**学习进度**

本周学习任务：

学习李沐老师《动手学深度学习（PyTorch版）》，关于**线性神经网络、多层感知机**及**深度学习计算**部分的内容。

1. 引言

李沐老师的动手学深度学习课程，分别从自定义组件及基于PyTorch的角度出发，详细阐述了“如何构建并应用深度学习模型”这一核心问题。本次周报，根据线性神经网络、多层感知机及深度学习计算部分的实践学习，试图简要总结：浅层神经网络模型的代码实现流程及重要组件，并额外介绍一种脱离课程的神经网络模型完整实现方式。

1. 数据操作

模型本质上依赖基于张量的计算方式，展开训练评估工作。即使开发框架提供了许多功能组件，在自行编写网络模型代码时，仍需手动实现网络中流动数据的具体处理方式，从而更好地联结功能组件以完成项目开发。

1. 生成

在权重初始化、模块定义及测试等环节，可以依赖开发框架Torch提供的方法，创建并初始化数据容器。具体方式如下：

✸生成：值为序列0-12的张量

**torch.arange**(12, dtype=torch.float32)

✸生成：值全为0的张量

**torch.zeros**((dim1\_len, dim2\_len, dim3\_len))

✸生成：值全为1的张量

**torch.ones**((dim1\_len, dim2\_len, dim3\_len))

✸随机生成：服从“均值为0，标准差为1”的标准高斯（正态）分布

**torch.randn**(dim1\_len, dim2\_len)

✸随机生成：服从自定义高斯分布的张量（size：输出张量的大小）

**torch.normal**(mean, std, size)

✸重塑生成

**torch.tensor**([[1,2,3], [4,5,6], [7,8,9]])

✸克隆生成

B = A**.clone()**

1. 查询

在代码纠错、数值计算及参数修改等环节，x**.shape**可以查询变量的维度、x**.numel()**可以查询变量的元素总数、**len**(x)可以查询变量的长度。除此之外，需要补充的是：

✸查询指定层：所有参数

net[2]**.state\_dict()**

✸查询指定层：指定参数

net[2]**.bias**

✸查询指定层：指定参数的值

net[2]**.bias.data**

✸查询指定层：指定参数的梯度

net[2]**.weight.grad**

✸查询所有层：所有参数

print(\*[(name, param.data) for name, param in **net.named\_parameters()**])

✸穿透嵌套块，查询参数

【net = nn.Sequential(block(), nn.Linear(4, 1))】访问块block中第二个子块的第一层偏置项数值

**net[0][1][0]**.bias.data

（3）其它

若需要保证或修改张量**维度**，可以使用x**.reshape()**；

若需要**拼接**张量，使用torch**.cat**((x, y), dim=0)按行追加，也可修改dim=1按列追加；

若需要**L1范数**：torch**.abs(x).sum()**，需要**L2范数**：torch**.norm(x)**；

若需要元素**求和**，使用x**.sum()**求和，也可按行求和x.sum(axis=0)；

若需要元素**相乘**，向量点积：torch**.dot**(x, y)，矩阵与向量相乘：torch**.matmul**(A, x)，矩阵相乘：torch**.mm**(A, B)。torch.matmul方法最为通用。

... ...

1. 流程组件

在神经网络模型的程序实现过程中：可以完全自定义功能组件，可以继承重写开发框架torch提供的类，也可以直接使用torch提供的封装功能组件。

1. 数据加载

关于数据获取，可采用官网下载、爬虫采集及自行构造等方式。当然，torchvision模块提供了部分典型数据集的下载方法，还可指定：存储路径、是否为训练集及数据变换方式。

以下载并读取FashionMNIST数据集作为训练集为例：

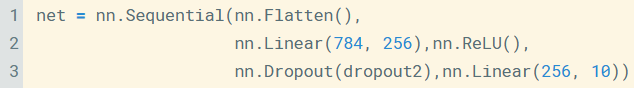
mnist\_train = **torchvision.datasets**.FashionMNIST(root="../data", train=True, transform=trans, download=True)

关于生成数据迭代对象，可以自行编写迭代函数（打乱数据集索引，每个迭代周期都依序抽取固定批次长度的数据），可以继承重写源自开发框架的DataLoader类，也可以直接使用data.DataLoader()生成数据迭代对象，从而达到分批读数据的目的：

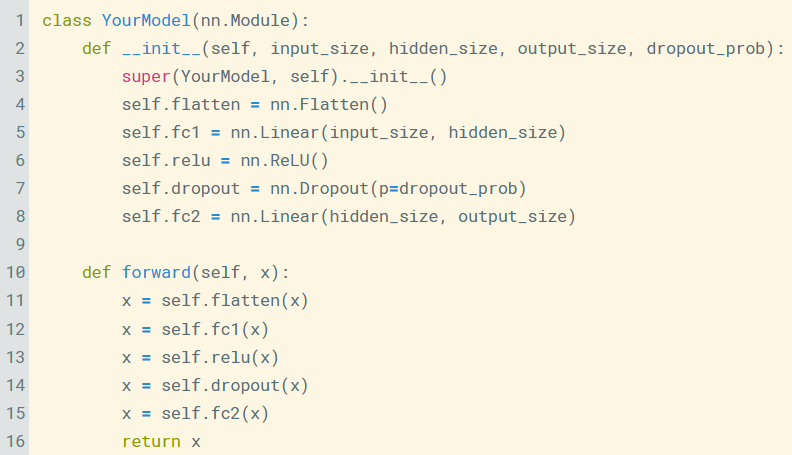
train\_iter = **data.DataLoader**(mnist\_train, batch\_size, shuffle=True,num\_workers=4)

1. 定义模型

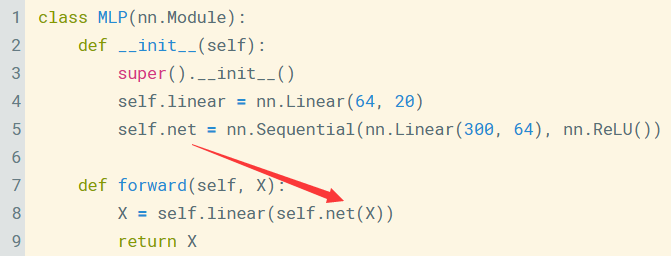
以构建多层感知机模型为例，可以使用开发框架torch提供的封装方法nn.Sequential()，完成多层感知机模型网络架构的定义：



**更为普遍的做法是：继承重写nn.Module类**（在初始化函数中罗列功能组件，在前向传播函数中明确网络架构），更灵活简便地完成模型定义。

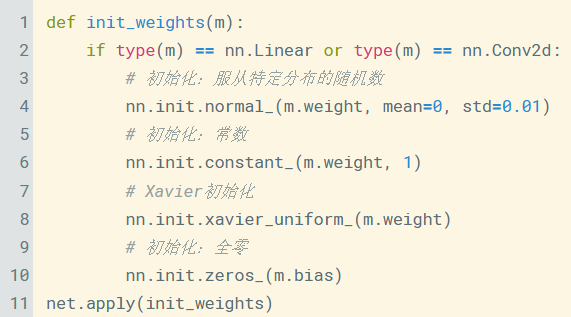


值得注意的是，在初始化方法\_\_init\_\_中，还可以声名嵌套块（子网络），并在前向传播函数forward的某个网络层中引用。



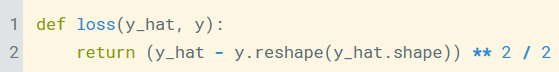
1. 参数初始化

当模型实例化后，开发框架torch默认完成了参数的随机初始化工作。当然，也可以自行编写参数初始化方法，并通过.apply方法将参数初始化策略加载到实例化模型对象中。



1. 定义损失函数

针对回归问题，以均方损失为例。可以直接使用loss = nn.MSELoss()完成定义损失函数，也可以自行定义均方损失函数：



针对分类问题，以交叉熵损失为例。可以直接使用loss = nn.CrossEntropyLoss()完成损失函数的定义，也可以自行定义交叉熵损失函数：

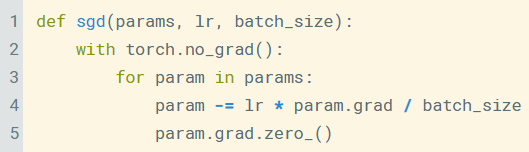


1. 定义优化函数

以使用随机梯度下降（SGD）法定义优化器为例，常见做法如下：

trainer = torch**.optim.SGD**(net.parameters(), lr=0.03, weight\_decay=0.001)

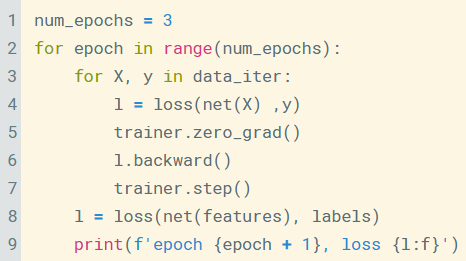
当然，也可以传入可学习参数、学习率及批次大小，自行编写sgd函数：



1. 训练模型

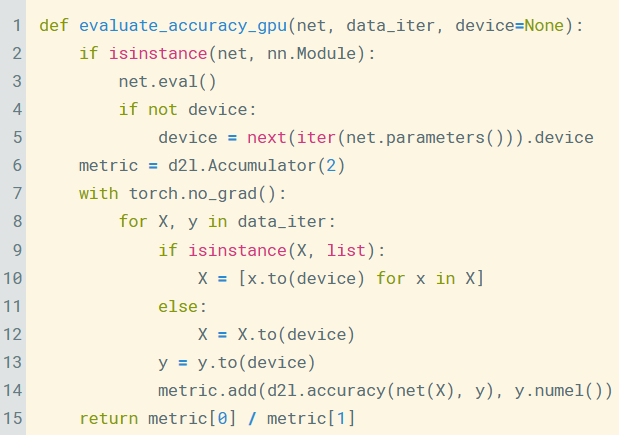
完成前述流程后，似乎可以正式开始模型训练了。模型训练的基本过程是：分周期从数据迭代对象中获取对应的批次数据，将数据投喂给实例化模型对象计算预测值，通过损失函数计算当前批次数据预测结果与真实值之间的损失，依赖损失计算可学习参数的梯度信息，通过格式化处理的优化器更新可学习参数的值。

基于开发框架torch完成模型训练的代码实例如下：



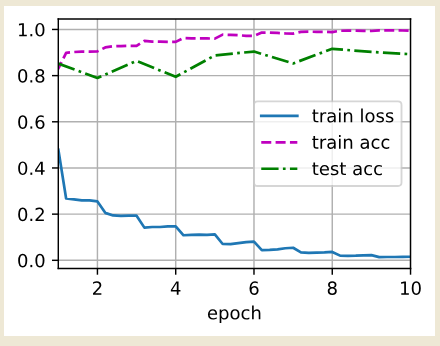
1. 评估模型

一般可选用准确率，作为模型泛化性能的度量指标。若需要依赖GPU计算模型在测试集上的精度，代码实现如下：



1. 可视化单元

自定义Animator类，作为模型训练与评估环节的可视化单元，用于：每次迭代周期末尾，逐步向图表中添加对应数据（迭代轮次、训练损失、训练准确率及测试准确率），实时绘制反映模型训练程度及泛化性能的曲线。训练结束后，最终的可视化结果如下所示：



Animator类的实现方法省略，详见书籍《动手学深度学习（PyTorch版）》P119。

4、重要补充

1. 调参

在神经网络模型中，除了可学习参数外，还需要手动设定超参值。针对具体的任务情景及数据特征，不同的超参取值组合，往往极大影响模型的泛化性能。经验是：以前人在类似环境中推荐使用的超参取值为基准，进行特异性调试。常见的调参方法有：网格搜索、随机搜索、贝叶斯优化及自动调参工具等。

这里简要介绍如何基于K折交叉验证法展开调参工作：首先，定义超参数网格字典对象（键：超参名、值：测试取值列表）；其次，迭代选取超参数网格中所有可能的取值组合；然后，依赖当前的超参取值训练并评估模型性能；最后，记忆更好效果的超参取值组合。由于篇幅有限，实现代码省略。

显而易见的是，调参是一个讲究经验的麻烦工作。这里简要介绍一些自动调参工具：

【参考：https://zhuanlan.zhihu.com/p/364613087】

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 工具 | 参数搜索 | 模型框架搜索 | 并行支持 | 是否支持Torch | 强化学习 |
| NNI | √ | √ | √ | √ |  |
| Google Vizier | √ | √ | √ | √ |  |
| **Ray Tune** | √ |  | √ | √ | √ |
| Hyteropt | √ |  | √ | √ |  |

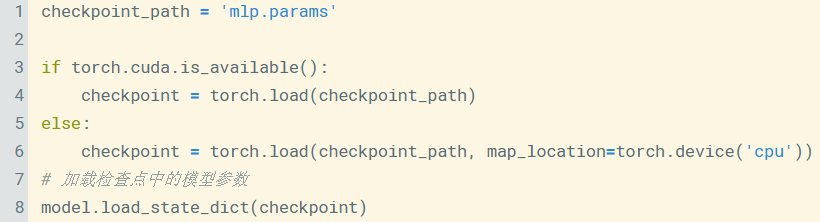
1. 检查点

由于深层神经网络模型需要花费大量时间，从海量数据中迭代学习目标规律，以不断地更新可学习参数的取值。当训练途中偶发意外，将导致之前的训练成果全部丢失。

为此，在实现模型训练与评估函数时，可以设置基于验证/测试集的性能提升作为触发条件，保存泛化性能更为优异的模型参数取值。实现方式如下：



若需要加载原先保存的模型参数，继续展开模型训练工作。可以在模式实例化之后，模型训练之前，加载对应检查点：

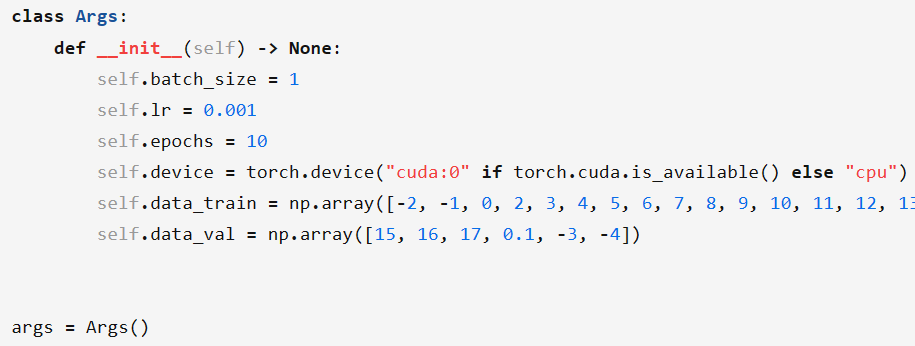


1. 神经网络模型的简洁实现流程

【参考：https://zhuanlan.zhihu.com/p/596449652】

首先，设置随机数种子。

其次，以类的方式定义参数（还可以使用config文件方法），示例如下：



然后，依赖参数实例args定义数据集，并结合数据迭代器DataLoader，完成训练集迭代对象与验证集迭代对象的加载工作。

再后，定义并实例化模型、损失函数及优化器。

此外，依赖循环迭代完成：模型训练、评估与可视化工作，并使用检查点保存模型参数。

最后，实例化模型并加载检查点，完成预测应用。