Artificial Neural Network HW1

刘泓尊　2018011446　计 84

Department of Computer Science, Tsinghua University

2020 年 10 月 4 日

目录

1 Running Guide

在 codes 文件夹下运行 *run_mlp.py* 脚本，参数设置与含义如下：

```
usage: run_mlp.py [-h] [--activ {relu,sigmoid,gelu,tanh}] [--lr LR]
                  [--loss {cross_entropy,enclidean,hinge}]
                  [--nlayer {1,2}] [--wd WD] [--mo MO] [--bsz BSZ]
optional arguments:
    -h, --help          show this help message and exit
    --activ {relu,sigmoid,gelu,tanh}
                        activation function, default=relu
    --loss {cross_entropy,enclidean,hinge}
                        loss function, default=cross_entropy
    --nlayer {1,2}      layers of mlp, default=1
    --lr LR             learning rate, default=0.001
    --wd WD             weight decay, default=0.001
    --mo MO             momentum, default=0.9
    --bsz BSZ           batch size, default=200
```

2 Network Structure and Hyperparameters

2.1 1-Layer MLP

网络结构为:

$784 Linear 360 \rightarrow ReLU/Sigmoid/GeLU \rightarrow 360 Linear 10 \rightarrow CELoss/Euclidean/HingeLoss$

权值初始化：$std = 0.01$

超参：$lr = 0.01, weight_decay = 0.001, momentum = 0.9, batchsize = 200 maxepoch = 100$，其余默认

您可以运行如下命令复现我的结果:

```
python run_mlp.py --nlayer 1 --activ relu --loss cross_entropy --lr 0.01 --
    wd 0.001 --mo 0.9
```

2.2 2-Layer MLP

网络结构为:

$784 Linear 512 \rightarrow ReLU/Sigmoid/GeLU \rightarrow 512 Linear 256 \rightarrow ReLU/Sigmoid/Gelu \rightarrow 256 Linear 10 \rightarrow CELoss/Euclidean/HingeLoss$

权值初始化：$std = 0.01$

超参：$lr = 0.01, weight\_decay = 0.001, momentum = 0.9, batchsize = 200, maxepoch = 100$，其余默认

您可以运行如下命令复现我的结果:

```
python run_mlp.py --nlayer 2 --activ relu --loss cross_entropy --lr 0.01 --
    wd 0.001 --mo 0.9
```

2.3 Additional Implementations

实现了 $Tanh$ 层和 $Softmax$ 层，并对 $Tanh$ 层的表现做了测试。

$Softmax$ 层的反向传播过程与 $Sigmoid$ 基本一致。$Tanh$ 层的反向传播实际上是 $\delta \cdot (1 - \tanh(x)^2)$

您可以在 $layers.py$ 中找到对应的代码。

3 Experiment Result

实验环境: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz

3.1 Overview

超参：$lr = 0.01, weight\_decay = 0.001, momentum = 0.9, batchsize = 200, maxepoch = 100$，其余默认。

损失函数选择 $CELoss$，不同激活函数的对比：

Metric	1L+Sigmoid	1L+ReLU	1L+GeLU	2L+Sigmoid	2L+ReLU	2L+GeLU
Train Loss	0.2073	0.0544	0.0609	0.2298	**0.0264**	0.0334
Train Acc	94.76	98.97	98.66	95.59	**99.64**	99.29
Test Loss	0.2159	0.0788	0.0827	0.2349	**0.0603**	0.0656
Test Acc	94.33	97.88	97.64	95.10	**98.17**	97.95
Time(s)/epoch	5.17	**3.47**	9.49	9.93	4.11	18.73

激活函数选择 *ReLU*, 不同损失函数的对比：

Metric	1L+Eud	1L+CE	1L+Hinge	2L+Eud	2L+CE	2L+Hinge
Train Loss	0.0332	0.0544	**0.0009**	0.0166	0.0264	0.0015
Train Acc	98.51	98.97	**99.77**	99.23	99.64	99.49
Test Loss	0.0386	0.0788	0.0041	0.0226	0.0603	**0.0037**
Test Acc	97.58	97.88	98.31	98.27	98.17	**98.41**
Time(s)/epoch	3.91	**3.47**	4.00	3.99	4.11	4.07

3.2 Loss

下面对不同的损失函数、激活函数下的 MLP 的 Loss 进行对比：

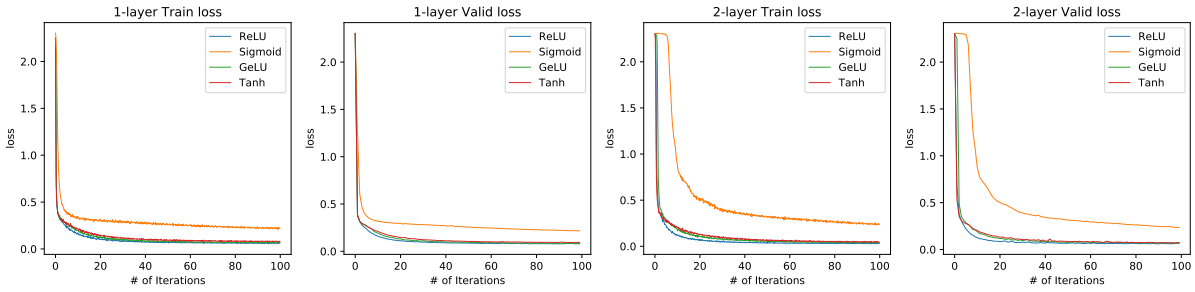

图 1: 不同激活函数的 MLP Train/Valid Loss (loss=CE, lr=0.01, wd=0.001, mo=0.9)

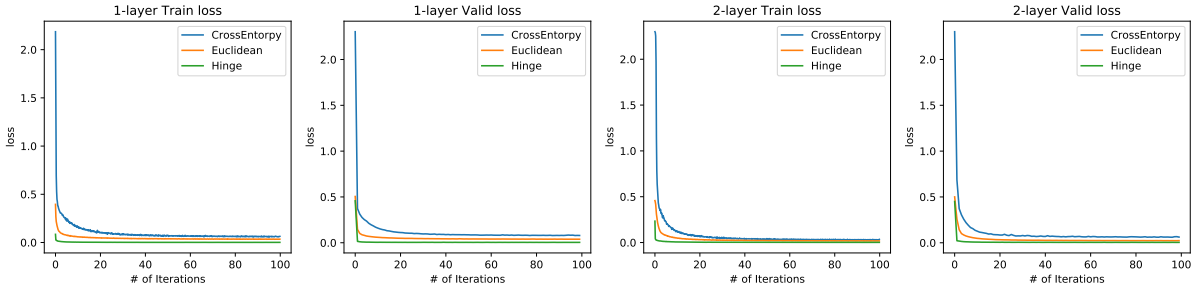

图 2: 不同损失函数的 MLP Train/Valid Loss (loss=CE, lr=0.01, wd=0.001, mo=0.9)

3.3 Accuracy

下面对不同的损失函数、激活函数下的 MLP 的 Accuracy 进行对比：

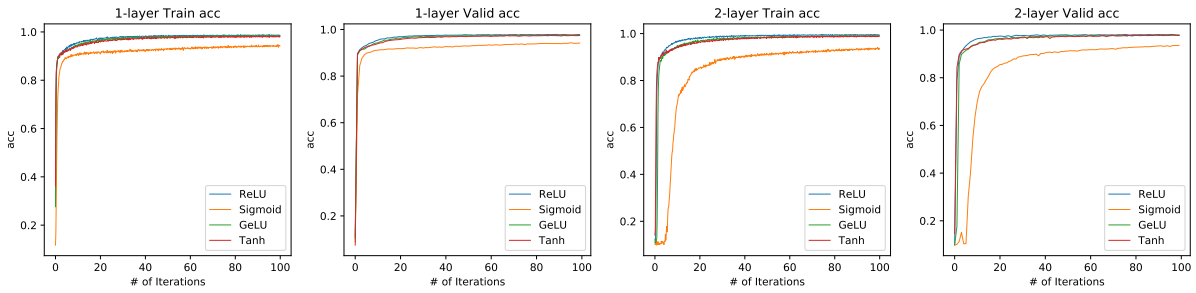

图 3: 不同激活函数的 MLP Train/Valid Acc (loss=CE, lr=0.01, wd=0.001, mo=0.9)

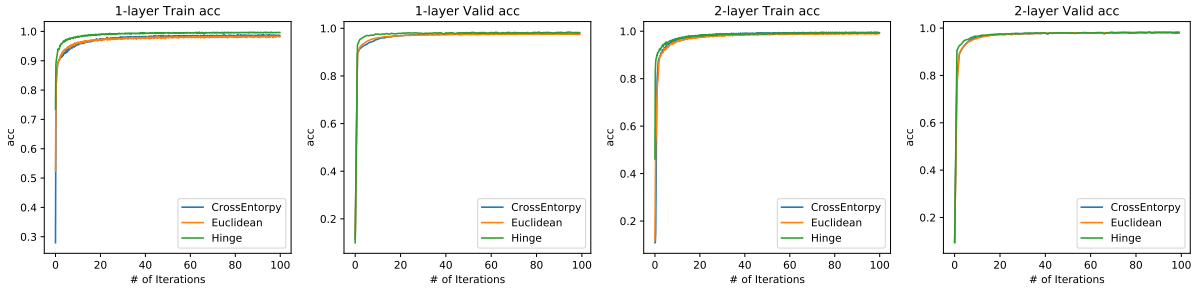

图 4: 不同损失函数的 MLP Train/Valid Acc (loss=CE, lr=0.01, wd=0.001, mo=0.9)

3.4 Other Comparations

我还对比了不同 learning rate, weight decay 和 momentum 的影响。在控制变量下对比了 $lr = 0.1, 0.01, 0.001, 0.0001$，$weightdecay = 0.0, 0.001, 0.01, 0.1$, $momentum = 0.0, 0.1, 0.5, 0.9$ 的影响。

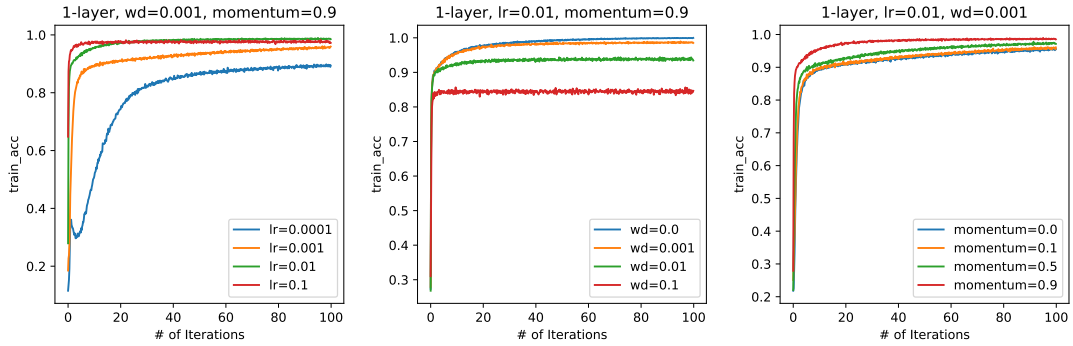

图 5: MLP Train Accuracy

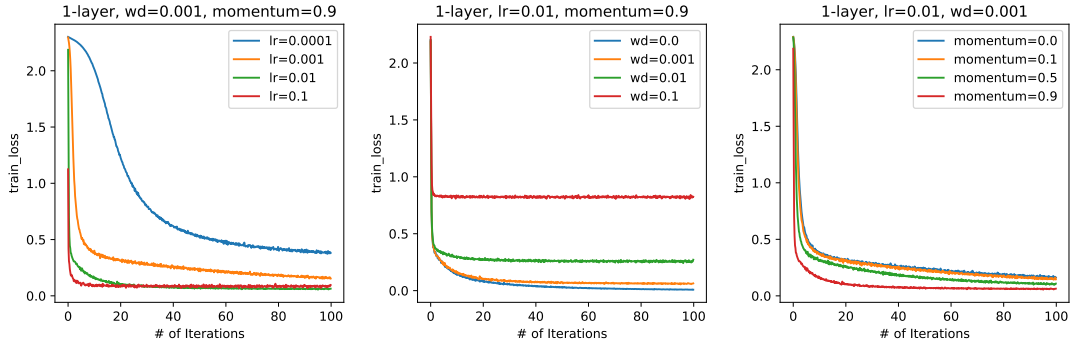

图 6: MLP Train Loss

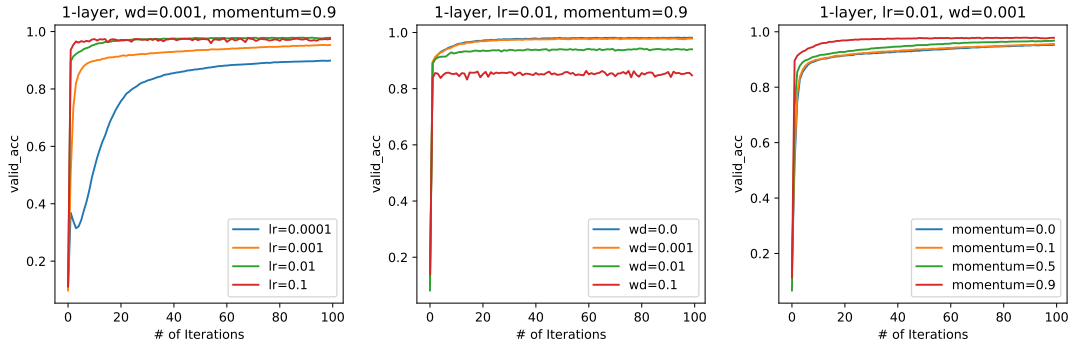

图 7: MLP Valid Accuracy

4 Summary

损失函数：对于不同的激活函数在测试集上的表现，$ReLU > GeLU > Tanh > Sigmoid$. 其中 $ReLU, GeLU, Tanh$ 三者表现相近，$Sigmoid$ 因为存在梯度消失的原因，在 train loss 上收敛很慢，valid loss 的表现也并不好；收敛速度：$GeLU > ReLU > Tanh \gg Sigmoid$；耗时：$GeLU > Sigmoid > ReLU$.

因为 MNIST 数据集的特点，各种激活函数的 Loss 都没有出现明显的过拟合，换成 CIFAR-10 应该会有更直接的体现；

下面对不同激活函数深入讨论一下，从 Valid Loss 曲线1上可以看到，2-layer MLP 使用 $Sigmoid$ 在初始阶段出现了平台，这反映了 $Sigmoid$ 对函数值变化的敏感性，当自变量 $|x| > 4.0$ 时，$Sigmoid$ 接近饱和，几乎不能更新权重。同时因为 $\frac{dSigmoid(x)}{dx} = Sigmoid(x)(1 - Sigmoid(x)) \leq 0.25$，所以

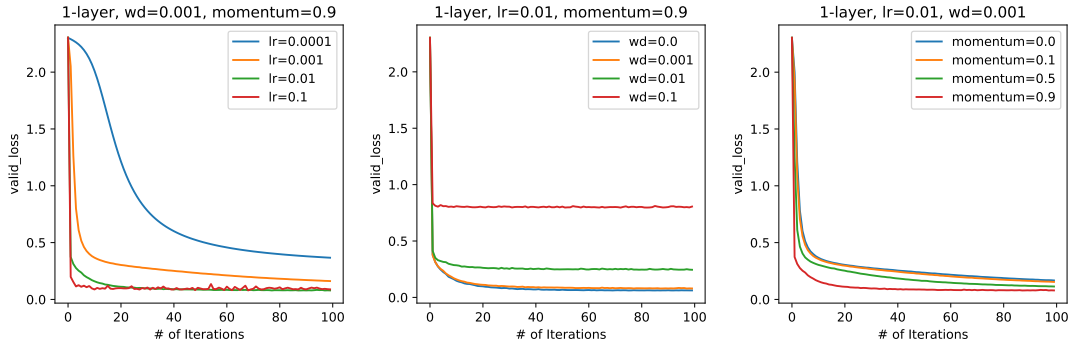

图 8: MLP Valid Loss

随着网络层数的加深会引发梯度消失的问题；

但与之有相似形状的 $Tanh$ 有明显不同，$Tanh$ 是**零均值**的，不存在很严重的梯度消失问题，实际上在特征相差明显时 $Tanh$ 效果会更好；$ReLU, GeLU$ 因为在 $x > 0$ 时导数较大，所以不存在梯度消失的问题，同时因为负半轴几乎为 0，所以保证了神经网络的稀疏性，起到类似 Dropout 的效果，减轻过拟合程度。

激活函数:对于不同损失函数在测试集上的表现，$HingeLoss > CELoss > Euclidean.$ $HingeLoss$ 比较适合做分类，因为它的优化目标是最大化间隔，使得模型有了更好的防止误分类的能力，泛化能力更强；$CELoss$ 是很常用的分类问题损失函数，它可以一直优化下去，效果也非常好；$Euclidean$ 会出现离群点 $(ourliers)$ 的损失占比过大的问题，但是在 MNIST 上依然表现不错。收敛速度：$Hinge > CELoss > Euclidean.$

网络层数：2 层网络表现略好于 1 层网络，但提升不明显。2 层网络增加了参数量，所以训练速度会比 1 层的变慢，但实际测试发现，numpy 的 exp 运算实际上占据了绝大部分时间：当激活函数使用 $ReLU$ 时，2 层 MLP 仅仅比 1 层 MLP 慢一点；使用 $Sigmoid, GeLU, Tanh$ 时，2 层 MLP 的速度和 1 层 MLP 的差距较大，可以推测激活函数层存在指数运算会明显增加训练时间。

学习率：从3.4节中可以看到，更小的学习率意味着更慢的收敛速度，但是往往最终测试准确率较高。但是学习率不宜过低，以免参数难以更新。实验发现，1-layer MLP 使用 $lr = 0.01$ 是较优的。过低的学习率会使得模型难以收敛到很好的准确率。

Weight Decay：权重衰减是在损失上加了 L2 正则化项，使得模型的参数值较小，以保证足够强的泛化能力，减少过拟合。实验发现，1-layer MLP 使用 0.001 的 weight decay 较为合适。较大的 weight decay 会使得模型收敛不到准确率高的点。实际上，weight decay = 0.0 时模型依然表现地很好。此外，weight decay 越小，收敛速度越快。

Momentum: 动量项在数学上实际是对梯度进行了指数移动平均 (Exponential Moving Average)，减轻了梯度的抖动，以保证更快的收敛速度。我的实验发现：momentum 较大 (意味着考虑更多的历史梯度) 时，模型的收敛速度和测试准确率都较高；而较低的 momentum(意味着梯度抖动很大) 会带来较低的准确率，收敛速度也有所下降。