

生物大数据分析第三次作业

任务 1: 利用 CNN 算法完成 final_symbols_split_ttv 数据集的分类

数据集介绍: 该数据集包含 50000 多张图片, 标签分别是 0-9 的数字和四则运算符号共 14 类。图片数据均是分辨率为 150*150 的灰度图像, 目标是搭建卷积神经网络模型实现多分类问题。

数据集已经被划分为 train、test 和 val 三个子数据集, 分别用作训练、测试和验证。在每个子数据集文件夹中, 使用子文件夹存储了每个类别的图像, 共 14 个类别。

1. 使用CNN算法

在我构建的CNN网络中, 我使用了三个搭配的卷积层(kernel大小为10)和pool层(kernel大小分别为2, 2, 5), 随后使用了两个全连接层。而卷积层通道数分别为1-6, 6-12, 12-24, 全连接层的通道数分别是384-70, 70-14。

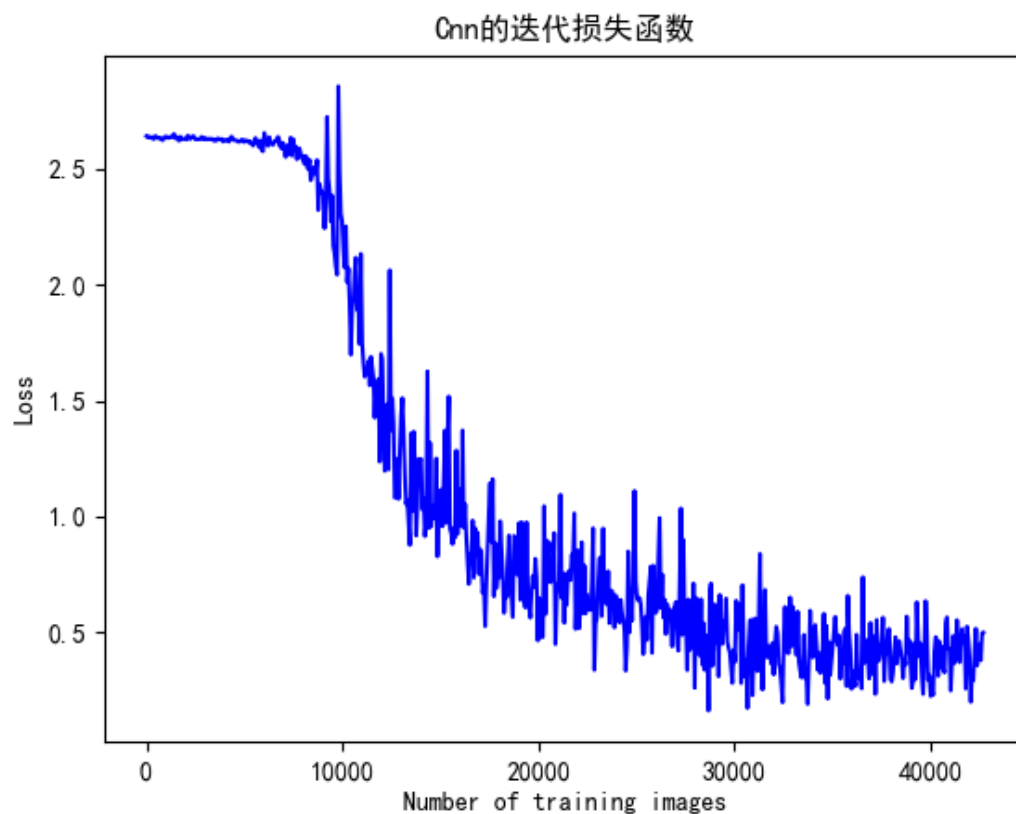
对于激活函数, 除最后一层使用了log_softmax()之外, 其余均使用ReLU()函数。

而对于损失函数和优化器的选择, 我使用了nll_loss损失函数(专用于分类任务的损失函数), 和学习率为0.1, momentum为0.5的SGD优化器。

在训练过程中, 每一次迭代后, 程序会记录并输出当前训练进展情况和当前迭代的计算的损失函数值, 如下图所示:

```
Train Epoch: 1 ----41024/42749 (95.96%)----Loss: 0.431220
Train Epoch: 1 ----41088/42749 (96.11%)----Loss: 0.580496
Train Epoch: 1 ----41152/42749 (96.26%)----Loss: 0.385437
Train Epoch: 1 ----41216/42749 (96.41%)----Loss: 0.337769
Train Epoch: 1 ----41280/42749 (96.56%)----Loss: 0.534053
Train Epoch: 1 ----41344/42749 (96.71%)----Loss: 0.820157
Train Epoch: 1 ----41408/42749 (96.86%)----Loss: 0.241018
Train Epoch: 1 ----41472/42749 (97.01%)----Loss: 0.611591
Train Epoch: 1 ----41536/42749 (97.16%)----Loss: 0.539750
Train Epoch: 1 ----41600/42749 (97.31%)----Loss: 0.596481
Train Epoch: 1 ----41664/42749 (97.46%)----Loss: 0.294032
Train Epoch: 1 ----41728/42749 (97.60%)----Loss: 0.463705
Train Epoch: 1 ----41792/42749 (97.75%)----Loss: 0.298847
Train Epoch: 1 ----41856/42749 (97.90%)----Loss: 0.384957
Train Epoch: 1 ----41920/42749 (98.05%)----Loss: 0.481104
Train Epoch: 1 ----41984/42749 (98.20%)----Loss: 0.624306
Train Epoch: 1 ----42048/42749 (98.35%)----Loss: 0.450722
Train Epoch: 1 ----42112/42749 (98.50%)----Loss: 0.352802
Train Epoch: 1 ----42176/42749 (98.65%)----Loss: 0.688960
Train Epoch: 1 ----42240/42749 (98.80%)----Loss: 0.475226
Train Epoch: 1 ----42304/42749 (98.95%)----Loss: 0.591929
Train Epoch: 1 ----42368/42749 (99.10%)----Loss: 0.431414
Train Epoch: 1 ----42432/42749 (99.25%)----Loss: 0.414986
Train Epoch: 1 ----42496/42749 (99.40%)----Loss: 0.594716
Train Epoch: 1 ----42560/42749 (99.55%)----Loss: 0.269960
```

最终绘制得到Loss图像为:

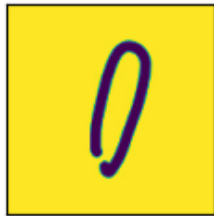


最终还使用了test集中的数据对训练得到的模型进行了性能测试（尤其是预测准确率），得到的最终准确率为：

```
Test accuracy :3850/4352(0.8847)
Test accuracy :3907/4416(0.8847)
Test accuracy :3963/4480(0.8846)
Test accuracy :4019/4544(0.8845)
Test accuracy :4074/4608(0.8841)
Test accuracy :4129/4672(0.8838)
Test accuracy :4185/4736(0.8837)
Test accuracy :4241/4800(0.8835)
Test accuracy :4291/4864(0.8822)
Test accuracy :4348/4928(0.8823)
Test accuracy :4405/4992(0.8824)
Test accuracy :4463/5056(0.8827)
Test accuracy :4518/5120(0.8824)
Test accuracy :4575/5184(0.8825)
Test accuracy :4635/5248(0.8832)
Test accuracy :4696/5312(0.8840)
Test accuracy :4733/5376(0.8804)
Test overall accuracy:4733/5356(0.88)
```

同时，最终还对预测结果进行了展示，如图所示：

Prediction: the symbol is 0



Prediction: the symbol is *



Prediction: the symbol is -



Prediction: the symbol is 3



Prediction: the symbol is 0



Prediction: the symbol is 9



具体细节可见代码cnn-number.py

2. 使用MLP实现分类任务

在该任务中，我搭建了4层MLP，它们的dimension size分别为：22500-1000，1000-225，225-25，25-14。

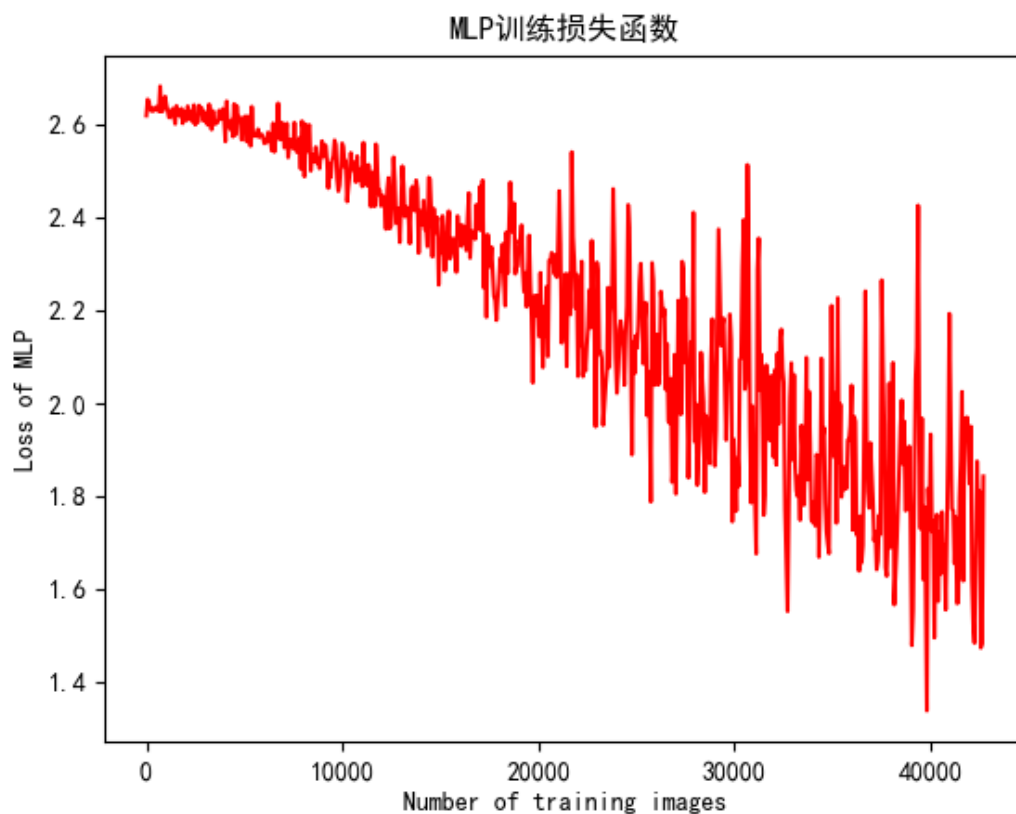
这里我使用的激活函数和CNN中一致：除最后一层使用了log_softmax之外，其余均使用ReLU函数。

同时此处我是用的损失函数和优化器同样和CNN中一致，分别是：nll_loss和学习率为0.1的SGD。

此处程序运行中的输出和CNN任务中仍然一致。比如，程序在训练中会输出每迭代一次的loss情况：

```
MLP Train Iteration: 1 ----64/42749 (0.15%)----Loss: 2.660316
MLP Train Iteration: 1 ----128/42749 (0.30%)----Loss: 2.639115
MLP Train Iteration: 1 ----192/42749 (0.45%)----Loss: 2.642941
MLP Train Iteration: 1 ----256/42749 (0.60%)----Loss: 2.620261
MLP Train Iteration: 1 ----320/42749 (0.75%)----Loss: 2.644115
MLP Train Iteration: 1 ----384/42749 (0.90%)----Loss: 2.627166
MLP Train Iteration: 1 ----448/42749 (1.05%)----Loss: 2.594782
MLP Train Iteration: 1 ----512/42749 (1.20%)----Loss: 2.594103
MLP Train Iteration: 1 ----576/42749 (1.35%)----Loss: 2.657781
MLP Train Iteration: 1 ----640/42749 (1.50%)----Loss: 2.640269
```

同时会给出MLP训练时迭代时的损失函数图像：



同样，最终使用了测试集数据对模型性能进行验证：

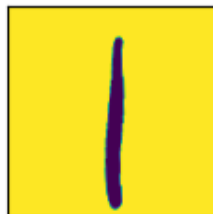
```
MLP accuracy :2290/4736(48.3530%)
MLP accuracy :2318/4800(48.2917%)
MLP accuracy :2351/4864(48.3347%)
MLP accuracy :2381/4928(48.3157%)
MLP accuracy :2407/4992(48.2171%)
MLP accuracy :2441/5056(48.2793%)
MLP accuracy :2468/5120(48.2031%)
MLP accuracy :2492/5184(48.0710%)
MLP accuracy :2521/5248(48.0373%)
MLP accuracy :2555/5312(48.0986%)
MLP accuracy :2578/5376(47.9539%)
MLP Overall accuracy: 48.1329%
```

最终也展示了部分预测结果：

Prediction: the symbol is 1



Prediction: the symbol is 1



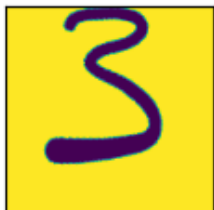
Prediction: the symbol is 9



Prediction: the symbol is 1



Prediction: the symbol is 2



Prediction: the symbol is *



相关实现细节可见mlp-number.py文件。

3. 总结

根据上述列举的结果，能够发现，我所训练的CNN分类性能明显优于MLP。CNN预测准确率达到89%左右，然而MLP预测准确率仅仅在50%左右。我认为实际上这样结果的出现和任务本身的性质和算法特性有关。本任务是关于图像识别的分类任务，实际上CNN算法本身就是用来进行该任务而发明的（卷积、pool等概念解决了从图像中提取数据这一问题）；然而MLP仅仅只是一个泛化的网络，本身对于该任务无任何针对性，自然性能不高，但也展示了MLP本身较强针对不同任务更好的延展性：即对于各种任务都能够获得不错的结果。

在训练过程中，主要超参数涉及到网络层数的设计、优化器、激活函数、损失函数及其参数的选择等问题。我认为第一步仍然是考虑前人经验，第二部则需要不断的进行实验，从而选取相应更优的参数。实际上，我在解决本问题时，就是同时进行了两项努力，最终才能使得CNN性能达到较好的水平。

任务二：利用 RNN 算法完成 PDB_protein_sst3 数据集的分类

数据集介绍：

该数据集包含 16000 多个蛋白质序列数据，标签 sst3 为序列中每个残基对应的二级结构分类，包括 H（各种 helix 结构）、E（ β -strand 和 β -bridge）和 C（loop 等其他不规则结构）三类。蛋白序列信息储存在 seq 列中；标签储存在 sst3 列中，蛋白序列的每一个残基对应一个标签。目标是使用循环神经网络模型解决给定蛋白序列的残基级别的二级结构三分类问题。这个数据集没有划分训练集和测试集，需要同学们自行划分并训练模型。

1.使用RNN算法

这里构建的网络由一个两层的RNN和一个全连接层组成。

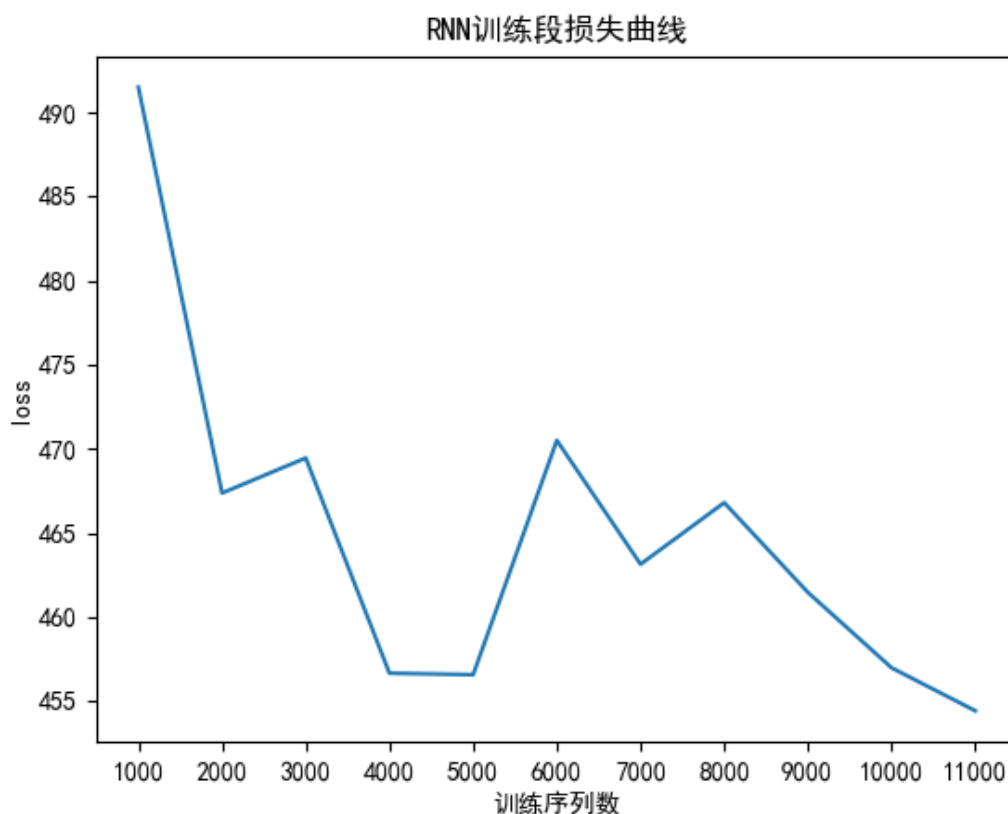
RNN参数为: inputsize:21;hidden_dim:12;全连接层参数为: inputsize:12;outputsize:3。

由于损失函数已经被指定, 这一仅仅指出分别对RNN和全连接层使用了RMSprop和SGD两个优化器。

在训练过程中, 每一次迭代后, 程序会记录并输出当前训练进展情况和当前迭代的计算的损失函数值, 同时得到最终的预测精度, 如图所示:

```
RNN:Already trained sequence:1000-----Loss:446.3703
RNN:Already trained sequence:2000-----Loss:473.3822
RNN:Already trained sequence:3000-----Loss:457.2271
RNN:Already trained sequence:4000-----Loss:463.2751
RNN:Already trained sequence:5000-----Loss:453.9478
RNN:Already trained sequence:6000-----Loss:459.5101
RNN:Already trained sequence:7000-----Loss:459.3282
RNN:Already trained sequence:8000-----Loss:454.9755
RNN:Already trained sequence:9000-----Loss:457.5326
RNN:Already trained sequence:10000-----Loss:465.8583
RNN:Already trained sequence:11000-----Loss:475.7498
Overall accuracy for RNN is:0.3723
```

同时绘制了每1000个序列的损失函数值图像:



此外, 程序能够输出真实结果和预测结果:

[illegible]

具体细节可见model.py和run.py文件，具体步骤还包括了序列转换张量，随机划分训练组和测试组、输出张量到文件中以检查结果等。

2.使用MLP算法

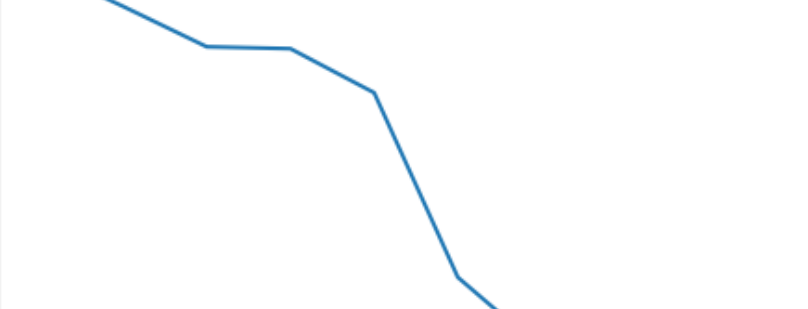
这里使用了三层网络进行分类任务。网络的size分别为：21-70，70-20，20-3。

使用到的激活函数包括: ReLU、Sigmoid等; 使用的损失函数为: MSELoss; 使用的优化器为: 学习率为 0.003的SGD。

这里所有输出和1中相互对应，包括了训练时损失函数值和总精确率：

```
MLP:Already trained sequence:1000-----Loss:127.3178
MLP:Already trained sequence:2000-----Loss:125.4256
MLP:Already trained sequence:3000-----Loss:123.4700
MLP:Already trained sequence:4000-----Loss:123.3784
MLP:Already trained sequence:5000-----Loss:121.2463
MLP:Already trained sequence:6000-----Loss:112.3153
MLP:Already trained sequence:7000-----Loss:108.8435
MLP:Already trained sequence:8000-----Loss:102.4026
MLP:Already trained sequence:9000-----Loss:97.1045
MLP:Already trained sequence:10000-----Loss:97.2924
MLP:Already trained sequence:11000-----Loss:97.5136
Overall accuracy for MLP is:0.3537
```

损失函数图像:



训练序列数	loss
1000	127.5
3000	123.5
4000	123.5
5000	121.0
6000	112.5
7000	109.0
8000	102.5
9000	97.5
11000	97.8

[illegible]

3. 总结

对于该问题，由于未能找到相关资料，因此所有超参数的选择均是由我进行手动实验而选择的。但是过程较为繁琐复杂，同时也未能够得到较好的效果，可能需要进一步进行参数搜索等编程工作。