

HW1 MNIST Digit Classification with MLP

计 02 刘明道 2020011156

默认参数

未经说明时， 默认采用以下参数

参数	值
Learning Rate	5×10^{-2}
Batch Size	100
Momentum	0.9
Model Architecture	Linear(784 → 128), Activation(), Linear(128 → 10)
Max Epoch Count	100
Weight Decay	1×10^{-5}
Loss	Cross Entropy Loss
Non-linearity	ReLU
Initialization Std	0.01
Margin Δ in Hinge Loss	0.1
Random Seed	2022

实验环境为

```
Python 3.8.0
numpy 1.22.3
CPU AMD EPYC 7742
```

为了简洁， EuclideanLoss 记为 MSE， SoftmaxCrossEntropyLoss 记为 CE， HingeLoss 记为 Hinge。

隐藏层实验

可视化结果均为随机数种子取 2022 时的结果。

定量结果都是随机数种子取 {2022, 2023, 2024, 2025, 2026} 这 5 次结果的平均值。

单隐藏层的模型结构为

```
Network(
    Linear(name=FC_0, in_num=784, out_num=128, init_std=0.01)
    Activation()
    Linear(name=FC_1, in_num=128, out_num=10, init_std=0.01)
)
```

双隐藏层的模型结构为

```

Network(
    Linear(name=FC_0, in_num=784, out_num=256, init_std=0.01)
    Activation()
    Linear(name=FC_1, in_num=256, out_num=128, init_std=0.01)
    Activation()
    Linear(name=FC_2, in_num=128, out_num=10, init_std=0.01)
)

```

定量结果

为了便于阅读，Loss 和 Accuracy 都放大 100 倍截断到小数点后两位。

Accuracy of Single Hidden Layer

	Train			Test		
	GeLU	ReLU	Sigmoid	GeLU	ReLU	Sigmoid
CE	100.00	100.00	99.98	98.20	98.07	98.09
Hinge	99.93	98.71	98.71	97.87	96.33	96.57
MSE	99.07	98.88	97.98	97.68	97.30	96.94

Accuracy for Double Hidden Layer

	Train			Test		
	GeLU	ReLU	Sigmoid	GeLU	ReLU	Sigmoid
CE	100.0	100.0	99.99	98.44	98.51	98.12
Hinge	99.97	99.96	99.26	98.01	98.12	97.50
MSE	99.84	99.94	98.50	98.29	98.38	97.50

Loss of Single Hidden Layer

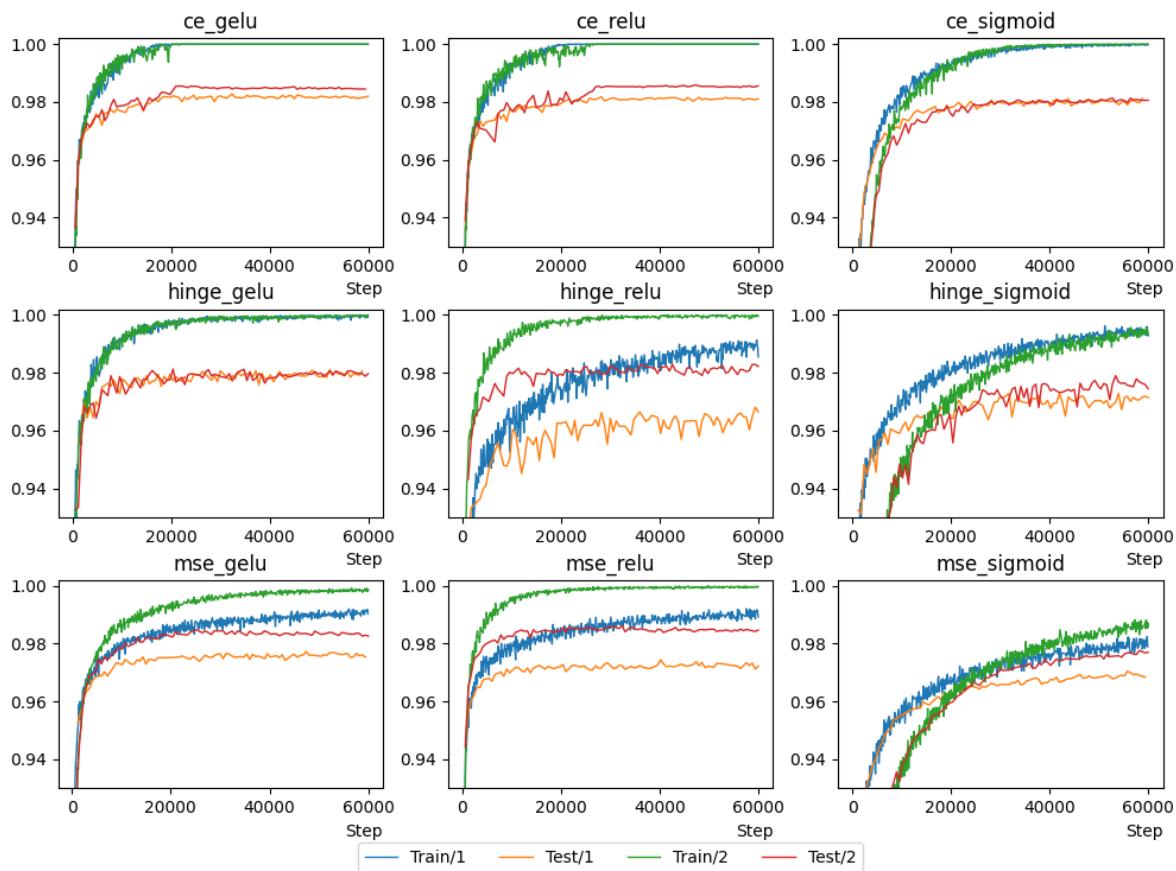
	Train			Test		
	GeLU	ReLU	Sigmoid	GeLU	ReLU	Sigmoid
CE	0.07	0.07	0.66	7.54	7.99	6.08
Hinge	0.01	0.42	0.39	1.22	3.20	1.74
MSE	2.73	2.59	3.38	3.94	4.08	4.10

Loss of Double Hidden Layer

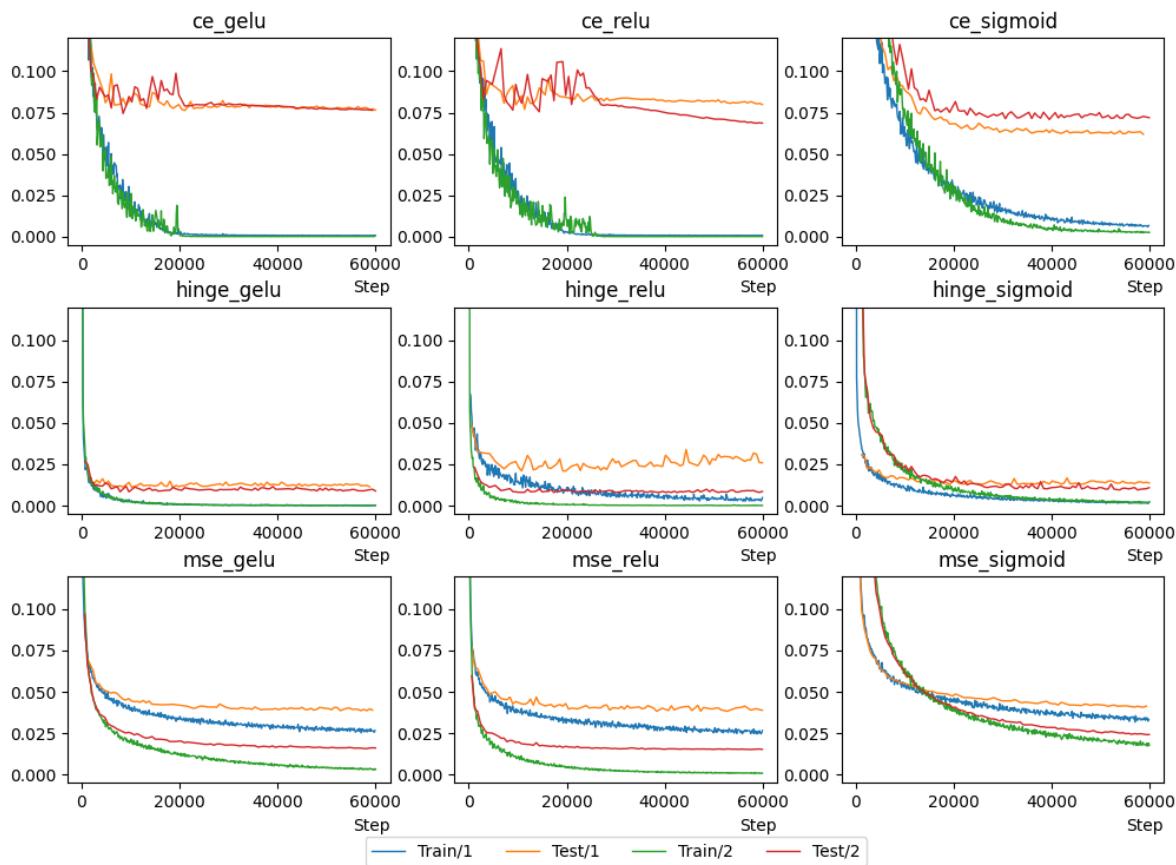
	Train			Test		
	GeLU	ReLU	Sigmoid	GeLU	ReLU	Sigmoid
CE	0.01	0.01	0.26	7.57	7.00	6.79
Hinge	0.01	0.01	0.23	0.93	0.86	1.20
MSE	0.38	0.10	1.89	1.68	1.55	2.54

训练过程

Accuracy

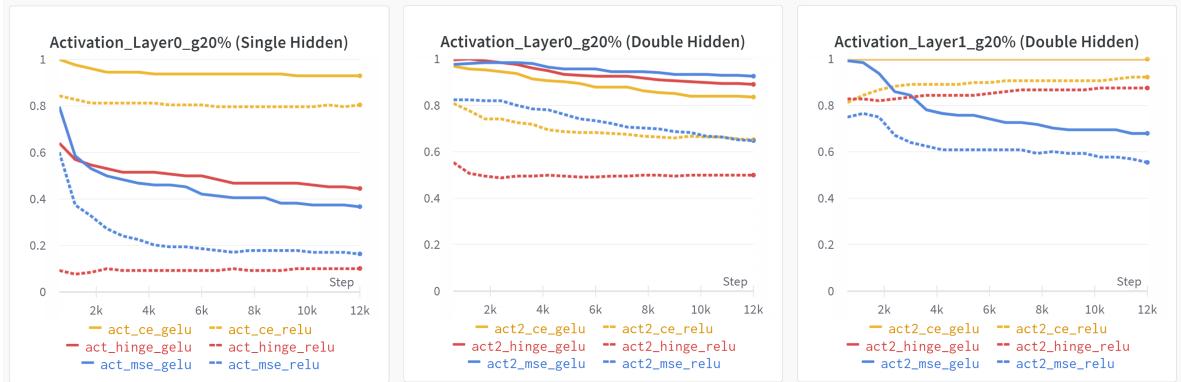


LOSS



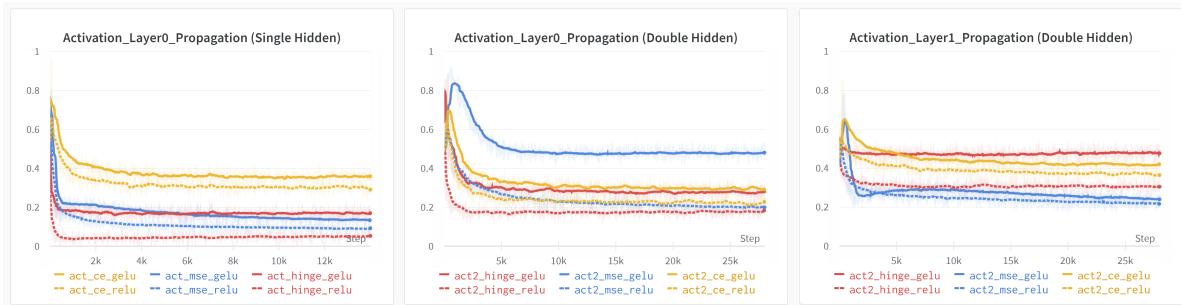
Activation Greater Than 0.2

下图是训练前 20 个 Epoch 中激活（这里指值大于 0）概率大于 0.2 的神经元所占的比例。其中 GeLU, ReLU 分别为实线、虚线；不同颜色表示不同激活函数。Layer0, Layer1 表示不同的非线性层。



Activation in Single Propagation

下图是训练过程中单次传播过程中神经元的激活（这里指值大于 0）的比例。



结果分析

收敛速度

- 就激活函数而言，收敛速度 $\text{GeLU} \approx \text{ReLU} \gg \text{Sigmoid}$
 - 这主要是因为 ReLU, GeLU 在 $(0, +\infty)$ 上导数较大，而 Sigmoid 在输入较大时梯度迅速减小，导致参数更新缓慢，模型收敛较慢。
 - 激活比例
 - 相比 ReLU, GeLU 在 0 附近的一小段负数区间导数非零，这导致更多参数可以被以更高的频率更新。从训练过程中激活概率大于 0.2 的神经元的占比来看，对于相同的损失函数，GeLU 比 ReLU 显著利用了更多神经元 ($+10\% \sim +35\%$)
 - 虽然 GeLU 对神经元的利用率和更新率比 ReLU 大，但是其收敛速度收敛还比 ReLU 略慢（例如在 MSE 中）。
 - 这可能是因为 ReLU 在每次更新的参数占活跃参数的比例稍大（因为 ReLU 活跃参数比 GeLU 更少），因此每次参数更新都会对模型的造成较大影响，因而可能导致收敛较快。（例如 MSE 的 Layer1，两者单次传播过程中激活的神经元数量接近，但 ReLU 的活跃神经元更少）
 - 损失函数
 - CE 在各种激活函数下都能较快收敛；而 MSE, Hinge 则在 Sigmoid 下收敛明显变慢。
 - 这可能是由于在 Softmax 概率的计算过程中，输入进行了指数运算，因此正确类别的输入只要稍微放大，损失函数就会迅速降低。而 Hinge 则要求正确类比错误类高出一个特定的值，

MSE 也要求输入足够接近 1 这个事先确定的值，这些与特定预设值相关的拟合可能需要整个网络参数进行一定程度的缩放，从而造成收敛速度较慢。（这时网络的初始化标准差与 Hinge 的 Margin Δ 或者 MSE 的 One-hot Target 的比值，可能就是一个新的需要调的超参了，虽然我也没调。）

- 网络层数

- 对于 ReLU, GeLU 来说，单隐藏层与双隐藏层收敛速度接近；而对于 Sigmoid 来说，单隐藏层比双隐藏层收敛更快。这可能是因为 Sigmoid 饱和后梯度很小，随网络层数增加，梯度难以有效传播；而 ReLU, GeLU 则没有正向的饱和梯度。

测试集准确率

- 激活函数

- 在单隐藏层中近似为 $\text{GeLU} > \text{ReLU} \approx \text{Sigmoid}$: GeLU 比 ReLU 高 $+0.13\% \sim +0.54\%$ ，而 ReLU 比 Sigmoid 高 $-0.24\% \sim +0.36\%$
- 在双隐藏层中近似为 $\text{ReLU} > \text{GeLU} > \text{Sigmoid}$: ReLU 比 GeLU 高 $+0.07\% \sim +0.11\%$ ，GeLU 比 Sigmoid 高 $+0.39\% \sim +0.88\%$
- 值得注意的是，在双隐藏层的情形下，虽然在所有的激活层 GeLU 都比 ReLU 利用了更多的神经元，但是 GeLU 的准确率却稳定地比 ReLU 低约 0.1%。在本次实验中，GeLU 没有像预期中表现出对 ReLU 的明显优势，这可能是因为本次实验中网络层数较少，神经元死亡问题并不严重，在深度模型中，GeLU 可能会有更明显的优势。

- 损失函数

- 在不同网络层数。不同激活函数下，都表现为 $\text{CE} > \text{Hinge} \approx \text{MSE}$ 。这可能是由于当前的初始化超参，如初始化的标准差、Hinge 的 Margin Δ 等的选择造成的。

- 网络层数

- 下面列出在测试集准确率上，双隐藏层较单隐藏层的提升值

-

	GeLU	ReLU	Sigmoid
CE	0.24	0.44	0.03
Hinge	0.14	1.79	0.93
MSE	0.61	1.08	0.56

- 相比单隐藏层，双隐藏层准确率在各种损失函数-激活函数组合下均有提高（ $+0.03\% \sim +1.79\%$ ）
- 双隐藏层引入了更多参数，同时增加了一层激活函数，提升了模型的非线性性，使得模型可以拟合更为复杂的映射。

计算速度

下表列出了训练 1 个 iteration 的平均用时（毫秒）

	Single			Double		
	GeLU	ReLU	Sigmoid	GeLU	ReLU	Sigmoid
CE	3.06	2.41	2.65	6.90	4.64	5.12
Hinge	3.25	2.34	2.65	6.80	4.66	5.05
MSE	3.50	2.46	2.54	6.70	4.94	5.01

- 对于相同的激活函数来说，不同损失函数的用时非常接近。说明损失函数的计算并非训练速度的决定性因素。

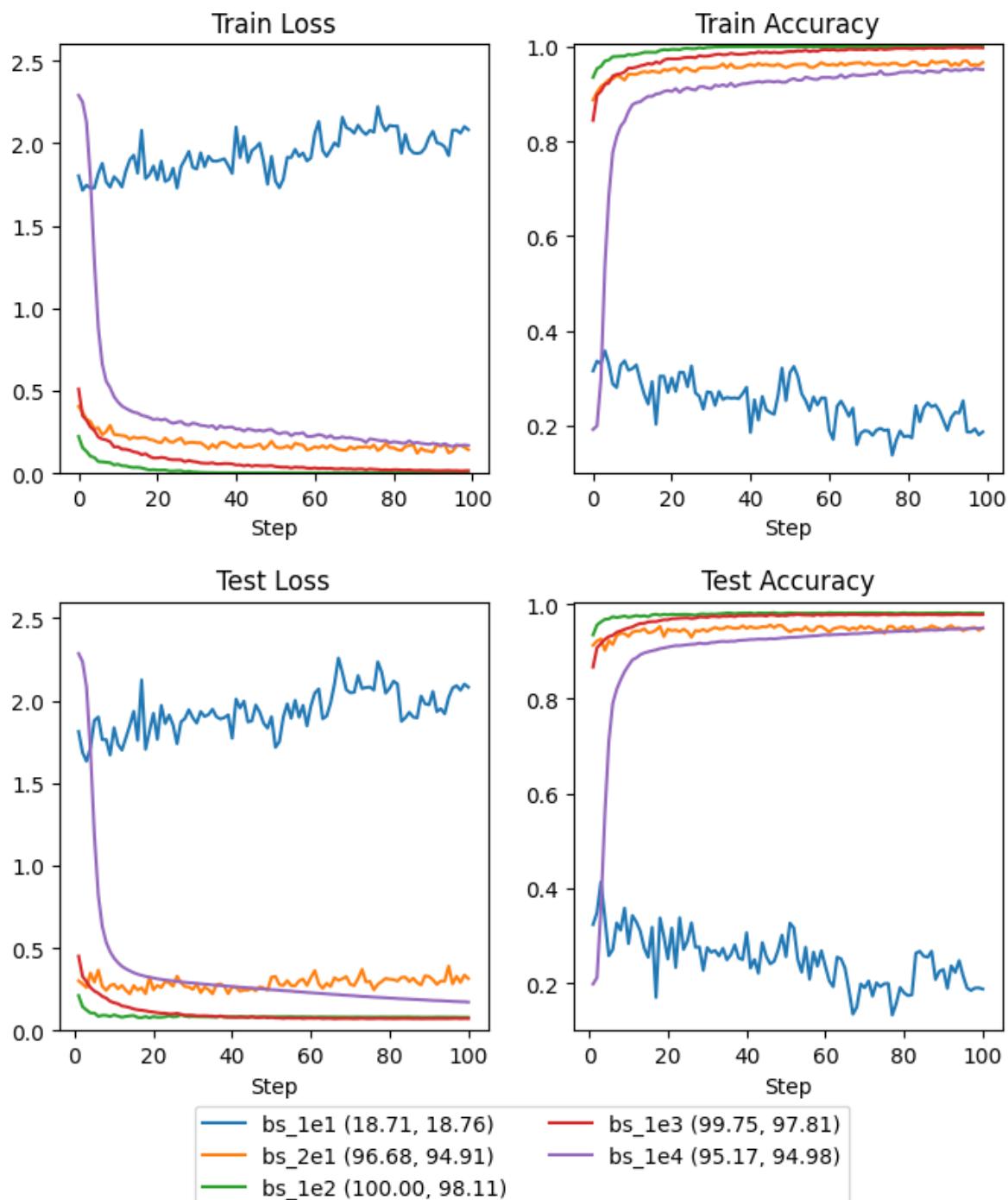
- 对于相同的损失函数，激活函数的用时明显为 GeLU > Sigmoid > ReLU。原因主要是 GeLU 函数在前向和反向传播时的计算都较为复杂，Sigmoid 次之，而 ReLU 的计算则最为简单。这一差异在双隐藏层的实验中则更加显著。
- 层数的增加显著增加了训练时间。随层数增加，网络的参数量从 101k 提升到 235k，在训练中需要更新的参数量显著变多。

超参数实验

在默认参数的基础上，对不同超参数进行调节，以观察其对模型的作用效果。

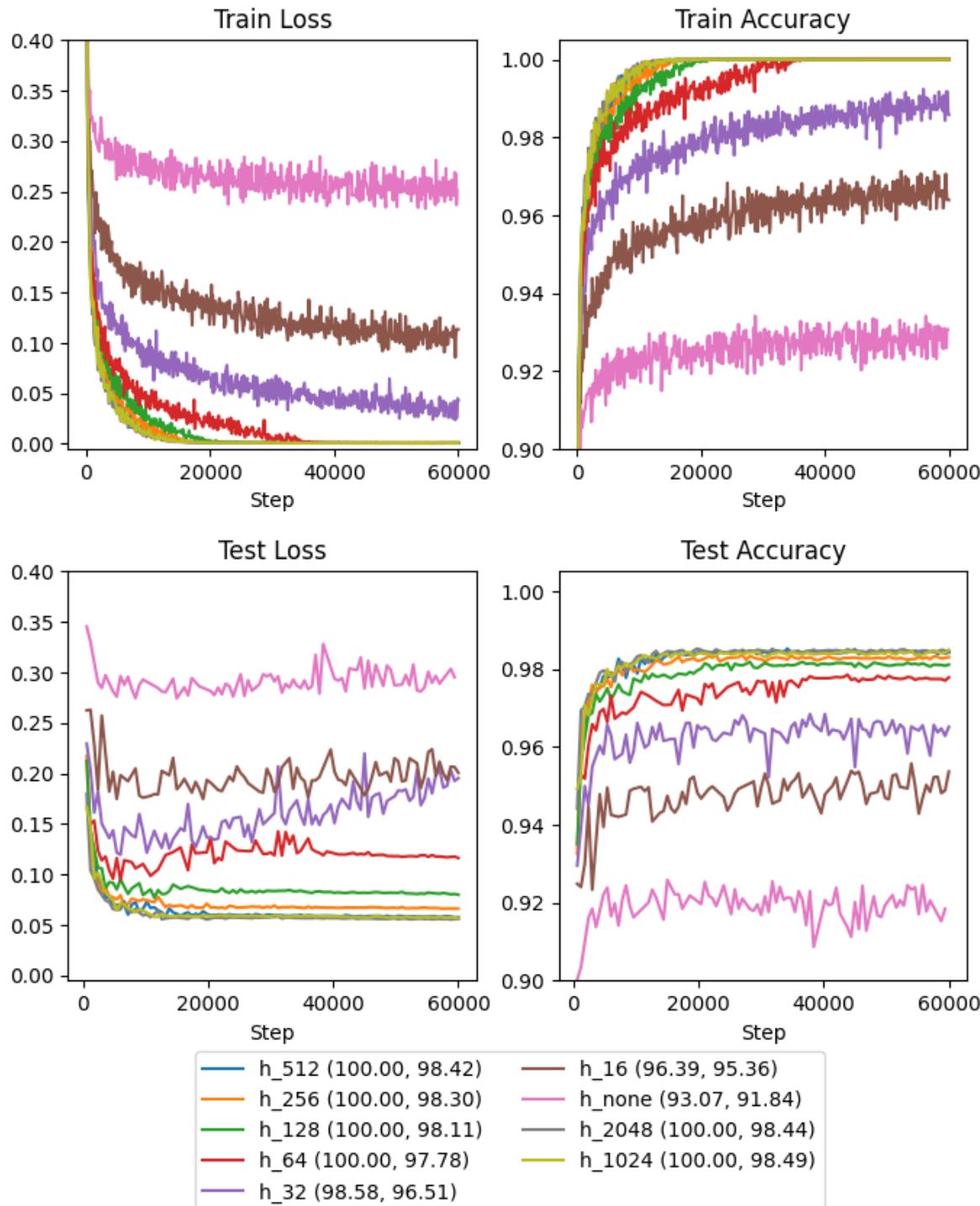
为了简洁，将准确率在图例中表示为（训练集准确率，测试集准确率）；所有结果都是在随机种子取 2022 时得到的。

Batch Size



- 当 batch size 较小时，模型准确率有所下降（如 $B = 20$ ），在训练过程中振荡更为严重，甚至并不收敛（如 $B = 10$ ）。这是由于每次梯度下降时的随机性较大，模型的参数更新也就有了更大的噪声；而在接近极小值点时，模型的参数的振荡仍然较为剧烈，难以收敛到接近最优的位置。
- 当 batch size 非常大（如 $B = 10^4$ ）时，模型收敛较慢，这可能是因为每个 epoch 中参数的更新次数更少了（例如 $B = 10^2$ 时更新 600 次，而 $B = 10^4$ 时仅更新 6 次）。此外，模型在收敛后的准确率也更低。这可能是因为 batch size 较大时，模型每次参数更新的将过于稳定，使得模型容易收敛于准确率较低的局部极小或者在鞍点上移动缓慢，造成准确率较低（如 $B = 10^4$ 时准确率与 $B = 20$ 接近）。

Hidden Size

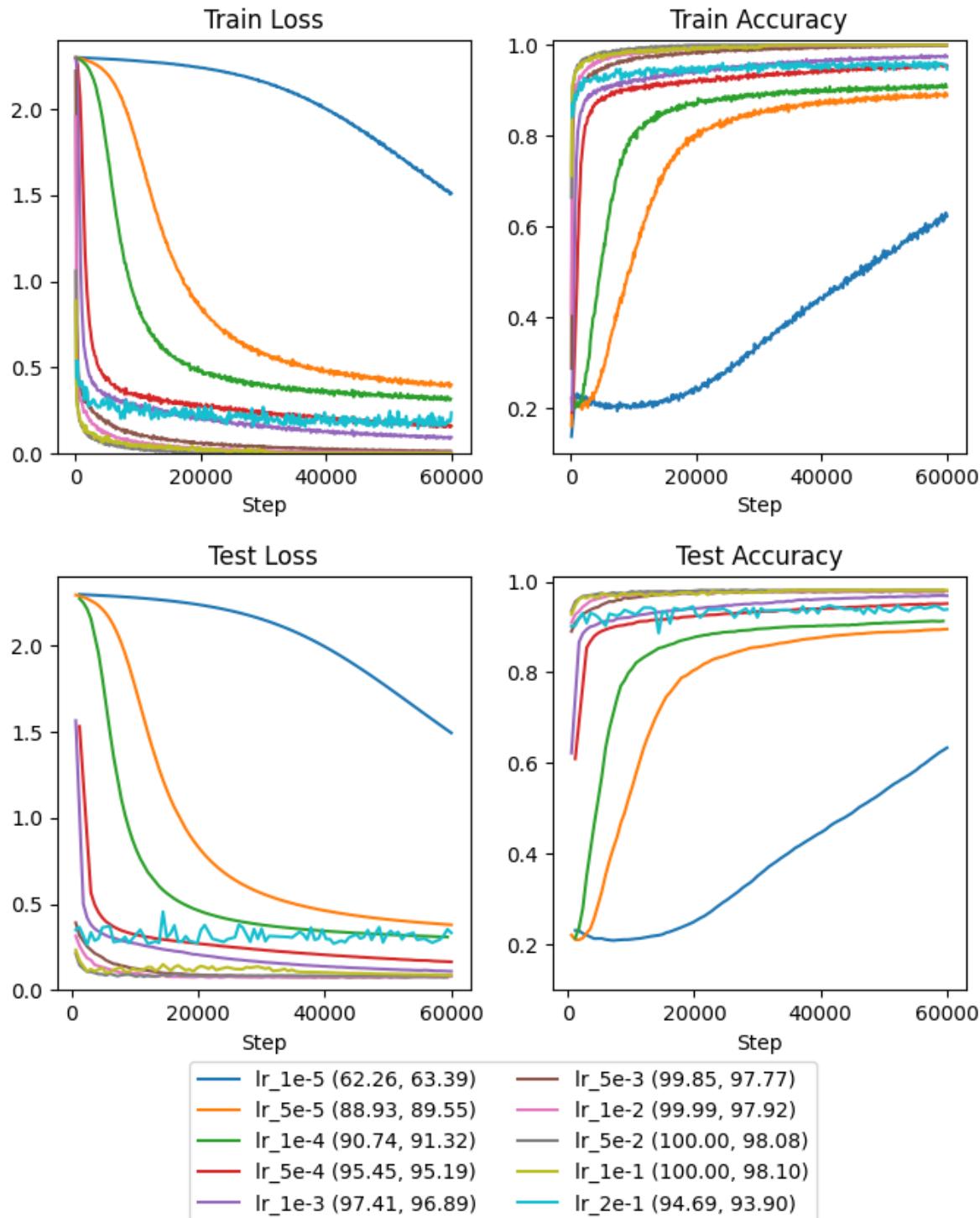


- 相比没有隐藏层来说，使用隐藏层带来了显著的准确率提升。
- 在 [16, 512] 范围内，随着隐藏层参数增加，模型收敛更快，在训练集与测试集上的准确率明显提升，同时在训练集与测试集上准确率差异也增大。这表明在模型架构相同的情况下，更多参数可以

显著增加模型的表现力；虽然模型呈现出逐渐过拟合的倾向，但在使用更多参数拟合测试集的过程中，模型也确实学到了更多特征。

- 在 $[512, 2048]$ 范围内，模型的表现基本类似。这表明此时隐藏层大小已经不是制约模型表现的主要因素。

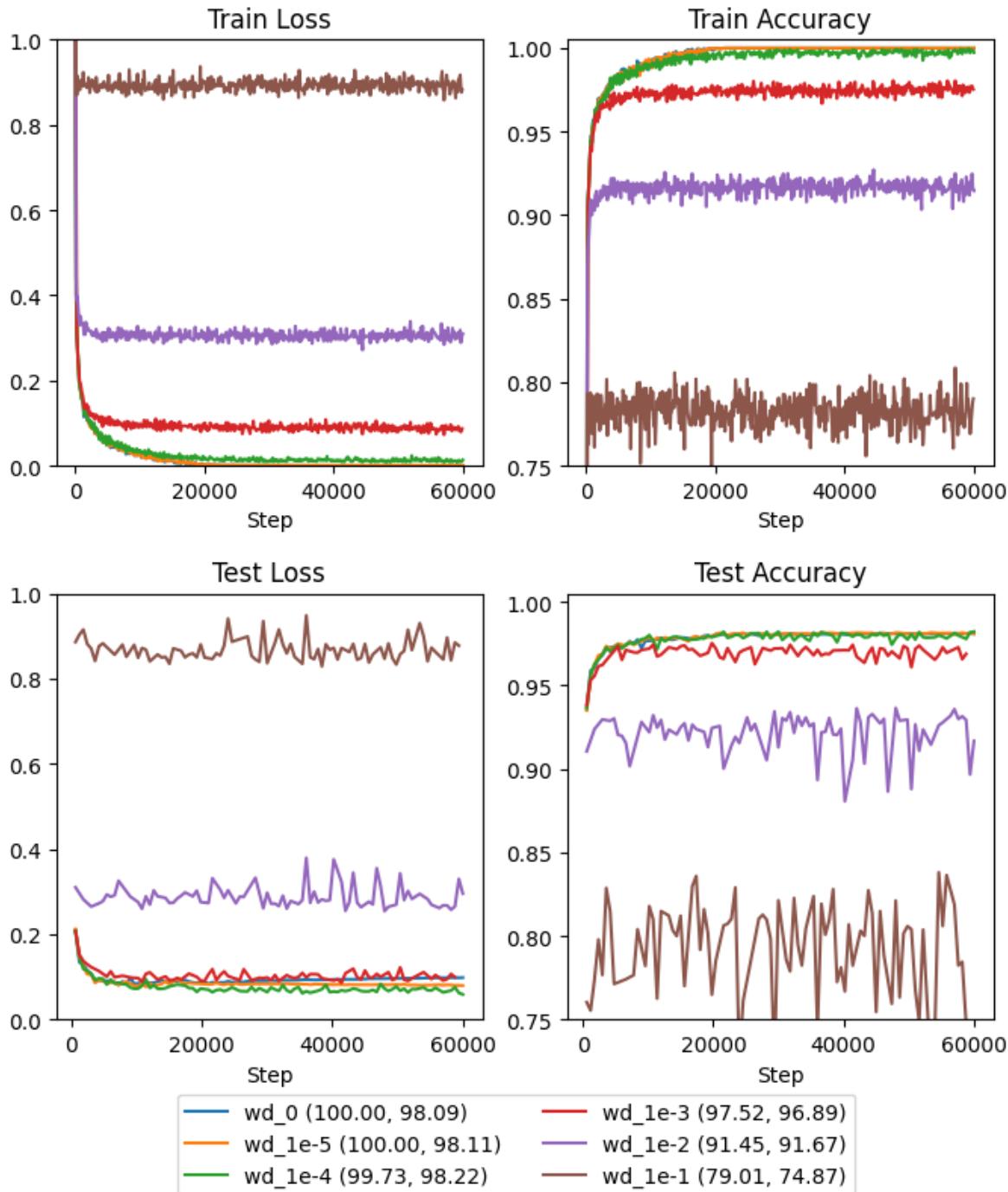
Learning Rate



- 学习率非常小的模型收敛较慢。这主要是因为学习率小的模型梯度更新较慢。同时，学习率小的模型收敛后的准确率较低。这可能是因为学习率过小的模型容易陷入局部极小或者难以离开鞍点。
- 学习率较大的模型收敛较快；在 $[1 \times 10^{-5}, 1 \times 10^{-1}]$ 范围内，模型在训练集及测试集上的准确率随学习率增大而提升。学习率较大时模型在训练过程中的震动也会增加，但这些震动有利于模型脱离局部极小和鞍点，使模型有更好的泛化性能。

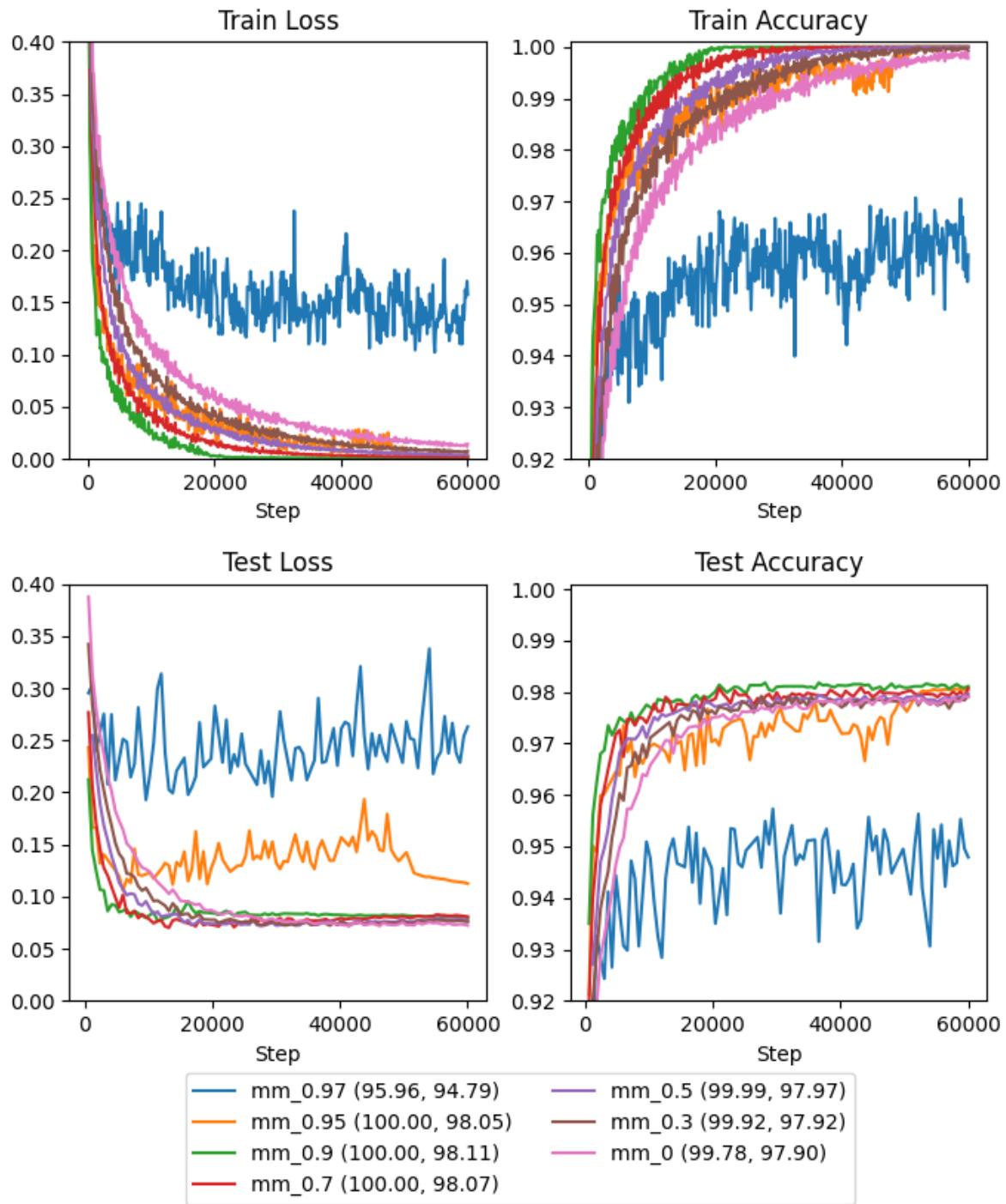
- 当学习率非常大（如 $\eta = 2 \times 10^{-1}$ ）时，由于参数梯度更新幅度过大，模型振荡严重，很难接近全局的极小值，造成准确率下降。

Weight Decay



- Weight decay 较大时，模型的准确率显著降低。对于本实验中这个参数数量较小的网络来说，过低的模型复杂性将限制模型的认知能力。
- Weight decay 较小时，模型的表现普遍较好。本次作业的采用的网络结构较为简单，模型复杂性本身不会很高；而且数据量相比模型参数来说很大，过拟合为问题也不严重。
- 因此，在本次实验中，较小的 Weight decay 也难以发挥防止过拟合的作用；而较大的 weight decay 则会抑制模型性能。Weight decay 并不很适合本次实验的网络架构，而可能更加适合模型参数量较大而数据量相对有限的任务。

Momentum



- 对于动量 $\beta \in [0, 0.9]$ 而言，动量的增大显著提升了收敛速度。动量可以视作对梯度进行了移动加权平均，梯度中有利于收敛的分量将会被积累放大，而不利于收敛的分量则会在平均的过程中被抑制。动量的平均机制使得 SGD 中由于采样而产生的随机噪声被抑制，从而加速了模型的收敛。
- 对于非常接近 1 的动量来说，模型的稳定性和准确率显著下降。类似于 $\sum_{i=0}^{+\infty} \beta^i = \frac{1}{1-\beta}$ ，当 β 非常接近 1 时，动量对梯度的加权平均也会让梯度的积累值变得非常大，造成每次参数更新幅度过大（与过高的学习率类似），模型难以收敛，从而造成模型的剧烈振荡和准确率下降。

数值稳定性

实验过程中观察到了一些溢出到 NaN 或 inf 的现象。他们将会导致模型参数迅速失效。因此在实现上也采取一些提高数值稳定性的措施。

对零取对数

在 CrossEntropy 计算 $\log(h_k)$ 时，若概率 h_k 过小，则会产生 NaN。因此在计算对数概率时，采用 $\log(h_k + \delta)$ 其中 $\delta = 1 \times 10^{-10}$ 。

指数溢出

在计算 Softmax 时，输入过大可能导致指数运算后结果溢出到 inf。注意到

$$h_k = \frac{\exp(x_k)}{\sum_j \exp(x_j)} \quad (a)$$

$$= \frac{\exp(x_k - \max_i x_i)}{\sum_j \exp(x_j - \max_i x_i)} \quad (b)$$

因此，实现时采用 (b) 式计算 Softmax 可以避免此类问题。

在计算 Sigmoid 时，如果输入非常小，也可能出现类似的溢出，此时采用

$$\sigma(x) = \frac{e^x}{e^x + 1} \quad (x < 0)$$

可以防止溢出。但由于在实验过程中没有观测到这类溢出，并没有对这一优化进行实现。

代码修改说明

- 实验中使用 `wandb` 记录和管理训练过程的数据，因此一些文件加入了相应的内容。
- `run_mlp.py`：对该文件进行了大幅度修改，主要是加入了根据命令行参数生成模型、初始化随机种子、初始化 `wandb` 等功能。
- `solve_net.py`：增加了计时功能，增加向 `wandb` 提交 log 的功能，增加对按 Epoch 进行记录的支持。
- `network.py`, `layers.py`: 为 `Layer` 及其派生类实现了 `__str__` 方法，便于打印模型以检查模型结构；为 `Gelu` 和 `Relu` 实现钩子便于记录激活情况；为 `Linear` 实现钩子记录梯度情况。
- `train.sh`: 训练脚本，其中包括训练使用的所有命令及参数。