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，其中单颗粒分析技术相关内容将在星期日报告后整合更新至上一周报告中。本周报告内容量以及文字量较少。

二、正文：

1. PyTorch 基础知识：

1.1. 张量和梯度：

正如文档中所说的，PyTorch 主要功能分别是提供类似于 numpy 库的 n 维张量和自动区分以构建和训练神经网络。在 python1 中通过 import torch 来导入 PyTorch 相关包并使用。

张量(tensor)是一种包含单一数据类型元素的多维矩阵，它可以是单个数字、向量或是任意 N 维数组。Torch 定义了八种 tensor 类型，分别是 16 位、32 位、64 位浮点张量（16 位浮点张量不适用于 CPU），8 位无符号整型张量以及 8 位、16 位、32 位、64 位符号整型张量。

Data tyoe	CPU tensor	GPU tensor
32-bit floating point	torch.FloatTensor	torch.cuda.FloatTensor
64-bit floating point	torch.DoubleTensor	torch.cuda.DoubleTensor
16-bit floating point	N/A	torch.cuda.HalfTensor
8-bit integer (unsigned)	torch.ByteTensor	torch.cuda.ByteTensor
8-bit integer (signed)	torch.CharTensor	torch.cuda.CharTensor
16-bit integer (signed)	torch.ShortTensor	torch.cuda.ShortTensor
32-bit integer (signed)	torch.IntTensor	torch.cuda.IntTensor
64-bit integer (signed)	torch.LongTensor	torch.cuda.LongTensor

图 不同类型张量

张量与 List 列表不同的地方在于，他对于每个维(行,列,通道等)的数有更加严格的限制。

同时，张量可以通过.to()方法移动至任何设备上，由于 GPU 可以大大加快我们的训练过程，而现在市面上较热门的 GPU 是 NVIDIA 厂家生产的，所以要涉及到 CUDA 的使用。我们可以通过定义一个简单的方法来让计算机决定使用的设备。

```
def GPU_if_avilable:
  if torch.cuda.is_available():
    return torch.device('cuda')
  else:
    return torch.device('cpu')
```

下面给出在实践过程中较常用的一些方法等。

```
a = torch.Tensor([1,2,3] , required_grad=True)
a.shape
a.backward
a.grad
a.dtype
torch.matmul(a,b)
torch.rand(rows, columns)
```

行 1 创建了一个张量 a，[1,2,3]代表张量 a 代表的向量。torch.Tensor 创建的是默认类型的张量，也就是浮点类型张量。

参数 required_grad 决定了张量的梯度是否需要计算，因实际过程中可能需要计算的参数过多，所以我们可以选择只计算部分梯度，或是在训练过程中，我们想要固定(freeze)某一个或者某些层时就可以通过修改这个参数来完成。

行 2 会输出这个张量的各个维的大小，例如对于图像来说，将分别输出图像的长度、宽度以及通道数。对于 numpy 类型同样适用。

也有出现顺序不同的情况，我们可以通过 a.permute 来更改顺序。

行 3

行 4 会计算这个张量的梯度值，若张量为向量，矩阵或是多维数组类型的形式，计算梯度后输出结果依然是相同的类型。

行 5 输出张量的类型。

在实际运用过程中，还会根据情况导入其他包辅助使用。

例如 numpy 包，numpy 包已经十分成熟，在许多不同的场景下都会用到，但 numpy 类型相比较于 tensor 类型有一个不足之处：numpy 类型无法用于 GPU 计算，而对于矩阵运算来说，GPU 的速度以及效率远远超过 CPU。

或是 matplotlib 以及 opencv 包，这两个包多用于图像处理，包括图像显示，图像切割以及其他图像操作等。

在实际运用过程中，有时会同时用到 numpy 和 tensor 类型，而这两种类型也可以互相转换。例如通过 torch.from_numpy()将 numpy 类型转换为 tensor 类型。这种转换方法用到相同的存储空间，而 torch.tensor()方法则会创建拷贝。总的来说，numpy 类型适用于 CPU，tensor 类型适用于 GPU 且 PyTorch 将数据搬运至 GPU 中。

行 6 为矩阵乘法过程。

行 7 为创建随机初始化矩阵。

1.2.数据集准备过程：

数据集是至关重要的一部分，数据集的好坏通常决定了这个网络能达到的表现的上限。更大的数据集可以使网络更完善，给予网络更好的细分能力。但数据集也并不是越大越好，由于实际生活中依然存在着算力限制，就算一个网络经过训练能达到几乎完美的表现，如果训练时长远超使用者可以承受的范围也是没有意义的，由于通常要用网络解决的问题也有时效性，如何平衡表现和时长也是实际运用中需要关注的一点。

数据集根据实际问题变化而变化，对于图像处理相关问题来说，数据集通常包含大量的图像以及标签或其他附加信息。

数据集中的图像最好要达到同一种标签中尽可能地多变，以使得训练后的网络具有不同的鲁棒性。并且标签的设定也需要严谨。例如网络课程中提到的，一种判定照片是拍摄于黑天还是白天的网络，要准备训练这种网络的训练集需要充分考虑不同的情况。例如黄昏、黎明、凌晨等时间段，在打标签环节种，应该打上白天还是黑天的标签。另外为了使网络能应对大多数的情况，数据集中也应当包含室内、室外、不同的窗户等不同情况拍摄的照片。

数据集中图像大小也会影响到整个训练过程，图像越大，包含的像素就越多，学习特征的过程就越长，学习后的效果也越好。包含像素少的图像又能在更短的时间内完成训练和学习，这同样是需要平衡的一点。举例说明，还是在上面提到的判定拍摄时间的网络中，图像就并不需要很大，因为需要判定的特征较为明显，但不能小到无法判定时间特征。

在完成数据集的准备后，我们需要将数据集分为三部分，分别是训练集(training set)、验证集(validation set)以及测试集(test set)。

其中训练集的作用是练模型，计算损失并根据梯度下降法更改权重，验证集的作用是训练模型时评估模型，修改超参数（学习率等）并选出最佳版本的模型。测试集的作用是对比不同模型等，并给出最终的准确率。

这三个分支占总数据集的百分比并没有一个最佳的选择，通常的做法是在数据集中，根据数据集的大小取80%~85%作为训练集，剩余部分作为验证集以及测试集，或是在训练集中取一部分作为验证集，数据集剩余的部分作为测试集。

在实际过程中，torch.utils.data.Dataset是代表数据集的抽象类。自定义数据集继承Dataset并覆盖以下方法：

__len__，以便len(dataset)返回数据集的大小。

__getitem__支持索引，以便可以使用dataset[i]获取第个样本。

对于一些无法取得足够大数据集的问题，我们可以通过数据增强来提高网络表现。可用的数据增强手段很多：

例如在U-Net中使用的，对数据集中图像进行弹性形变，平移旋转等。

例如在上文提到的判定拍摄时间的网络中，对数据集中图像进行灰度变化，对比度变化等。

在文档中提到，样本的大小可能不同。大多数神经网络期望图像的大小固定。因此，可以编写一些预处理代码。

Rescale：缩放图像

RandomCrop：从图像中随机裁剪，是数据增强的一种。

ToTensor：将 numpy 图像转换为 torch 图像。

在准备好一个可以满足我们使用需求的数据集之后，就可以进入模型训练过程了。

1.3.模型训练过程：

模型训练的过程大概分为以下几个步骤：生成预测、计算损失、计算梯度、修改权重来影响梯度、梯度置0。

```
optimizer.zero_grad()               ## 梯度清零
preds = model(inputs)               ## 生成预测
loss = criterion(preds, targets)    ## 求解 loss
loss.backward()                     ## 反向传播求解梯度
optimizer.step()                    ## 更新权重参数
```

伪代码表示：

```
for epoch in range()
  for data in dataset
     calculate loss
     compute gradients
     adjust weight
     gradients to zero
```

首先我们从伪代码的第一行开始，这是对每个 epoch 进行循环操作，由于用到了梯度下降法，梯度值为正时，微增加权重值会使得损失值也有细微的上升，梯度值为负时，微增加权重值会使得损失值细微下降。所以梯度下降法的重点就在于找到梯度的反方向，并更新权重参数，更新权重参数通常是增加或减小一个十分小的值，这个值被称为训练过程中的学习率(learning rate)是超参数之一。

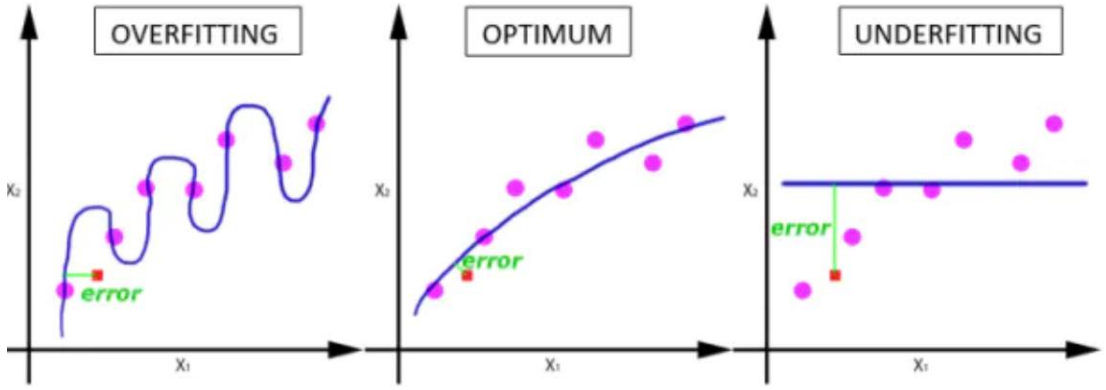
由于我们需要沿着梯度的反方向来更新权重参数，大概的代码表示如下：

```
weight = weight - grad * learning_rate
```

这样的代码实现就可以使权重在梯度为正值时，略微减小权重值，损失值也略微减小，梯度为负值时，权重值略微上升，损失值略微减小。

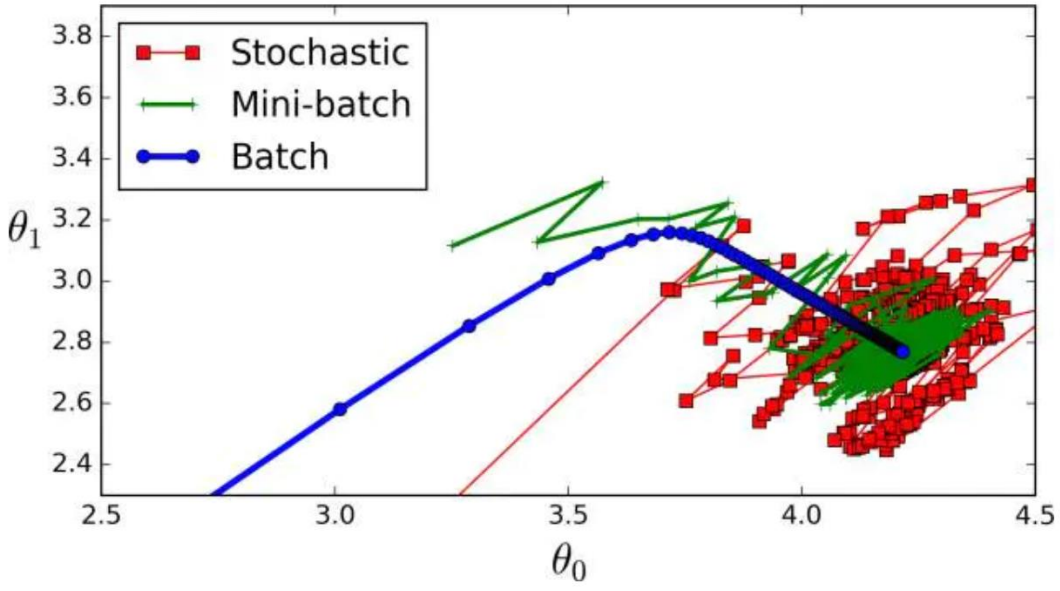
回到伪代码第一行，由于训练过程中，数据集难以一次性送入网络，于是我们会使用 epoch，batch，iteration 来协助分割数据集。

epoch 指的是网络完成单次反向传播的过程。但 epoch 通常很大，算力限制下需要再次分割。



伪代码中也有体现出，epoch 是一次循环的过程，所以单个 epoch 是无法完成网络的训练的，epoch 的数目的决定是网络训练中重要的一点，上图为不同数目的 epoch 训练的网络。从一开始的不拟合(underfitting)到优化拟合(optimum)到过拟合(overfitting)，从这张图我们可以看出，epoch 的数目和许多之前的问题讨论过的参数一样，并非越多越好。

batch 是 epoch 分割出的更小的部分。batch size 是每个 batch 送入网络的样本数目，不同的 batch size 下，最终网络的结果也不同。



蓝色为送入所有数据，单个 batch，红色为随机训练，每个 batch 仅包含一个样本，绿色为每个 batch 包含一小部分数据。图中的结果显示送入所有数据能达到最好的效果，但对算力的要求也最大，花费的时间可能也更长。

Iteration 是一次迭代，即 epoch 中包含的 batch 数目。

接下来到伪代码的第三行（第二行内容已在数据集部分涉及到）：

```
1.      calculate loss
```

模型中，样本的预测值与真实值的差值即为损失。损失越小，模型的表现越好。用于计算损失的函数称为损失函数。模型每一次预测的好坏用损失函数来度量。

其中损失函数大概基于下列几种：0-1 函数、平方函数、绝对值函数以及对数函数。0-1 函数通常用于二分类分类器，平方函数的损失为预测值和真实值差的平方，相当于扩大了损失的影响。

```
1.      compute gradients
2.      adjust weight
3.      gradients to zero
```

伪代码剩下的三步我认为可以合并为一步，这三行伪代码实际上指代的是一个基于梯度下降法，通过梯度值，改变权重值的过程。其中，由于 pytorch 支持的动态计算图机制，backword()和 optimizer.step()不会将梯度置为 0，所以需要手动添加一行梯度置为 0 的代码。

对于动态计算图我尚未理解透彻，这里引用网上的解释：

Tensorflow 先建立好图，在前向过程中可以选择执行图的某个部分(每次前向可以执行图的不同部分，前提是，图里必须包含了所有可能情况)。

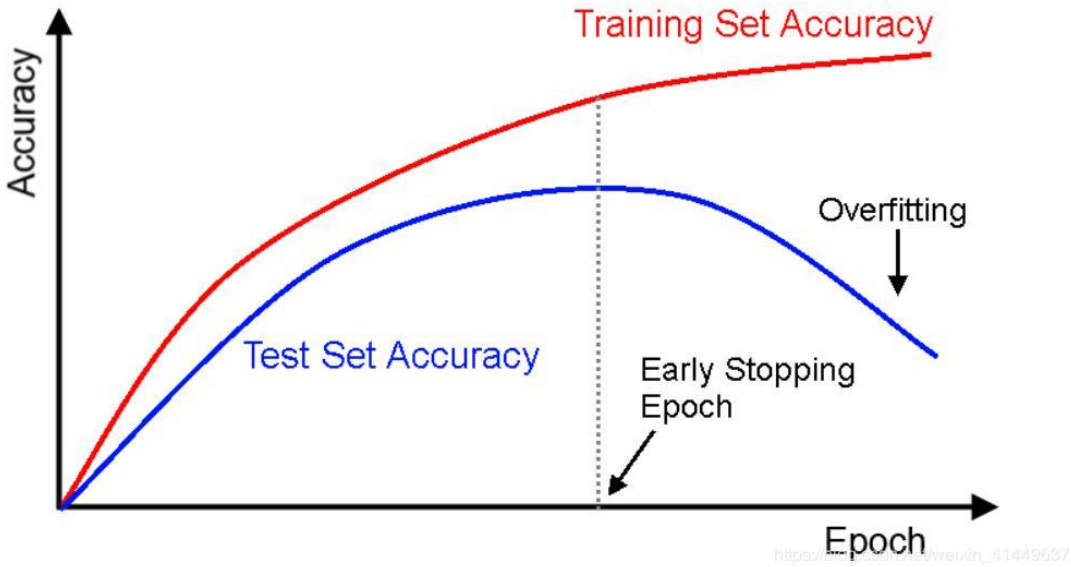
PyTorch 是每次前向过程都会重新建立一个图，反向的时候会释放，每次的图可以不一样。

1.4.超参数优化：

成功的超参数的微调可以加速模型训练，增强模型表现。超参数是机器学习模型中的参数的一类，与参数(Parameters)不同的是，超参数并非通过学习和估计得到，而是需要人为设定（但也有帮助我们优化的软件等）。

超参数的优化方法有很多，大多我还未了解，这里只简述我大概理解的一项，Callbacks 中的 Early-stopping。

Early-stopping 适用的场景是模型在训练集上表现很好，在验证集上表现很差，通常这种情况认定为模型出现了过拟合。



Early-stopping 可以设定不同的停止标准，在网络达到停止标准时便会停止训练，例如上图，如果出现了训练集准确率不断上升而测试集准确率不断下降时，即过拟合时，则会停止训练。

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。