


实操教程 | Pytorch Debug指南：15条重要建议

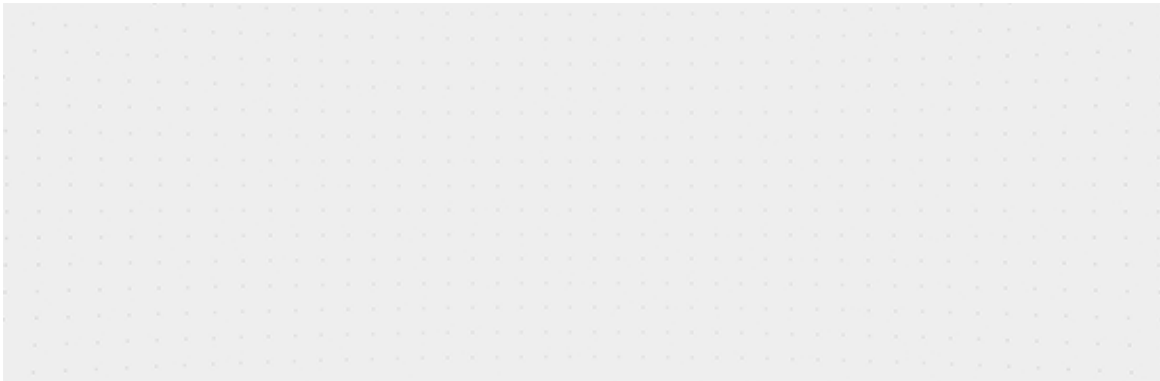
极市平台 2023-05-21 22:00:42 发表于广东 手机阅读 𐄂

以下文章来源于Coggle数据科学，作者Coggle

**Coggle数据科学**

Coggle全称Communication For Kaggle，专注数据科学领域竞赛相关资讯分享。

↑ 点击蓝字 关注极市平台



作者 | Coggle
来源 | Coggle数据科学
编辑 | 极市平台

极市导读

在使用 **Pytorch** 时你或多或少会遇到各种bug，为了缓解你的痛苦🥹，本文将对常见的错误进行解释，并说清楚来龙去脉 >>加入极市CV技术交流群，走在计算机视觉的最前沿

在使用 **Pytorch** 时你或多或少会遇到各种bug，为了缓解你的痛苦🥹，本文将对常见的错误进行解释，并说清楚来龙去脉。

细节就是魔鬼，虽然代码不报错但还是可能会对精度带来影响。如果本文对你有帮助，请收藏&转发！

CrossEntropyLoss和NLLLoss

最常见的错误是损失函数和输出激活函数之间的不匹配。 `nn.CrossEntropyLossPyTorch`

`h` 中的损失模块执行两个操作：`nn.LogSoftmax` 和 `nn.NLLLoss`。

因此 `nn.CrossEntropyLossPyTorch` 的输入应该是最后一个线性层的输出。不要在 `nn.CrossEntropyLossPyTorch` 之前应用 **Softmax**。否则将对 Softmax 输出计算 log-softmax，将会降低模型精度。

如果使用 `nn.NLLLoss` 模块，则需要自己应用 log-softmax。`nn.NLLLoss` 需要对数概率，而不是普通概率。因此确保应用 `nn.LogSoftmax` 或 `nn.functional.log_softmax`，而不是 `nn.Softmax`。

Softmax的计算维度

注意 Softmax 的计算维度。通常是输出张量的最后一个维度，例如 `nn.Softmax(dim=-1)`。如果混淆了维度，模型最终会得到随机预测。

类别数据与嵌入操作

对于类别数据，常见的做法是进行数值编码。但对于深度学习而言，这并不是一个很好的操作，数值会带来大小关系，且会丢失很多信息。因此对于类别数据建议使用 **one-hot** 或 **Embedding** 操作，对于 `nn.Embedding` 模块，你需要设置的参数包括：

- `num_embeddings`：数据类别的数量
- `embedding_dim`：每个类别的嵌入维度
- `padding_idx`：填充符号的索引

嵌入特征向量从随机初始化，不要用 Kaiming、Xavier 初始化方法。因为标准差为 1，初始化、激活函数等被设计为输入标准差为 1。`nn.Embedding` 模块的示例用法：

```
import torch
import torch.nn as nn
# Create 5 embedding vectors each with 32 features
embedding = nn.Embedding(num_embeddings=5,
                        embedding_dim=32)

# Example integer input
input_tensor = torch.LongTensor([[0, 4], [2, 3], [0, 1]])

# Get embeddings
embed_vectors = embedding(input_tensor)

print("Input shape:", input_tensor.shape)
```

```
print("Output shape:", embed_vectors.shape)
print("Example features:\n", embed_vectors[:, :, :2])
```

nn.LSTM 中 数据维度

默认情况下，PyTorch的 `nn.LSTM` 模块假定输入维度为 `[seq_len, batch_size, input_size]`，所以确保不要混淆序列长度和批大小的次数。如果混淆LSTM仍然可以正常运行，但会给出错误的结果。

维度不匹配

如果PyTorch执行矩阵乘法，并两个矩阵出现维度不匹配，PyTorch会报错并抛出错误。但是也存在PyTorch不会抛出错误的情况，此时未对齐的维度具有相同的大小。**建议使用多个不同的批量大小测试您的代码，以防止维度不对齐。**

训练和评估模式

在PyTorch中，神经网络有两种模式：`train` 和 `eval`。您可以使用 `model.eval()` 和 `model.train()` 对模型时进行切换。不同的模式决定是否使用 `dropout`，以及如何处理 `Batch Normalization`。常见的错误是在 `eval` 后忘记将模型设置回 `train` 模式，确定模型在预测阶段为 `eval` 模式。

参数继承

PyTorch支持 `nn.Modules`，一个模块可以包含另一个模块，另一个模块又可以包含一个模块，依此类推。

当调用 `.parameters()` 时，PyTorch会查找该模块内的所有模块，并将它们的参数添加到最高级别模块的参数中。

但是PyTorch不会检测列表、字典或类似结构中模块的参数。如果有一个模块列表，请确保将它们放入一个 `nn.ModuleList` 或 `nn.Sequential` 对象中。

参数初始化

正确初始化模型的参数非常重要。用标准正态分布初始化参数不是好的选择，推荐的方法有 `Kaiming` 或 `Xavier`。

zero_grad()

请记住在执行 `loss.backward()` 之前调用 `optimizer.zero_grad()` 。如果在执行反向传播之前没有重置所有参数的梯度，梯度将被添加到上一批的梯度中。

指标计算逻辑

在怀疑自己或模型之前，请经常检查您的指标计算逻辑计算两次或更多次。像准确性这样的指标很容易计算，但在代码中添加错误也很容易。例如，**检查您是否对批次维度进行了平均，而不是意外对类维度或任何其他维度进行平均。**

设备不匹配

如果使用GPU可能会看到一个错误，例如：

```
Runtime Error: Input type (torch.FloatTensor) and weight type (torch.cuda.FloatTensor) should be on the same device.
```

此错误表示输入数据在CPU上，而权重在GPU上。确保所有数据都在同一设备上。这通常是GPU，因为它支持训练和测试加速。

nn.Sequential和nn.ModuleList

如果模型有很多层，推荐将它们汇总为一个 `nn.Sequential` 或 `nn.ModuleList` 对象。在前向传递中，只需要调用`sequential`，或者遍历模块列表。

```
class MLP(nn.Module):

    def __init__(self, input_dims=64, hidden_dims=[128,256], output_dims=10):
        super().__init__()
        hidden_dims = [input_dims] + hidden_dims
        layers = []
        for idx in range(len(hidden_dims)-1):
            layers += [
                nn.Linear(hidden_dims[i], hidden_dims[i+1]),
                nn.ReLU(inplace=True)
            ]
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)
```

参数重复计算

在深度神经网络中，通常会有重复添加到模型中的块。如果这些块需要比更复杂的前向函数，建议在单独的模块中实现它们。例如，一个 ResNet 由多个具有残差连接的 ResNet 块组成。ResNet 模块应用一个小型神经网络，并将输出添加回输入。最好在单独的类中实现这种动态，以保持主模型类小而清晰。

输入相同的维度

如果您有多个具有相同输入的线性层或卷积，则可以将它们堆叠在一起以提高效率。假设我们有：

$$\begin{aligned}y_1 &= W_1 x + b_1 \\ y_2 &= W_2 x + b_2\end{aligned}$$

虽然可以通过两个线性层来实现它，但您可以通过将两层堆叠为一层来获得完全相同的神经网络。单层效率更高，因为这代表单个矩阵运算，而不是 GPU 的两个矩阵运算，因此我们可以并行化计算。

```
x = torch.randn(2, 10)

# Implementation of separate layers:
y1_layer = nn.Linear(10, 20)
y2_layer = nn.Linear(10, 30)
y1 = y1_layer(x)
y2 = y2_layer(x)

# Implementation of a stacked layer:
y_layer = nn.Linear(10, 50)
y = y_layer(x)
y1, y2 = y[:, :20], y[:, 20:50]
```

使用带logits的损失函数

分类损失函数（例如二元交叉熵）在 PyTorch 中有两个版本：`nn.BCELoss` 和 `nn.BCEWithLogitsLoss`，建议和推荐的做法是使用后者。这因为它在数值上更稳定，并在您的模型预测非常错误时防止出现任何不稳定性。

如果您不使用 logit 损失函数，则当模型预测不正确的非常高或非常低的值时，您可能会遇到问题。

公众号后台回复“**对比学习综述**”获取最新对比学习PDF资源**极市平台**

为计算机视觉开发者提供全流程算法开发训练平台，以及大咖技术分享、社区交流...
848篇原创内容

公众号

极市干货

极视角动态：极视角亮相BEYOND Expo，澳门特别行政区经济财政司司长李伟农一行莅临交流 | 极视角助力构建城市大脑中枢，芜湖市湾沚区智慧城市运行管理中心上线！

数据集：60+开源数据集资源大合集（医学图像、卫星图像、语义分割、自动驾驶、图像分类等）

多模态学习：CLIP：大规模语言-图像对比预训练实现不俗 Zero-Shot 性能 | ALBEF：图文对齐后再融合，借助动量蒸馏高效学习多模态表征

极市算法开发工具

算法开发效率提升25%

极市平台现已推出目标检测训练套件，涵盖了模型训练、调优、评估、测试、导出等功能，帮助开发者们更快速的通过平台训练导出模型！

亮点速览：

- 1) 训练套件拥有数据转换、划分、增强等数据预处理能力
- 2) 预置SOTA网络高性能实现，囊括主流CV任务
- 3) 提供 onnx, atlas , TensorRT等模型转换工具
- 4) 提供统一的跨硬件推理接口

开发套件体验活动招募中！使用套件完成开发后将使用体验和反馈给极市，我们将会送出的**瑞幸/奈雪的30代金券**～



长按扫码了解活动

获取目标套件使用指南



[点击阅读原文进入CV社区](#)

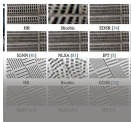
[收获更多技术干货](#)

[阅读原文](#)

喜欢此内容的人还喜欢

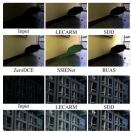
ICCV 2023 | 南开程明明团队提出适用于SR任务的新颖注意力机制（已开源）

极市平台



ICCV23 | 将隐式神经表征用于低光增强，北大张健团队提出NeRCo

极市平台



实践教程 | 从零开始用pytorch搭建Transformer模型

极市平台

