浙江大学

本科周报告

姓　　名：	赖梓林
学　　院：	计算机科学与技术
系：	计算机科学与技术
专　　业：	计算机科学与技术
学　　号：	3170104684

目录

摘要

报告简单综合了这一周关于 PyTorch 的学习内容以及单颗粒分析技术中二维图像处理相关内容。 以及 TensorFlow 基本相关知识，概念以及单颗粒分析中 Topaz 学习方法，Topaz 颗粒管道的相关内容

二、正文：

1. PyTorch 基础知识：

1.1. 张量和梯度：

1.2.数据集准备过程：

1.3.模型训练过程：

1.4.超参数优化：

2. TensorFlow 相关知识：

2.1. 计算图(Computation Graph)：

之前小节中提到，PyTorch 得益于它的动态计算图机制，而 TensorFlow 采用的是静态的计算图机制。在 PyTorch 中，计算图的构建和运算同时进行，更为灵活，适用于处理需要经常更改部分网络结构的问题。而 TensorFlow 中，计算图是静态的，必须先构建好计算图再进行运算，虽然相比之下无法灵活地更改结构，但在效率上有更好的表现。

定义中，计算图是一个有向图，图的每一个节点都对应着一个操作(Operation)或是一个变量(Variable)。变量可以将值赋予操作，操作也可以将输出提供给其他操作。图中的每一个节点都定义了针对于变量的函数。并且每一个操作都相当于可以在那个时间点评估的函数。

在 TensorFlow 中，构建节点以及构建计算图的过程并不涉及数据的操控，因为此时仅仅是构建计算图，没有任何数据的输入。此时如果控制台输入一个节点，并不会有预想中的结果输出。

在复杂的网络中，我们可以将计算图划分为多个子图来计算，且子图还可以更细划分，直到单个特定的操作。

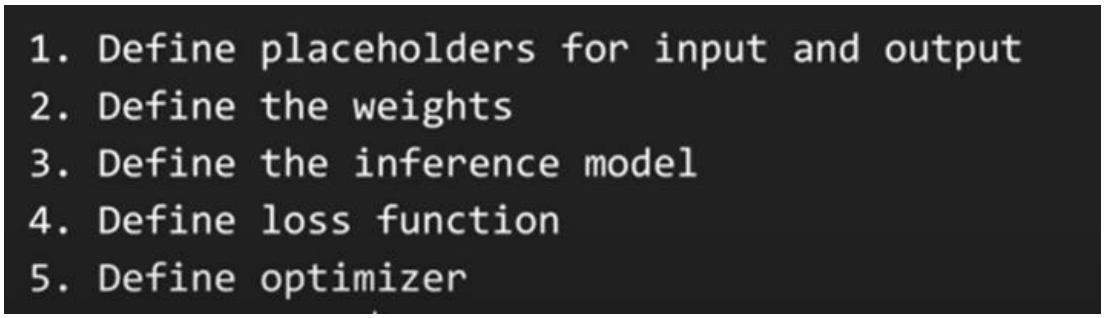

图 计算图的构建过程

2.2. Variables：

Variables，也就是变量。在定义中，变量就是有状态的节点，会输出它们的当前值，而且变量会在多次执行中保留它们的当前值。

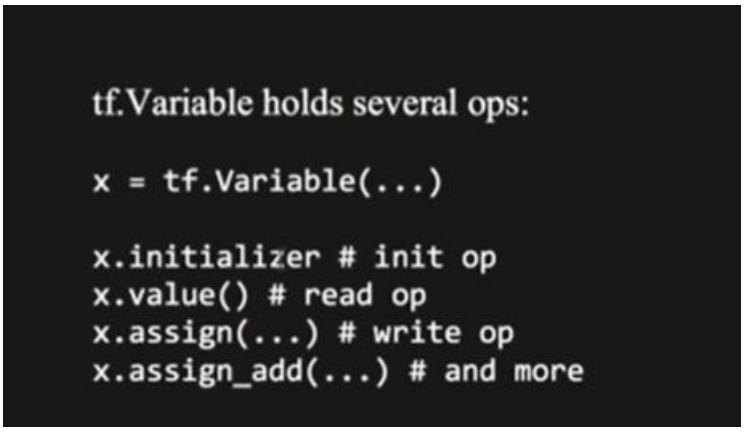

图 变量节点以及相关方法

在网络训练过程中，变量意味着我们感兴趣的，需要在训练过程中调整的参数(Parameters)，而不是输入的一维、二维数据或是图像等。我们在训练中不断修改变量以最小化损失(Loss)。

通过运用变量，我们可以方便地把数值保留在变量中，而变量可以在训练中或是训练后保存到硬盘中，这意味着我们可以保存这些模型参数，并在后续过程中发送给其他使用者等。

对于典型的线性关系$a = wx + b$，在我们当前讨论的情况下，权重和 bias 皆为变量。（课程中提到权重在定义中仍是一个操作？）

常用的构建变量节点相关操作如下：

$$tf.constant$$

$$tf.zeros\ /ones$$

$$tf.fill(dims, value)$$

$$tf.zeros\_like/ones\_like(a\ tensor)$$

$$tf.linspace(start, stop, num)$$

$$tf.range(start, limit = none, delta)$$

其中 constant 节点，由于在图的定义时即赋值，开销较为昂贵。

2.3. Placeholders：

在简单的定义中，Placeholders 代表在执行时输入的数值，并没有一个初始值，通常具有特定数据类型以及张量的维度信息等，以便于外部输入数据。

以下为一行创建 Placeholder 的代码，需要输入的参数分别为数据的类型，张量的维度信息，以及节点的名称。

$$\mathbf{tf.placeholder(tf.float32, shape = (None,), name = 'x')}$$

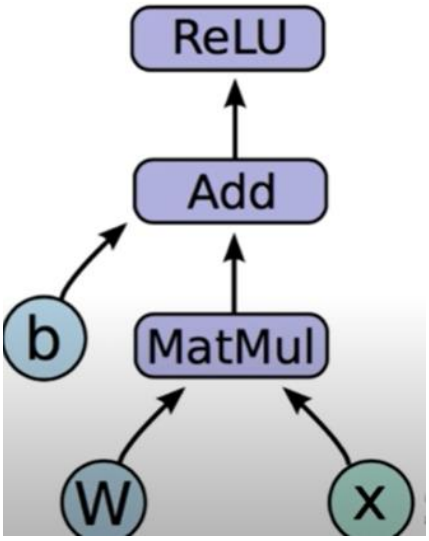

图 包含 placeholder 和 variables 的简单计算图

2.4. Sessions：

之前提到，数据运算等过程在构建计算图的过程中并不涉及，而是发生在构建完成后实际运行的过程中。而实际运行就需要用到 Session。

Session 是一种执行环境，而谷歌在研发一种 ，叫做 Tensor Processing Unit，这种可以加速张量的计算，从而加速整个网络训练过程。

因为我们需要将计算图部署在 Session 中，所以我们在部署前需要先实例化一个 Session，下为简单的实例化一个 Session 的代码。在实例化后即可执行 run 方法。

$$s = tf.Session()$$

同时，run 方法需要两个输入参数，分别是 Fetches 和 Feeds。其中，Fetches 指的是计算图节点的列表，而 Feeds 是一种字典映射，它将图节点和在模型中运行的实际值绑定在一起。从图的角度看，Fetches 即为计算图中的 Variables，而 Feeds 即为 Placeholders。

完成计算图构建和 Session 的实例化之后，即可通过代码来运行，下为课程中简单示例代码。(忽略对齐问题)

$$s = tf.Session()$$

$$s.run(tf.initialize\_all\_variables())$$

$$s.run(h,\{x:np.random.random(100,784)\})$$

从代码中可以看到，在运行前必须对所有 Variables 进行初始化操作。

在运行时，由于 Lazy Evaluation 机制的存在，计算机会针对你设置的要计算的目标操作，仅选取所有相关的节点加入计算，而不会选取无关操作节点。

总结来说，典型的步骤如下：构建计算图，初始化 Session，通过 Session.run 来训练。

2.5. 其他相关：

在简单的讨论过 Placeholders，Variables，Sessions 和计算图后，我们可以了解到在 TensorFlow 中网络的基础架构是如何完成的，任何复杂的网络都可以通过这几个基础部件组成。以下讨论几点相关内容，由于内容定义等在过去的报告中已大多提及多次，这里仅简要叙述并记录相关代码。

Loss：预测值与实际值(Label)的相差程度（通过不同损失函数定义）即为损失，通常以最小化 Loss 作为目标来训练网络。

下为 Loss 的定义过程代码。

$$pred = tf.nn.softmax(\dots)$$

$$label = tf.placeholder(tf.float32,[100,10])$$

$$crossEntropy = -tf.reduce\_sum(label * tf.\log(prediction))$$

在不同的损失函数中，对特定情况的敏感度也不同。例如在方差损失函数中，假如仅有少数偏离程度极大的点，也会产生很大损失。所以损失函数的定义通常取决于我们要解决的问题。

Gradient：梯度，我们通过略微梯度来调节略微权重值。在 TensorFlow 中，我们通过代码，来指示计算机，去采用特定的 Optimizer，学习率以最小化某个变量。

下为示例代码，在这行代码执行后，会在计算图中创建一个 Optimizer 节点，其中 Optimizer 采用梯度下降法，学习率 0.5，最小化损失，在训练过程中，计算机会找到所有损失依赖的可训练的变量节点，并通过梯度下降法或是定义的其他特定算法来更新这些变量节点的值。

```
tf.train.GradientDescentOptimizer

tf.train.AdagradOptimizer

tf.train.MomentumOptimizer

tf.train.AdamOptimizer

tf.train.ProximalGradientDescentOptimizer

tf.train.ProximalAdagradOptimizer

tf.train.RMSPropOptimizer

And more
```

图 基于不同算法的的优化器

$$trainStep = tf.train.GradientDescentOptimizer(0.5).minimize(loss)$$

Variable 共享：在训练中，我们可以通过多个设备的并行计算来加速这个过程。更大更复杂的模型也意味着需要共享更大的变量集合，课程中

提到，我们通常希望在同一个地方初始化所有变量，以下是提到的几种方法。

一是在代码的顶端建立一个变量字典，通过映射来共享，但这样做会对封装?(encapsulation)造成破坏，并且随着变量集的增大，这个字典也会越来越臃肿。

```
variables_dict = {
    "weights": tf.Variable(tf.random_normal([782, 100]),
                           name="weights")
    "biases": tf.Variable(tf.zeros([100]), name="biases")
    }
```

二是规定代码作用域，这是更为常见，且更高效简洁的做法。

tf.variable_scope()	provides simple name-spacing to avoid clashes
tf.get_variable()	creates/accesses variables from within a variable scope

通过以上的内容，基本的模型架构已经可以完成。这些内容大多是使用 TensorFlow 中偏底层的部分，而现今 TensorFlow 已经有许多更抽象的，使用起来更方便的 API，例如 Keras 等。

2.6. Keras 相关：

在之前的内容中可以看出，即使只是构建一个简单的线性回归模型，都没有想象中那么简单。假如我们想要对某一个已知结构的网络，从头编写代码，可能要花费大量的时间和人力。在计算机中，偏底层的相关操作通常耗时更短，而非底层的相关操作更易于理解。

所以，在这里我们讨论 Keras,TensorFlow 中已经较为成熟的 API。我个人感觉 Keras 和基本 TensorFlow 的区别大致类似 C 语言和 Python 的区别。

有了 Keras 的帮助，我们通常不需要从头编写全连接层、卷积层等等代码。只需要像搭建积木一样，调用不同的层，修改参数，将它们组合。

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

图 简单序列模型

但是，操作简单化的同时，性能也会有部分损失，在网络讨论中，有人指出对于它的问题来说，不使用 Keras API 要快于使用约 2.5 倍。并且相同的较为流行的网络结构在不同的框架下训练出的结果对比中，Keras 的表现也是偏差的。

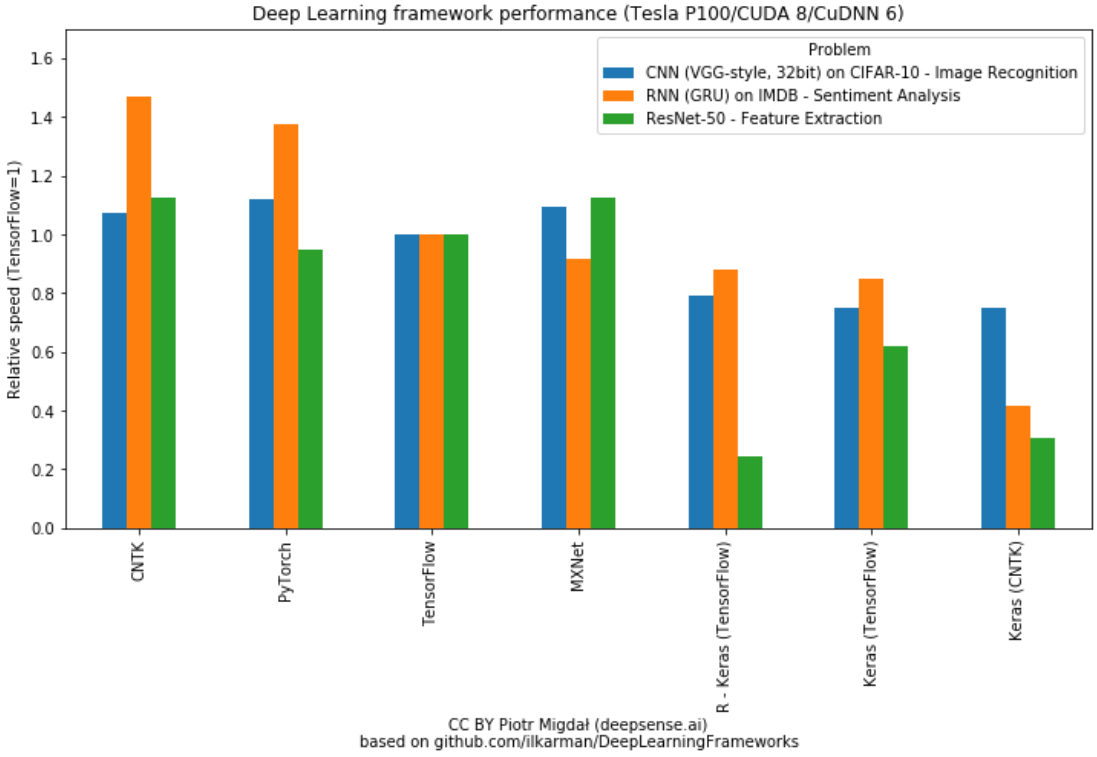

图 三种常用神经网络在不同平台框架下表现对比

3.1. 冷冻电镜单颗粒分析相关：

冷冻电镜逐渐成为确定蛋白质结构的流行方法，且结果证明它是有效的。但由于过程中涉及到许多平均操作，必须要有足够的数据、良好的信噪比才能获得能接受的结果。但冷冻电镜图像的信噪比偏低，要完成准确的重构就需要大量的观察。除此之外，网格上蛋白质浓度、数据收集的效率、颗粒识别的完整性准确性等因素决定了最后有多少颗粒能用来重构。很重要的一步就是颗粒挑选。

颗粒挑选耗费大量的时间，仅通过人力完成是不实际的，通常需要施加自动化方法来协助完成。常见的有 DoG 和 template-based approaches 等，但这些传统方法没办法检测形状有异常的颗粒，且有较高的假阳性，需要后期挑选。

另外一个存在的问题是，随着研究的推动，需要观测的蛋白质越来越小，小到难以和噪声区分，非球体的蛋白还会出现视线缺失的问题，典型的冷冻电镜显微图像的信噪比估计仅 0.11 以下，这在任何成像领域都是最低的。

更新的方法有基于 CNN 的方法，这种方法通过正标记负标记显微图像的区域来训练 CNN 分类器，并且为剩余的其他区域预测标签。但是因为信噪比低，研究者需要花费大量时间在打标签上。

所以有研究者提出，可以对图像进行降噪处理，降低信噪比。这样做也可以改善接下来的其他步骤，减少了数据收集过程的时间开销，并且可以选择更低电子剂量的电镜图像来分析。

深度神经网络主要通过学习复杂的非线性关系来填充丢失的像素，需要成对的噪声和 Ground Truth 图像，但对于冷冻电镜图像来说，没有 Ground Truth 图像指导学习，所以这种方法对冷冻电镜图像来说不适用。

为了解决这个问题，研究者提出了一种通用的机器学习框架，Noise2Noise，这种学习框架可以从噪声中学习降噪模型，而不需要 Ground Truth 图像。

3.1.1 Topaz 简介

Topaz 是一种深度学习方法，通过这个方法能够可靠且快速地提升图像的信噪比，这个方法在成千上万张在广泛成像条件下收集的显微图像中，学习出一种能够捕捉到冷冻电镜图像形成过程复杂性的模型。基于这种方法，研究者开发了 Topaz 管道。

Topaz 管道是一种高效且准确的颗粒采集管道，使用经过训练的神经网络来训练，网络采用的是 Positive-unlabeled Learning 学习方法。

为了解决上述提到的问题，研究者把颗粒挑选当作一个 PU 学习问题来看，仅用极少的标记数据点训练出高准确率的颗粒识别器，并且通过和自动编码器的正则化结合（）进一步减少需要的标记数据量。

从文章中可以看到，一个 Topaz Pipeline 主要步骤如下：

使用新设计的可选混合(optional mixed)模型对整个显微图像预处理，使用 PU 学习框架对神经网络分类器训练，显微图像的滑动窗口分类和通过非最大抑制来取得颗粒坐标。

3.1.2 Topaz 的优点

与传统方法相比，Topaz 框架可以使用很少的稀疏标记的颗粒来训练，并且从结果上看，Topaz 比起其他替代方法能挑选到更多的颗粒并保持一个较低的假阳性率。对于传统方法来说具有挑战性的蛋白质，例如体型较小、非球体、不对称的蛋白质颗粒，Topaz 的颗粒挑选仍能取得不错的表现，并且不需要后处理。

在实验数据中，Topaz 对一种具挑战性的蛋白质实现了 3.7Å 的重建，并解决了其他方法无法解决的二级结构。

Topaz 更好地检测了常规方法检测起来困难的视角，取得了更大比例的斜视、侧视和俯视图，降低了各向异性(anisotropy)。

从颗粒挑选看，Topaz 同时具有高召回率和高准确率，颗粒预测十分准确，几乎没有假阳性颗粒，在和其他常用方法对比中，经过四轮二维分类和滤波后，Topaz 的假阳性仅为 0.5%。

但需要注意的一点是，FCNN 和小型 U-Net 在小型数据集训练结果比完整 U-Net 模型更好。总体上看，性能最好的是完整 U-Net 加上大型数据集，这个模型在所有数据集上也优于传统的低通滤波去噪。

3. 参考资料：

1. 深度学习技巧之 Early Stopping（早停法）

https://blog.csdn.net/df19900725/article/details/82973049

2. Boost your CNN image classifier performance with progressive resizing in Keras

https://towardsdatascience.com/boost-your-cnn-image-classifier-performance-with-progressive-resizing-in-keras-a7d96da06e20（这篇是针对 Keras 的，仅大概扫了一下）

3. PyTorch for Deep Learning - Full Course / Tutorial

Pytorch Basics & Linear Regression

https://www.youtube.com/watch?v=GIsg-ZUyOMY

4. PyTorch for Deep Learning - Full Course / Tutorial

Image Classification & CNN

https://www.youtube.com/watch?v=GIsg-ZUyOMY

以及学姐提供的相关资料和论文。