ANN hw1

默认参数

如果没有特别指出，实验采用的参数为：

参数	Value
Activation Function	Relu
Loss Function	SoftmaxCrossEntropyLoss
learning_rate	0.01
weight_decay	0.0005
momentum	0.9
batch_size	100
Hidden layers	1

如果隐含层为一层，特征维数的变化为$784 \to 128 \to 10$.

如果隐含层为两层，特征维数的变化为$784 \to 256 \to 64 \to 10$.

实验环境

```
Python==3.9.16
Numpy==1.23.5
GPU: Apple M2
```

不同激活函数与损失函数

这次实验中，一共采用了`Selu, Swish, Relu, Gelu`四种激活函数，`MSELoss, SoftmaxCrossEntropyLoss, HingeLoss, FocalLoss`四种损失函数。故激活函数与损失函数两两搭配共有$4 \times 4 = 16$个实验结果，以下将其列出。

Single Hidden Layer

- Acc

	Train				**Test**			
	Selu	Swish	Relu	Gelu	Selu	Swish	Relu	Gelu
MSE	0.9554	0.9844	0.9828	0.9802	0.952	0.9787	0.9735	0.9736
SoftmaxCE	0.9868	0.993	0.9908	0.9876	0.9785	0.9767	0.9788	0.9795
Hinge	0.9938	0.9966	0.9962	0.997	0.9815	0.9808	0.9797	0.9824
Focal	0.9346	0.927	0.9476	0.9426	0.9372	0.919	0.9449	0.9414

Two Hidden Layers

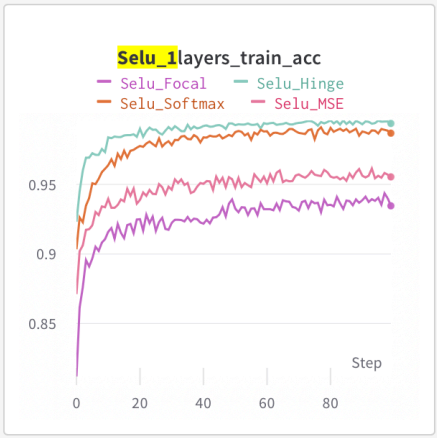
- Acc

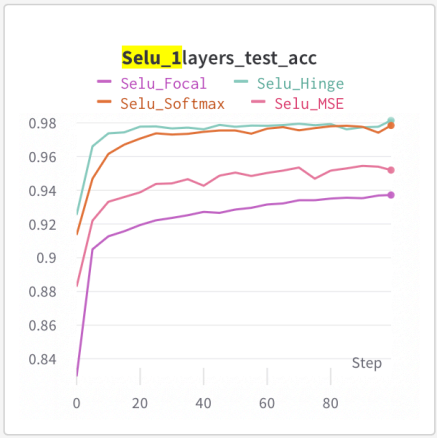
	Train				**Test**			
	Selu	Swish	Relu	Gelu	Selu	Swish	Relu	Gelu
MSE	0.987	0.9844	0.9974	0.989	0.9716	0.9787	0.9861	0.98
SoftmaxCE	0.9762	0.993	0.9962	0.9944	0.9773	0.9767	0.9814	0.9773
Hinge	0.9954	0.9966	0.9944	0.9956	0.9817	0.9808	0.9823	0.9817
Focal	0.9428	0.927	0.9502	0.938	0.9428	0.919	0.9516	0.9368

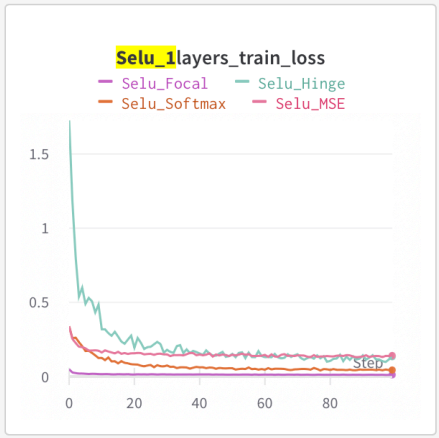
以图表的形式展示不同模型的收敛速度。在本次实验中，以激活函数作为分类依据。

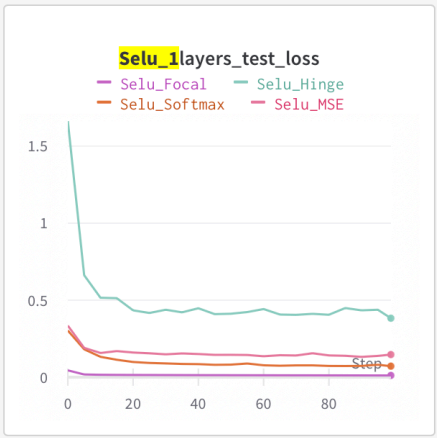
Single Hidden Layer

- Selu

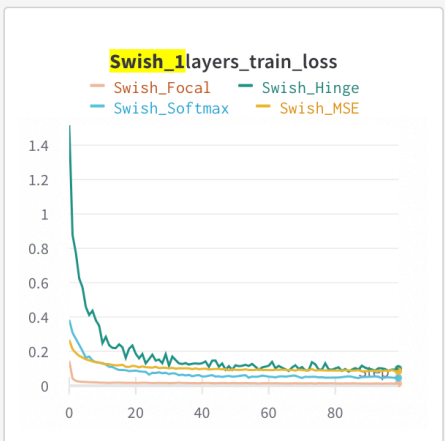

Selu_1layers_train_acc
Selu_Focal
Selu_Hinge
Selu_Softmax
Selu_MSE
0.95
0.9
0.85
Step
0
20
40
60
80


Selu_1layers_test_acc
Selu_Focal
Selu_Hinge
Selu_Softmax
Selu_MSE
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
Step
0
20
40
60
80


Selu_1layers_train_loss
Selu_Focal
Selu_Hinge
Selu_Softmax
Selu_MSE
1.5
1
0.5
0
Step
0
20
40
60
80

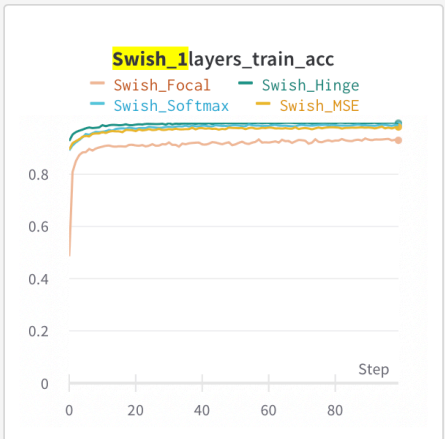

Selu_1layers_test_loss
Selu_Focal
Selu_Hinge
Selu_Softmax
Selu_MSE
1.5
1
0.5
0
Step
0
20
40
60
80

- Swish

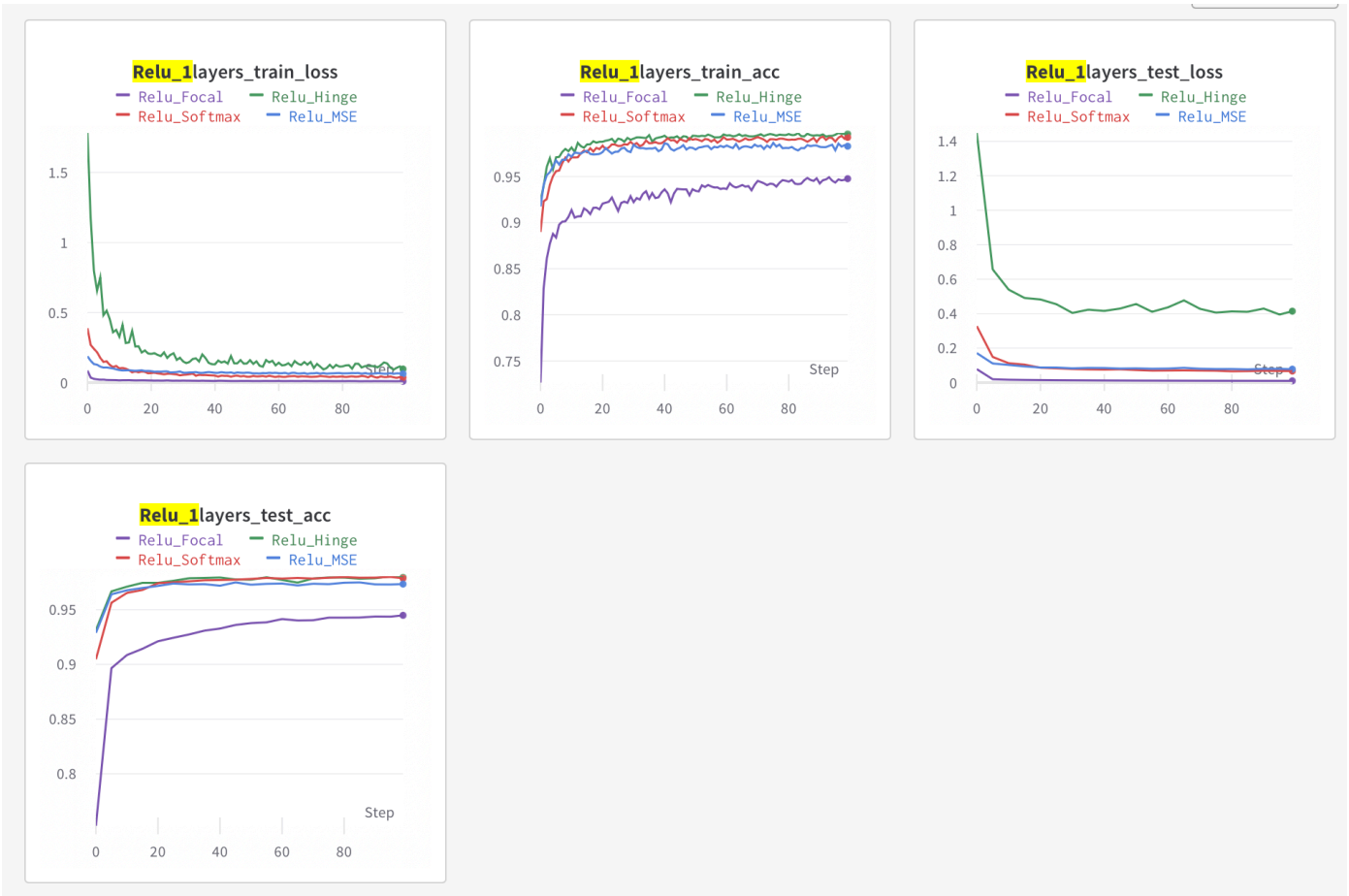

Swish_1layers_train_loss
Swish_Focal
Swish_Hinge
Swish_Softmax
Swish_MSE
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Step
0
20
40
60
80


Swish_1layers_test_loss
Swish_Focal
Swish_Hinge
Swish_Softmax
Swish_MSE
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Step
0
20
40
60
80


Swish_1layers_test_acc
Swish_Focal
Swish_Hinge
Swish_Softmax
Swish_MSE
0.8
0.6
0.4
0.2
0
Step
0
20
40
60
80


Swish_1layers_train_acc
Swish_Focal
Swish_Hinge
Swish_Softmax
Swish_MSE
0.8
0.6
0.4
0.2
0
Step
0
20
40
60
80

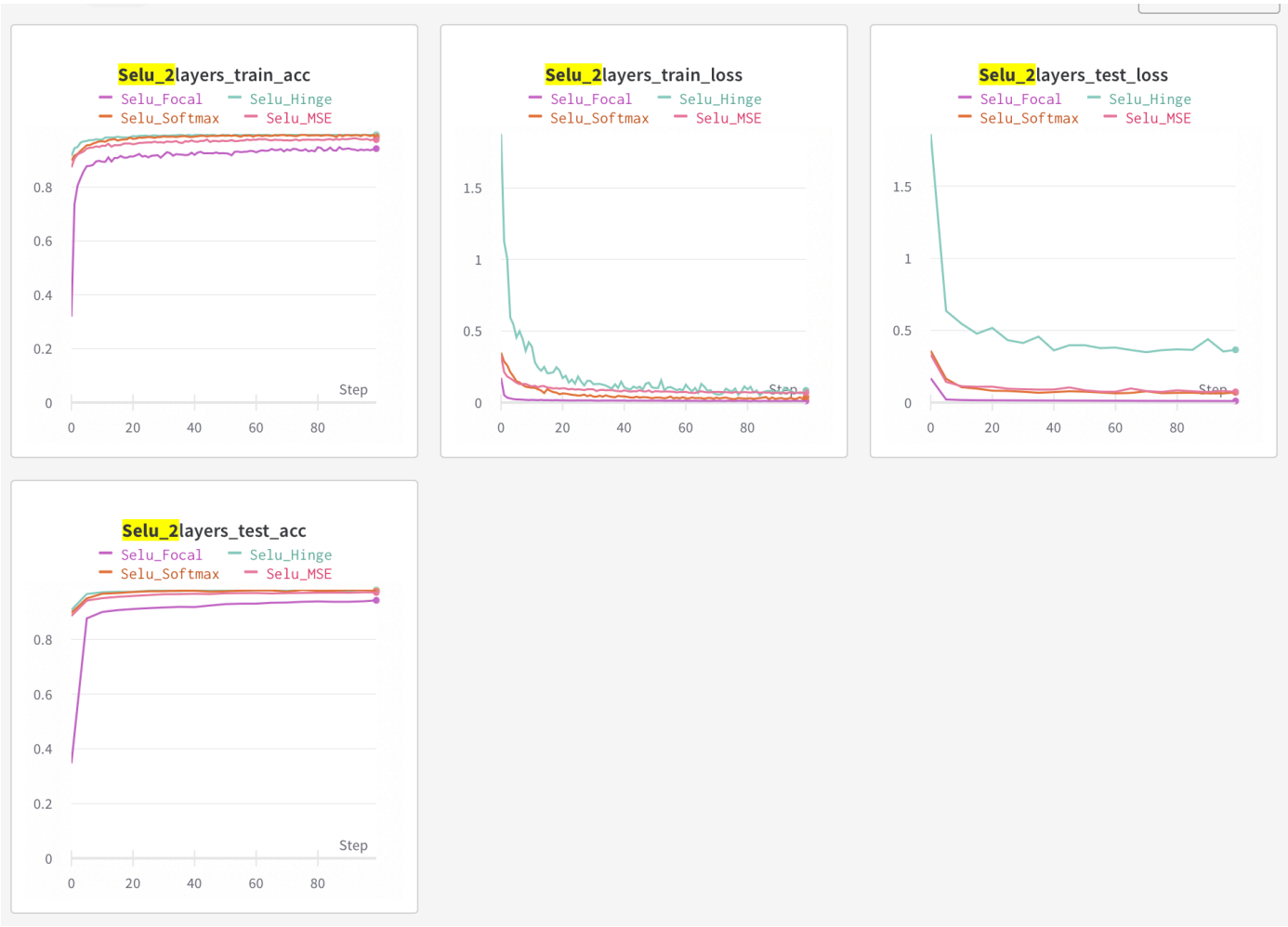
- Relu


Relu_1layers_train_loss
Relu_Focal
Relu_Hinge
Relu_Softmax
Relu_MSE
1.5
1
0.5
0
0
20
40
60
80
Step
Relu_1layers_train_acc
Relu_Focal
Relu_Hinge
Relu_Softmax
Relu_MSE
0.95
0.9
0.85
0.8
0.75
0
20
40
60
80
Step
Relu_1layers_test_loss
Relu_Focal
Relu_Hinge
Relu_Softmax
Relu_MSE
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
Step
Relu_1layers_test_acc
Relu_Focal
Relu_Hinge
Relu_Softmax
Relu_MSE
0.95
0.9
0.85
0.8
0
20
40
60
80
Step

- Gelu

Two Hidden Layers

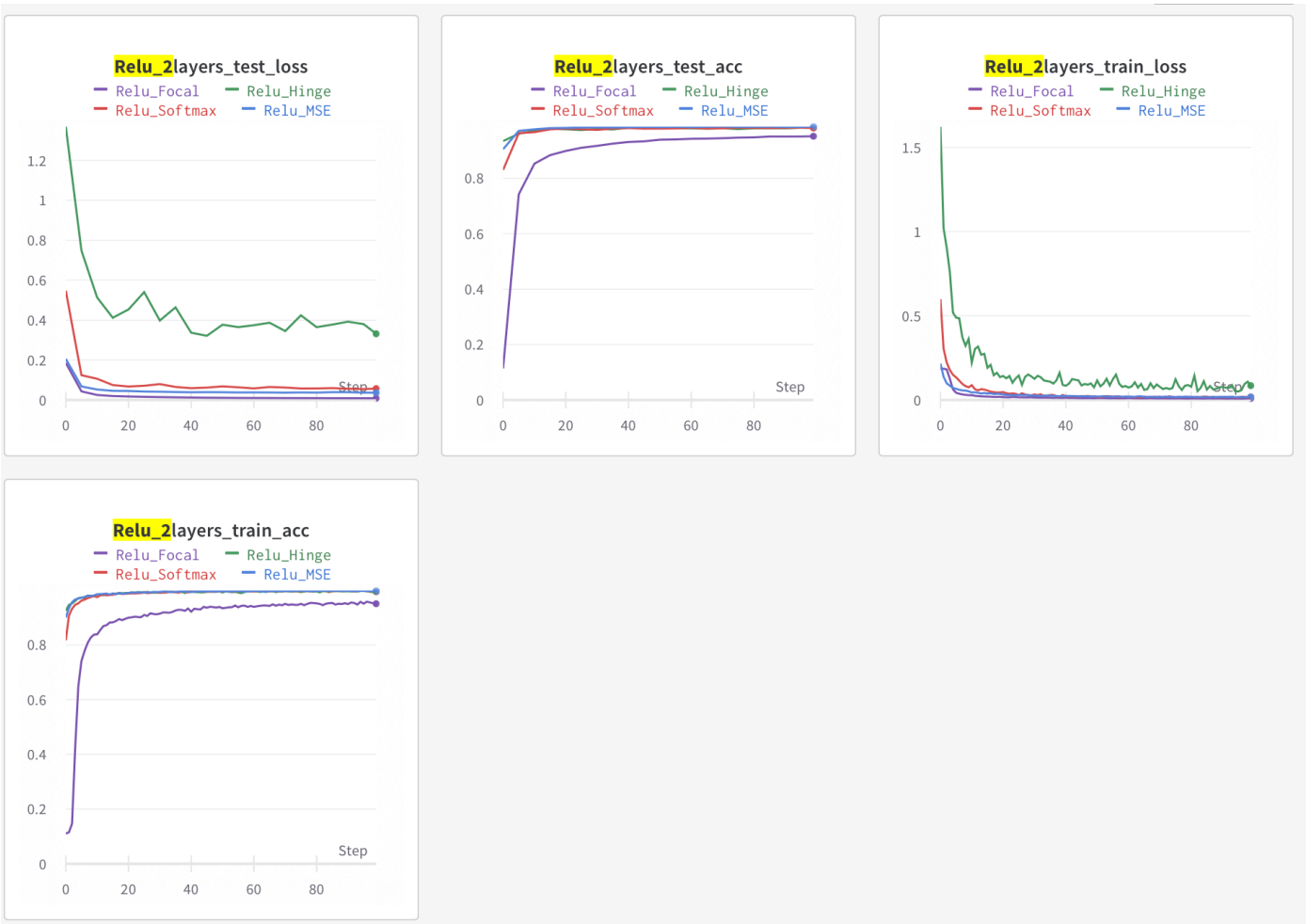
- Selu


Selu_2layers_train_acc
Selu_Focal
Selu_Hinge
Selu_Softmax
Selu_MSE
Step
Selu_2layers_train_loss
Selu_Focal
Selu_Hinge
Selu_Softmax
Selu_MSE
Step
Selu_2layers_test_loss
Selu_Focal
Selu_Hinge
Selu_Softmax
Selu_MSE
Step
Selu_2layers_test_acc
Selu_Focal
Selu_Hinge
Selu_Softmax
Selu_MSE
Step

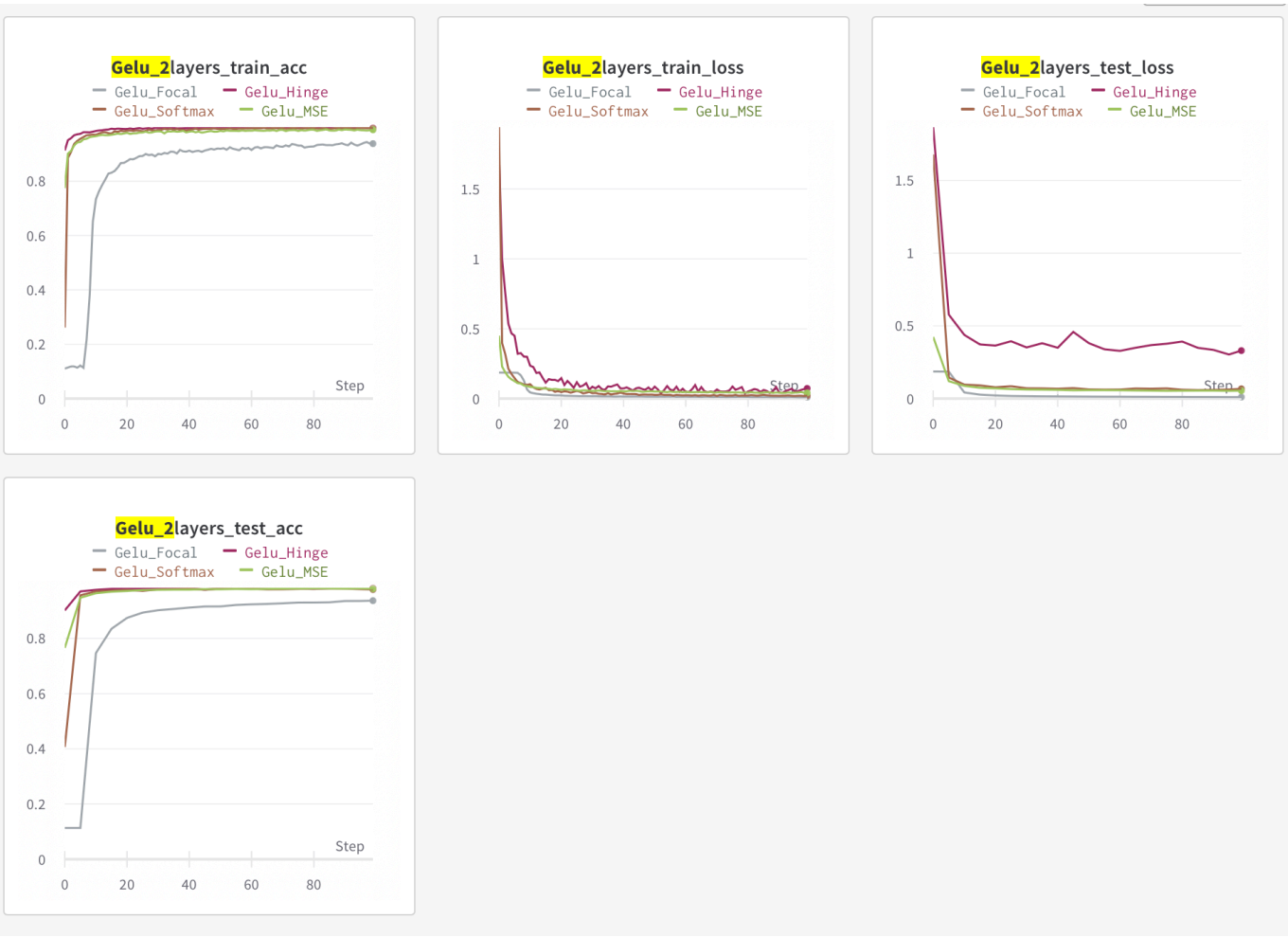
- Swish


Swish_2layers_test_loss
Swish_Focal
Swish_Hinge
Swish_Softmax
Swish_MSE
Step
Swish_2layers_test_acc
Swish_Focal
Swish_Hinge
Swish_Softmax
Swish_MSE
Step
Swish_2layers_train_acc
Swish_Focal
Swish_Hinge
Swish_Softmax
Swish_MSE
Step
Swish_2layers_train_loss
Swish_Focal
Swish_Hinge
Swish_Softmax
Swish_MSE
Step

- Relu


Relu_2layers_test_loss
Relu_Focal
Relu_Hinge
Relu_Softmax
Relu_MSE
Step
Relu_2layers_test_acc
Relu_Focal
Relu_Hinge
Relu_Softmax
Relu_MSE
Step
Relu_2layers_train_loss
Relu_Focal
Relu_Hinge
Relu_Softmax
Relu_MSE
Step
Relu_2layers_train_acc
Relu_Focal
Relu_Hinge
Relu_Softmax
Relu_MSE
Step

- Gelu



结果分析

收敛速度

- 损失函数
  - 从以上的图表中可以看出，不论激活函数选择如何，`FocalLoss`总是收敛得最快（而且损失函数值最小），随后是`MSELoss`与`SoftmaxCELoss`。`HingeLoss`总是收敛得最慢（需要最多`epoch`）。
  - 即收敛速度大致为：

$$FocalLoss > MSELoss > SoftmaxCELoss > HingeLoss \tag{1}$$

- 激活函数
  - 观察可知，`Swish`的导数值较小，所以在梯度下降中的收敛速度不如其他三者。
  - 即收敛速度大致为：

$$Gelu \approx Selu \approx Relu > Swish \tag{2}$$

Test上的准确率

- 损失函数
  - 总的来说：效果为

$$Hinge > MSE \approx SoftmaxCE > Focal \tag{3}$$

  - 有意思的是，`HingeLoss`的性能比剩下三者好的情况在Train上就已经体现出来了。这说明`HingeLoss`的优势不是在于泛化能力上，而（可能）在于其能更好地刻画分类问题的损失（MSE, CE实际上都是用Bayes解释）。
- 激活函数
  - 在用`MSELoss`作为损失时，`Selu`的效果明显劣于剩下三者，而在其他损失函数的情况下这四个激活函数表现相当。
- 总体而言，单隐藏层表现最好的组合为`Hinge + Gelu`，而且这个优势在Train上没有体现出来。故可能是`Gelu`的泛化能力更强（事实上可能确实如此，`Gelu`类似Dropout的行为会减少模型复杂度，提升泛化能力）导致的结果。
- 提升隐含层的个数并没有使最大准确率上升，但是使不同损失函数、激活函数的效果整体提升了一些。然而这导致了几乎双倍的计算成本，故是得不偿失的。

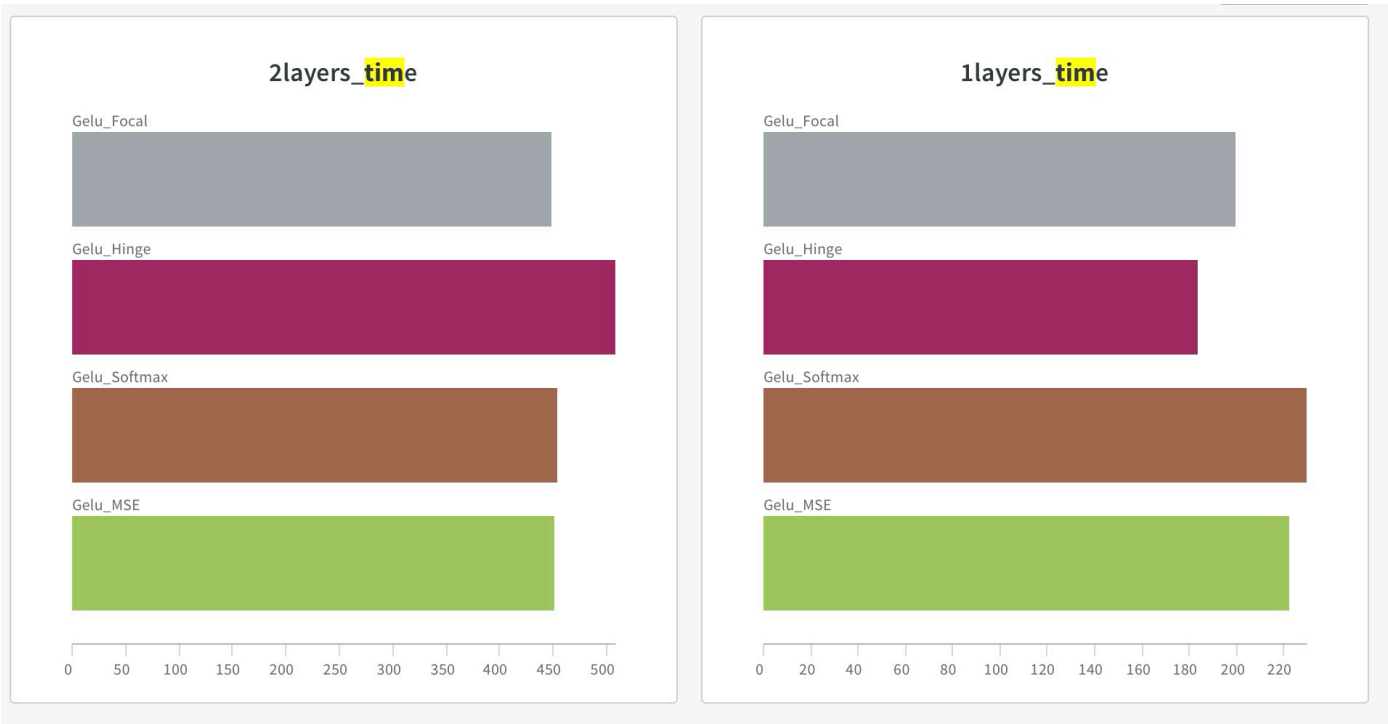
计算速度

- Selu

- Swish

- Relu

- Gelu



可以看出，不同激活函数之间的耗费时间排序为：

$$Swish > Gelu \approx Relu > Selu \tag{4}$$

这可能是因为`Swish`的导数计算相对复杂。

而不同损失函数之间的耗费时间排序为：

$$SoftmaxCE \approx Focal > MSE > Hinge \tag{5}$$

`SoftmaxCE`与`Focal`的导数计算是类似的，有大量的指数运算，故耗时较久。

另外，随着层数增加一层，训练时间基本上增加了一倍，可以看出模型参数量的增加。

计算稳定性

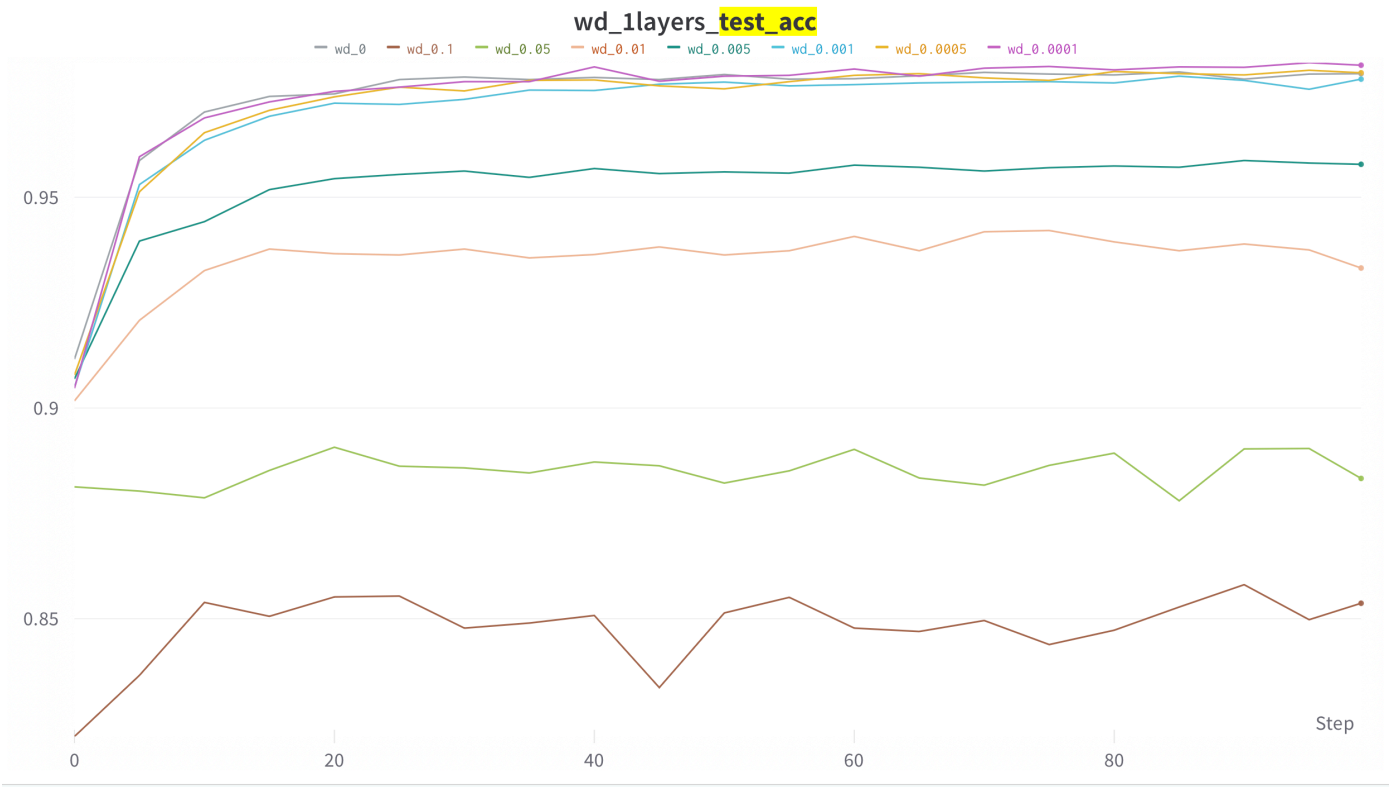
在`SoftmaxCE`和`Focal`中，都使用了对数计算。为了防止出现数值错误，一个可行的解决方法是在每个用到`log`的地方都加上一个很小的偏置$\epsilon \approx 10^{-8}$。

另外，使用`Softmax`，`tanh`作为激活函数可能会出现梯度消失或梯度爆炸的问题。可以改用`Selu`作为激活函数，或者是采用ResNet的结构解决梯度不在阈值范围内的问题。

解决过拟合的方法

- 简化模型：L1/L2正则化，在这次实验中使用了L2正则化，体现为`weight_decay`。L1正则化相比L2正则化更可能输出稀疏解。
- Dropout：训练时，随机忽略几个神经元的输入。
- 增加数据量：增加Training Size，使得模型拥有更好的泛化能力。
- 适时停止训练：通过观察模型在验证集上的表现，决定何时终止。

本次实验对`weight_decay`进行研究，得出图表如下：



可以看出：

- 过大的`weight_decay`会导致模型过于简单，失去拟合能力。
- 合适的`weight_decay`可以使模型有效的避免过拟合，在测试集上获得更高的准确率。
- 本实验所涉及的模型本身就足够简单，故正则化不能很好地体现效果。当遇到更大的模型时，正则化的效果将会更明显。