人工智能与商业创新

第 3 次作业、手写数字识别

姓 名: 孟 念
学 号: 1800092850
班 号: 02839210

人工智能与商业创新
(秋季, 2021)

北京大学
光华管理学院
彭一杰老师
高华西助教

2021 年 11 月 25 日

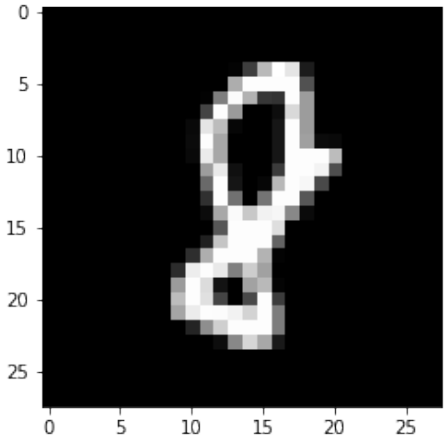
目 录

1 基本要求

使用神经网络对手写数字进行分类。
实验报告，主要内容如下:

- 模型结构和分析
- 实验结果和比较
- 思考
- 心得体会

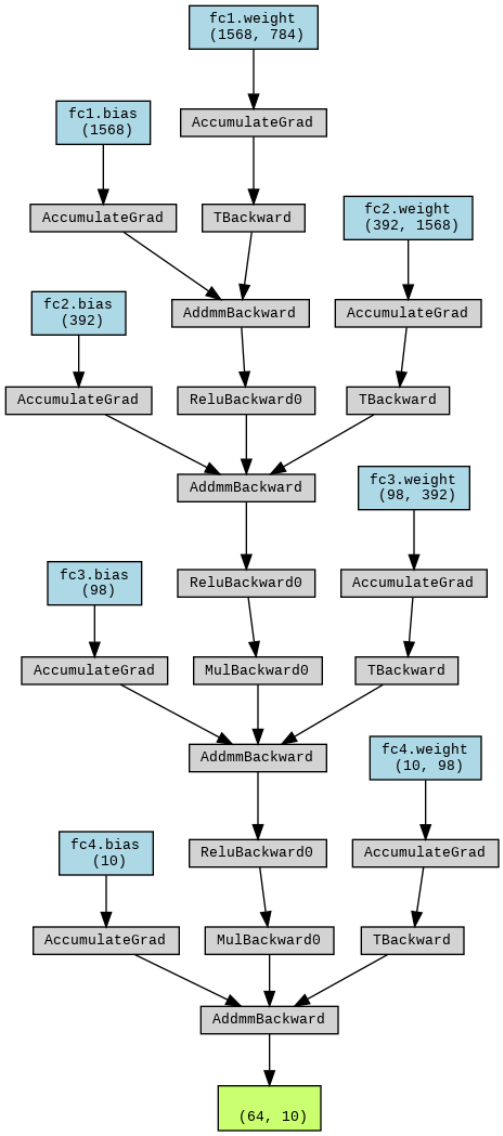
使用的数据集是本来的 MNIST 数据集(https://www.kaggle.com/oddrationale/mnist-in-csv). 数据的例子:



2 模型结构和分析

2.1 MLP

第一个模型是一个 MLP。下面是它的结构图。

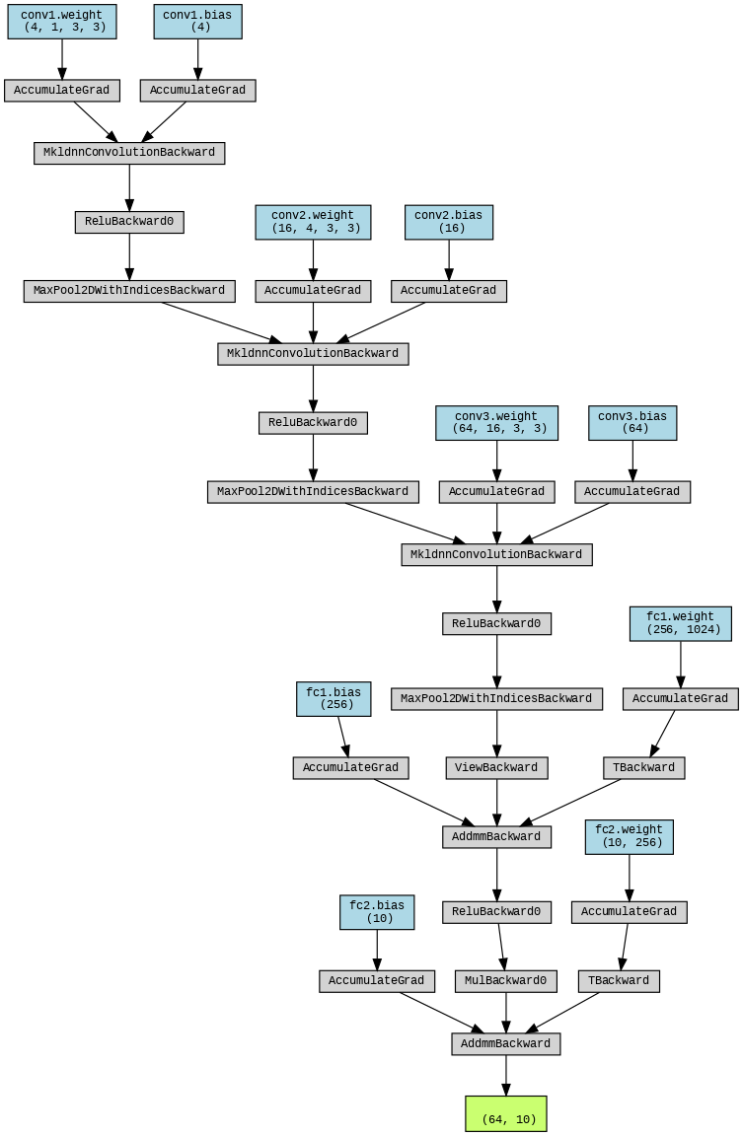


实验提供的 MNIST 数据的结构是 28x28x1。所以最开始把他的维度成为 784。第一个全连接层就有 784 个权值。这个是图中的 fc1.weight。把输入的数据跟它相乘之后，就要加上 fc1.bias。
模型的第一层增加了维度成为 1568。后来各层，一步步减少它。最后维度成为 10，因为数据有 10 类。

用的激活函数是 ReLU，因为它的效果好而简单。

2.2 CNN

下面是实施的 CNN 图。

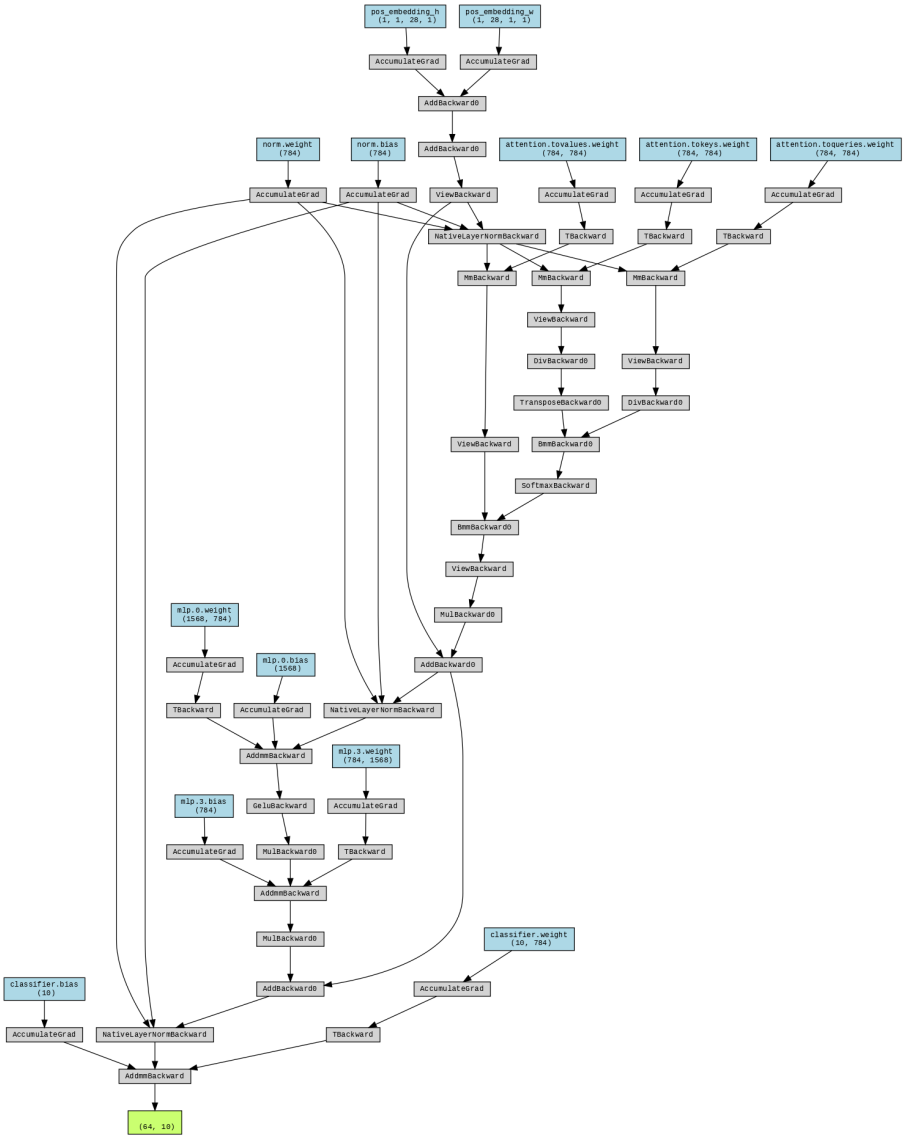


用的层跟 MLP 差不多，可是因为卷积层不是全连接的，参数更少。每个卷积层后，先用 ReLU 的激活函数，才用 MaxPool。用的卷积层的 kernel 都是 3x3 的，padding 都是 1，stride 也都是 1。这样做它不会改变图片维度，所以要用 MaxPool。MaxPool 把一些邻居的数据的最高值为输出。用的 MaxPool 的 kernel 都是 2x2，stride 是 2, 所以输出的图片维度是一半。

然后，每个卷积层的深度是更高，最后到 64 (conv3)。那时候模型对数据的了解已经很深，所以把数据成为一维（ViewBackward）。然后用两个 fc 就输出 (64,10) 的张量。(64 是我用的 batch size)

2.3 VIT

因为好奇，也实施了一个 Vision Transformer[1]。它用的算法是注意力。下面是它的图。



[1]这个模型类是谷歌提出的 https://openreview.net/pdf?id=YicbFdNTTy。我把它简化了一点，因为 MNIST 数据比较小。

因为注意力 (attention) 不看位置，需要加上一种位置信息 (positional embedding)。所以最开始加上两个参数。他们代表照片的高度和宽度。然后从那 3 个 (784,784) 的权值到 MulBackward0 就是注意力机制。长线是 residual 的连接。他们解决深度学习的梯度消失问题，因为用的模型只有一层注意力，可能不需要 residual 的连接，但它们也不会使模型恶化。注意力后，transformer 模型有一个很深的 MLP。因为它参数也变得很多。

3 实验结果与比较

3.1 结果分析

下面是实验结果的表格。最佳精度为 99.11%。

表 1: 实验结果

模型	**参数**	**Epochs**	**Val 准确率**	**Test 准确率**	**描述**
MLP-Base	1885K	3	0.975333	0.976200	
CNN-Base	275K	14	0.987000	0.988400	
VIT-Base	4314K	8	0.975000	0.976400	
MLP-Dropout	1885K	8	**0.979833**	**0.980300**	
CNN-Dropout	275K	11	0.988333	0.990400	
VIT-Dropout	4314K	9	0.976000	**0.977600**	
MLP-Dropout-Zeros	1885K	11	0.112333	0.113500	
CNN-Dropout-Zeros	275K	11	0.112333	0.113500	
VIT-Dropout-Zeros	4314K	15	0.926500	0.925500	
MLP-Dropout-Uniform-(0-1)	1885K	15	0.143667	0.145800	
CNN-Dropout-Uniform-(0-1)	275K	15	0.218000	0.222800	
VIT-Dropout-Uniform-(0-1)	4314K	15	0.959167	0.959100	
MLP-Dropout-Uniform-(-1,1)	1885K	15	0.125000	0.118800	
CNN-Dropout-Uniform-(-1,1)	275K	15	0.793167	0.796400	
VIT-Dropout-Uniform-(-1,1)	4314K	15	0.965000	0.965700	
MLP-Dropout-Uniform-(-0.1,0.1)	1885K	15	0.974167	0.972700	
CNN-Dropout-Uniform-(-0.1,0.1)	275K	15	0.988333	0.989200	
VIT-Dropout-Uniform-(-0.1,0.1)	4314K	7	0.972500	0.969500	
MLP-Dropout-Xavier	1885K	9	0.970167	0.968800	
CNN-Dropout-Xavier	275K	15	0.989167	0.987100	
VIT-Dropout-Xavier	4314K	10	**0.976500**	0.976500	
CNN-Dropout-BatchNorm	275K	13	**0.991667**	**0.991100**	最佳

前六个模型都使用高斯分布初始化，方差为 0.02。CNN 模型一般是最

好的。所有实验都使用 Adam 优化器，学习率为 1e-4。我最初用 SGD 优化器，然而 Adam 优化器比 SGD 容易调整。学习率越高，模型的训练和过拟合就越快。为了让模型看每张图片 5-15 次左右（即 5-15 个 epochs），1e-4 比较合适。Batch Size 是 64。VIT 模型使用的参数最多。注意力机制对深的模型更合适。实施的 VIT 只用了一个注意力层，也许更多会让它超过 CNN。MLP 的层数比 CNN 少，但参数更多，因为 MLP 的层都是全连接的。

最后，我尝试着改进我的 CNN 的架构。加了归一化（BatchNorm2d 和 1d 模块）之后，我达到了 99.11% 的最佳 test 准确率。

3.2 思考

- **合适停止：** 为了确保模型在合适停止，每个 epoch 之后用验证数据进行评估它。根据验证损失，我保存了最佳模型，并在表中报告了损失。用验证数据可以测试模型的泛化能力，如果用固定迭代次数，这是不可以的。使用验证数据的问题是，可以使用更少的数据进行训练（我使用 10% 的数据进行验证）。解决这个问题的办法可以是首先使用验证数据来找到最佳长度，然后用全部数据进行相同长度的训练。

- **初始化：** 我尝试对模型权值进行均匀分布、高斯分布和 xavier 的初始化。偏差的初始为零。如果所有的权值都是 0，所有的模型都很差，因为把权值与数据相乘时，数据也变成了 0。将所有权值初始化为正值（表中的 Uniform-(0-1)）对完全依赖 ReLU 非线性的模型来说是不好的。因为从 0 到 $+\infty$，ReLU 就是线性的，所以如果所有的权值是正值就没有了非线性。VIT 模型在注意力机制中包括 softmax 非线性，它的 Uniform(0-1) 版因此比别的模型更好。从-1 到 1 的初始化对所有模型都好多。将范围减小到-0.1，0.1 更好，因为它使权值更接近 ReLU 中的非线性（0）。Xavier 的初始化好一点。用高斯分布而不是均匀分布进行初始化就是最好的（表中的前六个模型）。

- **过拟合：** 过拟合通过合适停止和 dropout 模块防止。dropout 随机地停用节点。这就要求网络减少对单一节点的依赖。dropout 的概率是 0.1。我在验证和测试过程中停用了它。

- **卷积层 vs 全连接：** 全连接层有两个问题。它们需要大量的参数，因为每个节点都要与每个后续节点相乘。它们对输入的顺序也是不变的。这对图像数据来说是个问题，因为像素的位置很重要。卷积层解决了这两个问题。kernel 的权值是重复使用的，因此需要更少的参数。如果 kernel 小于图像的维度，每个 kernel 也只看图像的一部分。这使得模型能够理解位置。VIT 模型通过 positional embedding 来解决位置

问题 (但 positional embedding 的位置信息比 CNN 还差)。对于参数问题，可以使用稀疏注意机制。

3.3 心得体会

- **VIT 的 positional embedding 要是二维的:** 之前我用 BERT 等模型将注意力机制应用于语言。由于语言只有一个方向，只要把输入数据加上一个简单的一维张量为 positional embedding。但图像有两个方向，所以我加上高度和宽度的两个 positional embedding[2]。我认为将注意力机制调整到二维是一个有趣的研究课题。而一旦我们有了更快的芯片，也许为三维的视频再需要调整它。

- **GPU 和 CPU 的准确率不一样:** 在我的实验中，我意识到，尽管设置了相同的 seed，但用 CPU 和 GPU 的结果有一点不同。我了解到，PyTorch 在 GPU 上的一些操作使用了不同的算法，因此结果不同。上面的表中的所有结果都是用 GPU 的。

[2]原始的 VIT 模型先把照片减少才加上一个 positional embedding。在我的实验中，先加上 positional embedding 更好 - https://github.com/google-research/vision_transformer/blob/master/vit_jax/models.py