增。 我们也可以通过传递标志来恢复最后一次，而不是开始新的运行。 在这种情况下，我们不为其创建新文件夹，但使用编号最大的文件夹并加载最新的模型快照。

首先，我们需要导入forge.experiment_tools并定义以下标志。

```
from os import path as osp
from forge import experiment_tools as fet

flags.DEFINE_string('results_dir', '../checkpoints',
 'Top directory for all experimental results.')
flags.DEFINE_string('run_name', 'test_run',
 'Name of this job. Results will be stored in a corresponding folder.')
flags.DEFINE_boolean('resume', False,
 'Tries to resume a job if True.')
```

然后我们可以解析标志并初始化我们的checkpoints。

```
config = forge.config()  # parse flags
# initialize smart checkpoint
logdir = osp.join(config.results_dir, config.run_name)
logdir, resume_checkpoint = fet.init_checkpoint(logdir,
 config.data_config, config.model_config, config.resume)
```

fet.init_checkpoint做了一些有用的事情：

- 创建上面提到的目录结构。
- 将数据和模型配置文件复制到checkpoints文件夹。
- 在flags.json中存储所有配置标志和当前git提交的哈希（如果我们在git repo中，对于可重复性非常有用），或者如果restore为True则恢复标志。
- 弄清楚是否存在应加载的模型快照文件。

logdir是我们的checkpoint文件夹的路径，并且计算结果为results_dir / run_name / <integer>。 如果resume为True，则resume_checkpoint是checkpoints的路径，通常为results_dir / run_name / <integer> /model.ckpt- <maximum global step>，否则为None。

现在我们需要使用logdir和resume_checkpoint来存储任何日志和模型快照。 例如：

```
... # load data/model and do other setup
# Try to restore the model from a checkpoint
saver = tf.train.Saver(max_to_keep=10000)
if resume_checkpoint is not None:
    print "Restoring checkpoint from '{}'".format(resume_checkpoint)
    saver.restore(sess, resume_checkpoint)

...
# somewhere inside the train loop
saver.save(sess, checkpoint_name, global_step=train_itr)
...
```

如果我们想在另一个脚本eval.py中加载我们的模型快照，那么我们可以非常直接的这样做。

```
import tensorflow as tf
from forge import load_from_checkpoint

checkpoint_dir = '../checkpoints/mnist/1'
checkpoint_iter = int(1e4)

# `data` contains any outputs of the data config file
# `model` contains any outputs of the model config file
data, model, restore_func = load_from_checkpoint(
checkpoint_dir, checkpoint_iter)

# Calling `restore_func` restores all model parameters
sess = tf.Session()
restore_func(sess)

... # do exciting stuff with the model
```

原文链接：

https://akosiorek.github.io/ml/2018/11/28/forge.html

Github地址：

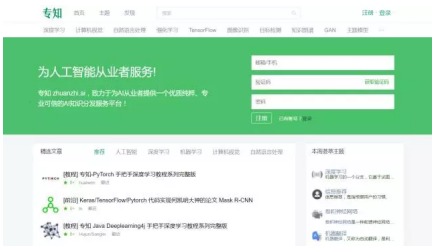
https://github.com/akosiorek/forge

-END-

专·知

人工智能领域26个主题知识资料全集获取与加入专知人工智能服务群: 欢迎微信扫一扫加入专知人工智能知识星球群，获取专业知识教程视频资料和与专家交流咨询！

PC登录www.zhuanzhi.ai或者点击阅读原文，可以获取更多AI知识资料！



加入专知主题群（请备注主题类型：AI、NLP、CV、 KG等）可以其他同行一起交流~ 请加专知小助手微信（扫一扫如下二维码添加），

AI 项目技术 & 商务合作：bd@zhuanzhi.ai, 或扫描上面二维码联系！

请关注专知公众号，获取人工智能的专业知识！

点击“阅读原文”，使用专知

阅读原文