MLP for MNIST 实验报告

本实验尝试了多种超参数配置下的 MLP 在 MNIST 数据集上的效果，并尝试使用自动编码技机对输入进行预处理。

本实验使用 Keras 编写神经网络代码，使用 tensorflow 作为 backend。数据集分割方面直接调用老师给的数据集获取函数，训练集被分割成 50000 规模的训练集和 10000 规模的验证集。

github 地址：https://github.com/Erutan-pku/DNN_TA_PKU/tree/master/mnist_MLP

- 多种超参数配置下 MLP 在 MNIST 上的效果
  - 基础超参数配置

由于本人精力有限，无力对指数级的超参数组合进行实验与分析。以下对修改超参数的尝试主要是在这个基础超参数配置上进行的单变量或二变量的修改。基础超参数配置的配置与 github 中 keras 上 MLP for MNIST 的配置相同[1]，如表 1 所示。

隐含层层数	2
隐含层神经元个数	512，512
激活函数	relu
正则化	不使用正则化
Dropout	0.2
损失函数	categorical_crossentropy
优化方法	RMSprop
batch_size	128
迭代轮数	30

表 1. 基础超参数配置

评价指标有两个：主评价指标为在 30 轮迭代中，验证集效果最好的那一轮（记为 nd_c）时的测试集准确率，称为 te_acc_c。辅评价指标为测试集效果最好的一轮迭代（记为 nd_b）时的测试集准确率，称为 te_acc_b。

[1] https://github.com/fchollet/keras/blob/master/examples/mnist_mlp.py

我尝试过很多设置，也 run 过 GitHub 上声称可以复现的 keras 示例，但是发现依然不能保证每次运行程序的表现都完全相同。我不清楚这是网络参数随机初始化的问题还是优化过程的问题（我个人倾向于后者，因为前者在固定了 random.seed 与 np.random.seed 之后没理由还搞不定）。所以本报告中提及的实验结果在复现时有小幅度的偏差是完全正常的。

例如基础设置运行 10 遍后，te_acc_c 的范围在 0.9822～0.9838 之间，而 te_acc_b 的范围在 0.9832～0.9848 之间，nd_c 的范围在 19~30 之间，nd_b 的范围在 11～29 之间。这组实验结果也可以看出 test 集合 validation 集在分布上略有偏差，前者往往需要稍多一点的迭代轮数才能达到最好。（当然这也有可能是巧合，毕竟 2 个区间重叠度也很高的）

- 隐藏层数量与隐藏节点个数对效果的影响：

首先尝试每层节点数相同的设置，结果如表 2 所示。每个中的格中数字为 1-te_acc_c%。绿色表明每行或每列最好的一组参数设置，红色的表示全局最好的。比较时，te_acc_c 的如有平分情况发生，则比较 te_acc_b。

节点数 / 层数	64	128	256	512	1024
1	**2.50%**	2.16%	1.82%	**1.64%**	1.70%
2	2.66%	**1.87%**	**1.69%**	1.83%	**1.63%**
3	2.63%	2.06%	1.79%	1.82%	**1.72%**
4	2.66%	1.92%	**1.76%**	1.84%	2.05%
5	2.68%	1.95%	1.90%	**1.88%**	2.04%
6	2.90%	2.23%	**2.04%**	2.57%	2.84%

表 2. 每层节点数相同的设置下隐藏层数量与隐藏节点个数对效果的影响

观察上表可以发现，节点数较大层数适中时，实验效果普遍不会太差。不过总是在层数较少时效果较好，即使每层节点数较少，不知道是不是受到一些如 dropout 等因素的影响。在这组实验中，基础设置的效果最好的设置是 2 个隐层，每层 1024 个节点的情况，基础设置的效果反而比之前 10 遍的结果都要差一点。

同时我还尝试了一些每层节点数不同的设置，如表 3 所示。

隐含层节点数	1-te_acc_c%	隐含层节点数	1-te_acc_c%
512-256-128	1.83%	128-256-512	2.05%
256-128-64	1.69%	64-128-256	2.74%
512-128-32-10	2.09%	512-128-32-8-5	3.01%

表 3. 每层节点数不同时的实验效果

从上表中可以看出，隐含层节点数递减一般要优于隐含层节点数递增的情况。但如果在隐含层将节点数量缩减得过小，也会使最终结果变差。

- 激活函数对效果的影响

修改不同的激活函数，实验结果如表 4 所示。可以发现，在基础超参数设置下，使用不同的激活函数对实验效果的影响很小。

激活函数	1-te_acc_c%
softplus	**1.70%**
relu	1.75%
tanh	1.97%
sigmoid	1.72%
hard_sigmoid	1.78%

表 4. 不同的激活函数对实验效果的影响

- dropout 系数对效果的影响

修改不同的 dropout 系数，实验结果如表 5 所示。可以发现，在基础超参数设置下，使用不同的 dropout 系数对效果对实验效果的影响不大。而且不同的 dropout 系数并没有对模型的收敛速度造成较为显著的影响。

dropout 系数	1-te_acc_c%	nd_c
0	1.85%	29
0.1	1.72%	12
0.2	**1.49%**	22
0.3	1.65%	22
0.5	1.68%	20

表 5. 不同的 dropout 系数对实验效果的影响

- 不同的随机优化算法与学习速率对效果的影响

修改不同的随机优化算法与学习速率，实验效果如表 6 所示。各个随机优化算法的学习率是在基础学习率（keras 的 default 参数）的基础上成倍调整的。基础学习率为：SGD = 0.01，RMSprop = 0.001, Adagrad = 0.01, Adadelta = 1.0, Adam = 0.001, Adamax = 0.002。每格内的格式为：（nd_c ： 1-te_acc_c%）。红色标明了每行的效果最优值。可以发现，对 SGD 而言，默认的学习率有点太小了，随着学习率的增加，实验效果在逐步提升，对其他优化算法而言，默认学习率都还不错。有些算法好像不支持过大的学习率，在学习率设置过大时就不学了。各个随机优化算法在学习率合适的时候效果都不错。但是学习率的修改对于验证集取到最好效果的迭代轮次并没有显著地影响。

倍率 / 算法	1/2	1	2	4	8	16
SGD	30 : 5.49%	30 : 3.84%	30 : 2.64%	30 : 2.12%	29 : 1.77%	**30 : 1.60%**
RMSprop	20 : 1.71%	**28 : 1.59%**	17 : 1.80%	28 : 2.08%	28 : 3.03%	error
Adagrad	29 : 1.67%	**29 : 1.53%**	28 : 1.70%	error	error	error
Adadelta	27 : 1.79%	**29 : 1.59%**	**30 : 1.59%**	15 : 1.67%	28 : 1.70%	21 : 1.61%
Adam	**27 : 1.64%**	25 : 1.65%	**26 : 1.64%**	29 : 2.03%	27 : 2.60%	27 : 4.94%
Adamax	28 : 1.76%	26 : 1.67%	15 : 1.74%	**26 : 1.48%**	25 : 1.92%	30 : 2.55%

表 6. 不同的随机优化算法与学习速率对效果的影响

- 探究自动编码机预训练对 MLP 在 MNIST 上的效果的影响

这部分尝试了几种自动编码机和稀疏自动编码机的配置，不过效果都比较差。（查过，应该不是代码的问题）在本节中全程固定分类器参数。分类器参数如表 7 所示。而 encoder-decoder 过程有部分参数固定，如表 8 所示。参数配置参考 Keras 自动编码机的教程[2]。

隐含层层数	2
隐含层神经元个数	32，32
激活函数	relu
正则化，Dropout	均无
损失函数	categorical_crossentropy
优化方法	RMSprop
batch_size	128
迭代轮数	100

表 7. 本节中全程固定的分类器的参数设置

激活函数	Decoder 最后一层使用 sigmoid，其余为 relu
Dropout	0
优化方法	adadelta
batch_size	256

表 8. encoder-decoder 过程中固定的参数

[2] https://blog.keras.io/building-autoencoders-in-keras.html

监督设置 1：AutoEncoderModel 迭代 100 轮，所有层一起进行监督训练。目标函数为：binary_crossentropy。

监督设置 2：AutoEncoderModel 迭代 50 轮，每层固定上一轮的输出并以此作为监督。每层进行一次监督训练。目标函数为：mean_absolute_error。在另外两个设置下，binary_crossentropy 的效果更好，但是这个设置下，目标函数设成 binary_crossentropy 会导致程序出问题。看到 stackoverflow [3]也有人提同样的问题，但并没有得到解决。这可能是 keras 的一个 bug 吧。

监督设置 3：AutoEncoderModel 迭代 50 轮，每层固定上一轮的输出并以初始的 784 维向量作为监督。每层进行一次监督训练。目标函数为：binary_crossentropy。

实验结果如表 9、10 所示。表 9 中每格内的数据格式为 nd_c：1-te_acc_c%，表 10 中每格内的数据格式为自动编码机（最后一层）的 loss：分类最后恶训练集准确率。综合分析以下两表可以发现，使用自动编码机进行预训练，无论从效率上还是从效果上都没有提升。增加自动编码机的层数对编码效果没有帮助。相对最好的效果出现在编码维度最高的一组表明自动编码机不加效果可能才是最好的。而对照训练集的收敛效果可以发现，导致自动编码机效果不好的原因主要还是因为降维丢失了过多的必要区分信息。

	32	128-64-32	512-128
监督设置 1	87：3.32%	96：3.61%	53：**3.16%**
监督设置 2	84：6.27%	97：9.12%	84：**4.12%**
监督设置 3	64：3.70%	81：3.92%	84：**2.99%**

表 9. 自动编码机预训练在 mnist 上的效果

	32	128-64-32	512-128
监督设置 1	.0997：2.84%	.1070：3.20%	.0760：2.29%
监督设置 2*	.0905：6.06%	6.0398：10.28%	.9057：3.46%
监督设置 3	.1098：2.72%	.1061：3.13%	.0835：2.68%

表 10. 自动编码机降维对训练集收敛效果的影响

尝试过在上面设置的基础上在编码过程中加入正则项（activity_l1(10e-5)）构建稀疏自动编码机，不过在自动编码部分偏差还要大很多，而分类决策的表现几乎随机。在此也就不列了。

- 总结：神经网络的效果当超参数在某个合理范围内时波动不大。但具体最好点在何种参数组合下可以取到，结果比较随机。

[3] http://stackoverflow.com/questions/42264649/keras-binary-crossentropy-has-negative-values