Source Code:

Colab: https://colab.research.google.com/drive/1kfgURo8tWuPyDrPZpg5nOa5CHFPIjVnC
GitHub: https://github.com/steve3p0/cs510dl/tree/master/hw2

1. Construct a simple LeNet CNN

Construct a simple CNN that follows the architecture of LeNet. However, the images have 3 channels. The first kernel will be 6 5x5x3 kernels that takes us to 28x28x6. The remainder of the network is the same. Stride is 1 for convolution and 2 for pooling layers. You need to train the network for different choices of parameters: learning rate 0.1, 0.01, 0.001, activation function = sigmoid, Tanh, loss function = MSE, cross-entropy.

a. Submit plots of training error vs epoch and test error vs epoch for each combination of parameters.
NOTE: For all plots of all 12 cases please see Appendix A.
Keep in mind that the results here may slightly vary from the results on the Jupyter Colab Notebook as those results were run at a different time.

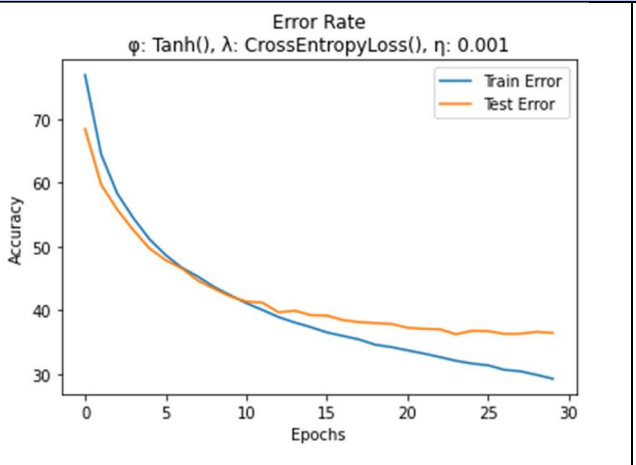
Best Performing Model

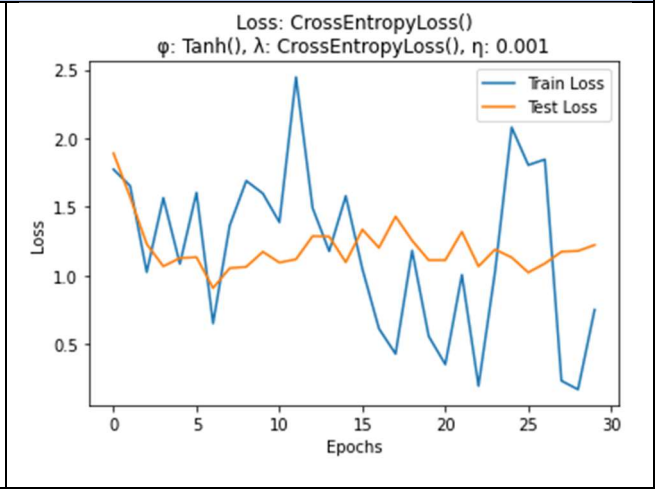
	Accuracy	Error	Loss
Train	70.69%	29.31%	0.75
Test	63.51%	36.49%	1.22

Total Training Time: 3:33:10.605737 (12 iterations, 30 epochs each)

plane	car	bird	cat	deer	dog	Frog	horse	ship	truck
71.00%	72.00%	49.40%	43.30%	60.70%	50.40%	83.40%	63.90%	76.70%	64.30%

ϕ: `Tanh()`, λ: `MSELoss()`, η: `0.001`





Describe the reasons for what you observe.

The best performing model used the Tanh Activation function, with Cross Entropy Loss and a learning rate of 0.001. This demonstrates that Tanh, with its upper and lower bounds of 1 and -1, respectively, was able to outperform Sigmoid by minimizing loss with a slower learning rate of 0.001. Sigmoid seemed to suffer across the board. With Cross Entropy loss, and a learning rate of 0.001, Sigmoid suffered from unstable gradients. Even with more stable gradients with using a learning rate of 0.01, its test error rate never dipped below 40%. And when it did hit 40%, the graph is obvious that the model does not generalize well, as the error rate plunged for training, but not for testing data. Several of the Sigmoid models, especially the ones with lower learning rates appear to suffer from vanishing gradients.

b. For one trained network and parameter choice, display feature maps of 10 images at the last convolution layer. What has your network learned?

LeNet-5 was originally hand-designed in 1995 by Yann LeCun using local connections to restrict the computation and memory complexity and to break symmetry.

When the LeNet-5 architecture was published by LeCun in 1995, he was restricted by the available hardware at time. He hand-designed LeNet-5 by using local connections to restrict the computation and memory complexity and to break symmetry. In 2021 we have access to GPUs and PyTorch's unbounded Conv2d object that ***maps all input features to all output features****. Furthermore, new methods in breaking symmetry include random weight initialization, dropouts, and others.*

2. ReLU, Learning Rate .001, Cross Entropy Loss, and 3x3 Kernels

Select ReLU, learning rate of 0.001, cross-entropy loss. Change the network to use 3x3 kernels for both convolutions rather than 5x5. Same pooling.

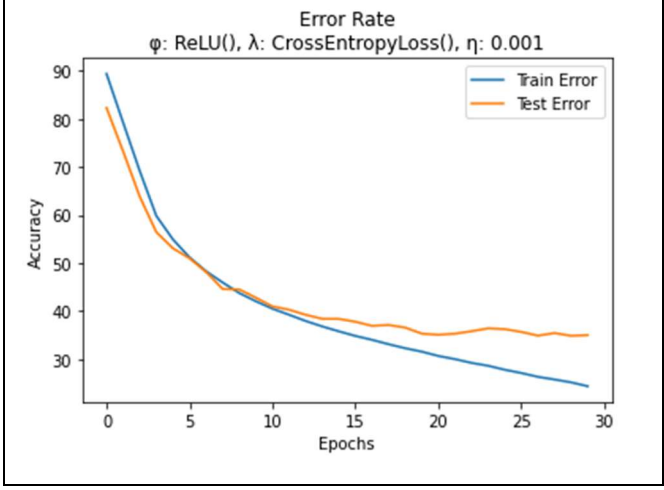
a. Plot the error (test, train) vs epoch and discuss how your results differ from the previous network.

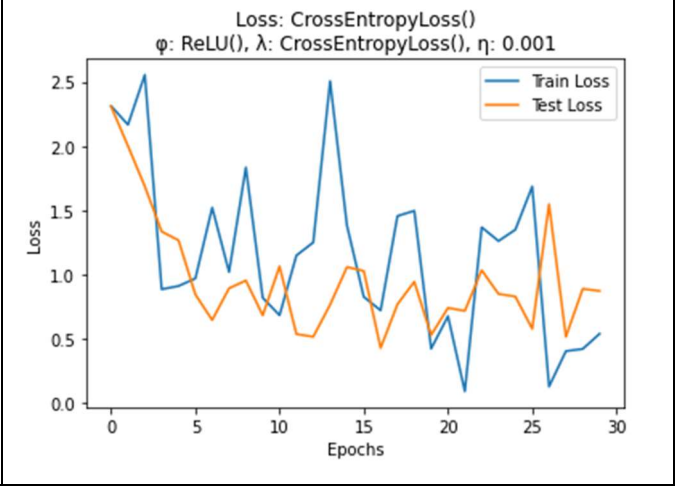
	Accuracy	Error	Loss
Train	75.65%	24.35%	0.54
Test	65.02%	34.98%	0.87

Training Time: 0:17:49.092880

plane	car	bird	cat	deer	dog	frog	horse	ship	truck
63.50%	76.20%	56.10%	52.60%	52.10%	49.20%	72.00%	75.50%	84.30%	70.70%

ϕ: `ReLU()`, λ: `CrossEntropyLoss()`, η: `0.001`, K: `3x3`





Actually, we get very similar results with part 1's best model using Tanh, but with the same loss function and learning rate as this one using ReLU. We see only marginal improvement with a 2.5% reduction in the error rate. One noticeable difference is that there is a gap in the error rate between the test and train, suggesting that this ReLU model does not generalize as well as the Tanh one.

I did see slightly less erratic test loss reported for Tanh than with ReLu. However, the testing loss reported by this model seemed to mimic the behavior of its training loss more than the previous model did.

3. CNN with 5 Layers

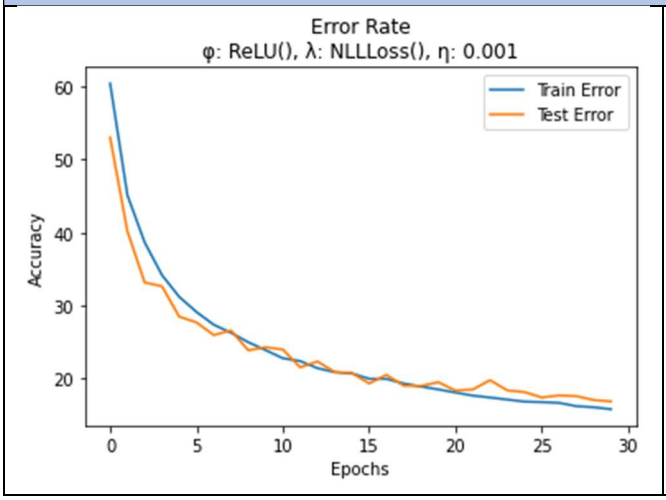
Build a CNN with 5 convolution layers with 3x3 kernels and corresponding 2x2 average pooling with stride 1. Use padding at each convolution layer so that the size of the output matches the input after each convolution layer. Train the network for any choice of model parameters.

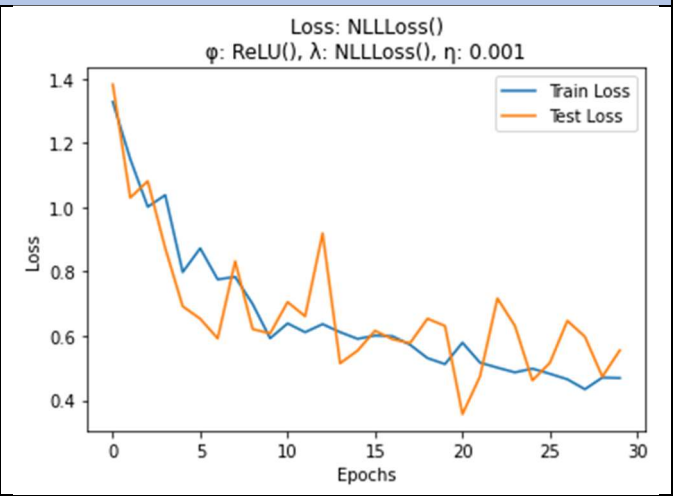
	Accuracy	Error	Loss
Train	84.16%	15.84%	0.47
Test	83.10%	16.90%	0.56

Training Time: 0:08:04.733747

plane	car	bird	cat	deer	dog	frog	horse	ship	truck
83.70%	92.50%	76.40%	66.50%	80.10%	74.70%	92.60%	84.70%	87.70%	89.30%

φ: ReLU(), λ: NLLLoss(), η: 0.001, K:3x3





a. How long did it take to train? Why?

It took approximately 8 minutes to train with 30 epochs. This was less than half of what all the other models took to train (17 minutes average). This was due to using batched normalization.

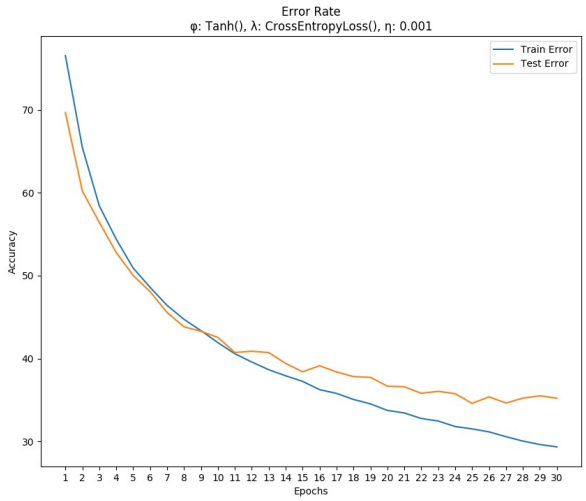
b. How does the accuracy of this network compare against the previous two? Explain.

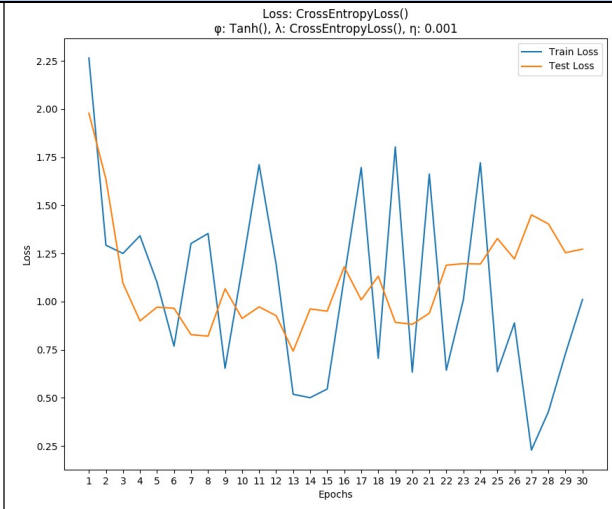
We saw significant improvement with this model using batched normalization, ReLu, minimizing log likelihood loss with a learning rate of 0.001, and the use of dropouts.

APPENDIX A: Plots of LeNet-5 CIFAR-10 Models

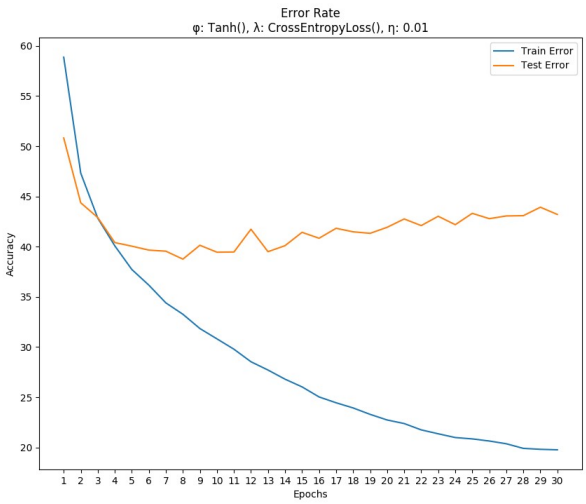
NOTE: For the most updated results see the Colab Jupyter Notebook:

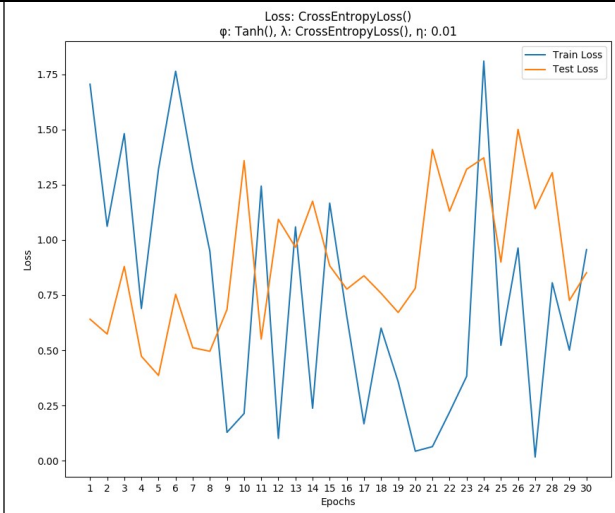
ϕ: Tanh(), λ: CrossEntropyLoss(), η: 0.001



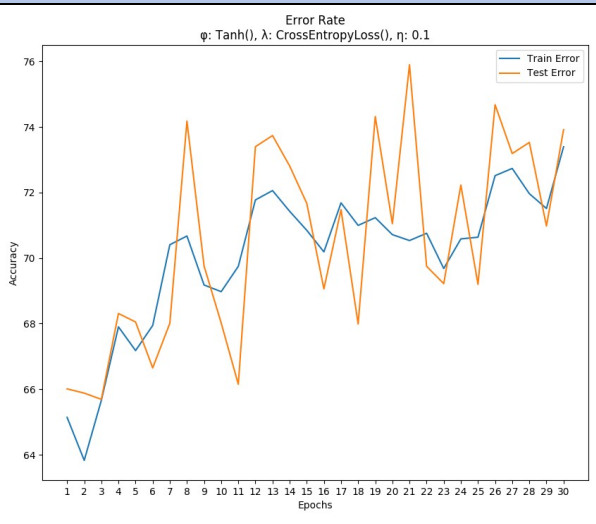


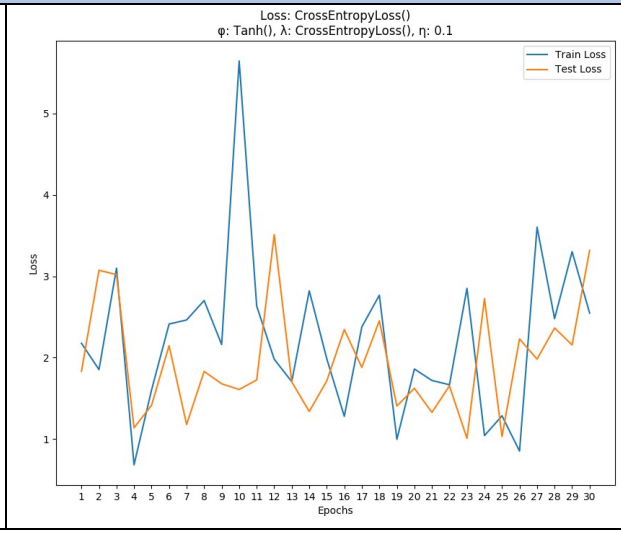
ϕ: Tanh(), λ: CrossEntropyLoss(), η: 0.01



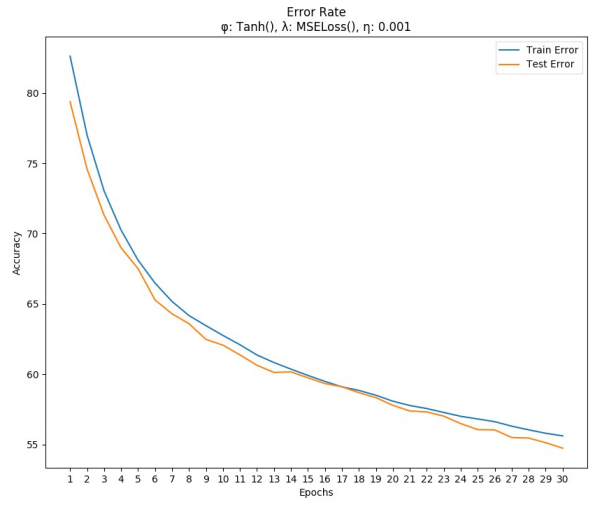


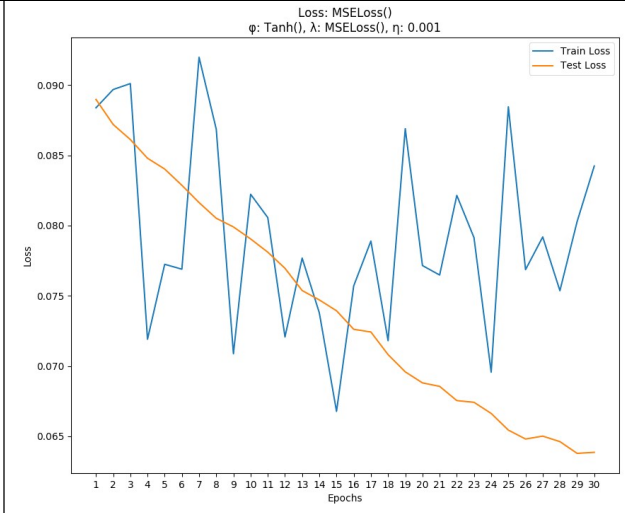
ϕ: Tanh(), λ: CrossEntropyLoss(), η: 0.1



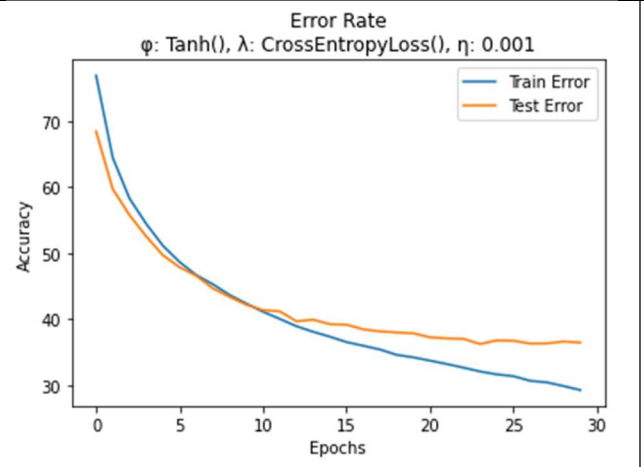


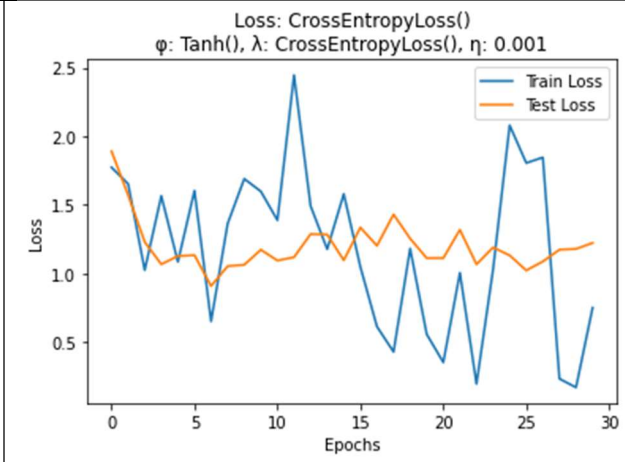
φ: Tanh(), λ: MSELoss(), η: 0.001



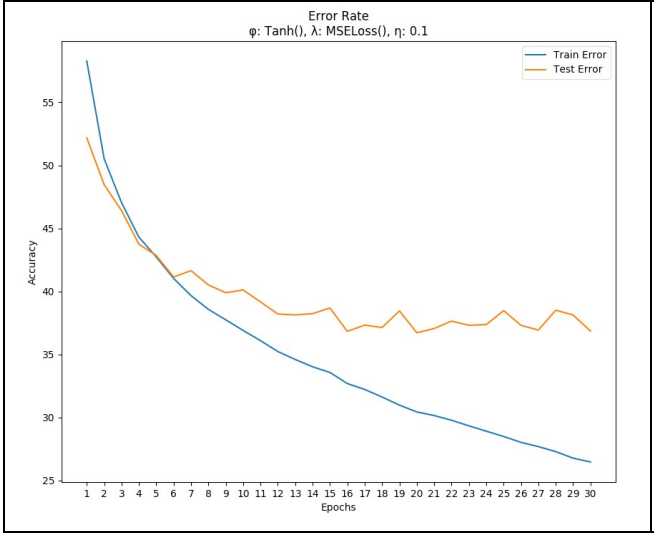


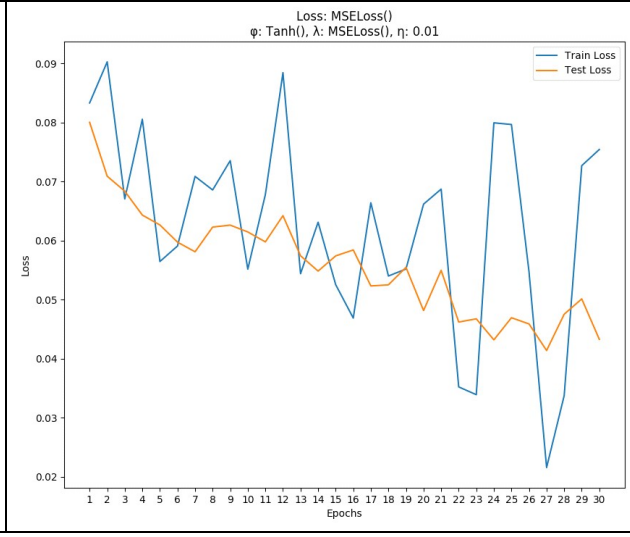
φ: Tanh(), λ: MSELoss(), η: 0.01



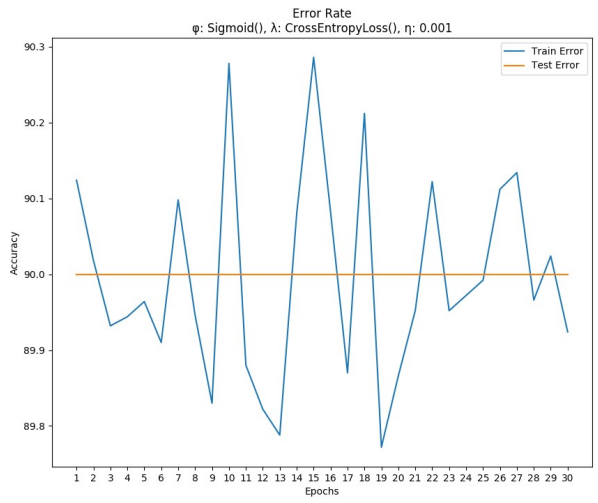


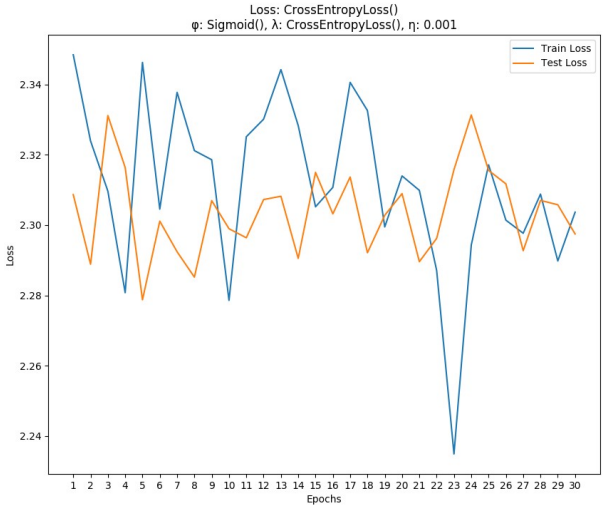
φ: Tanh(), λ: MSELoss(), η: 0.1



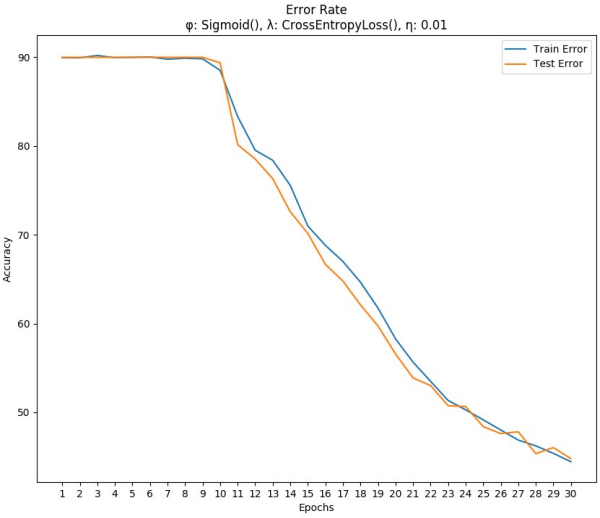


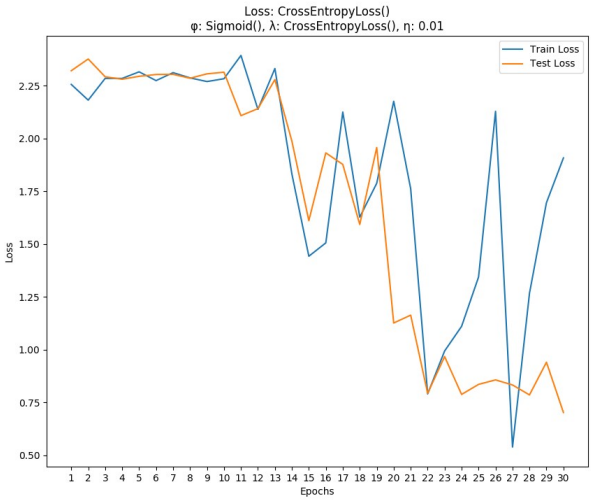
ϕ: Sigmoid(), λ: CrossEntropyLoss(), η: 0.001



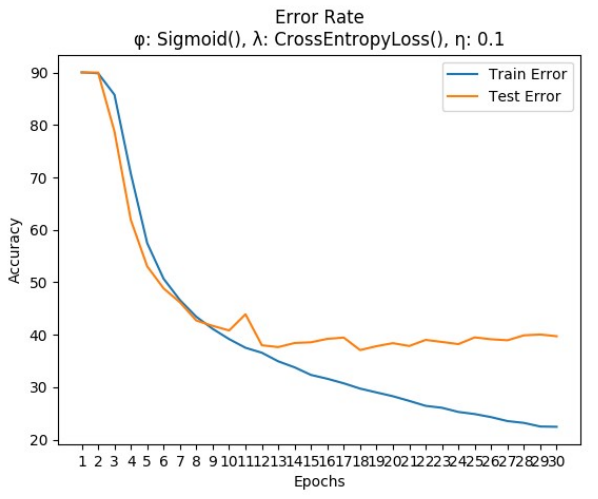


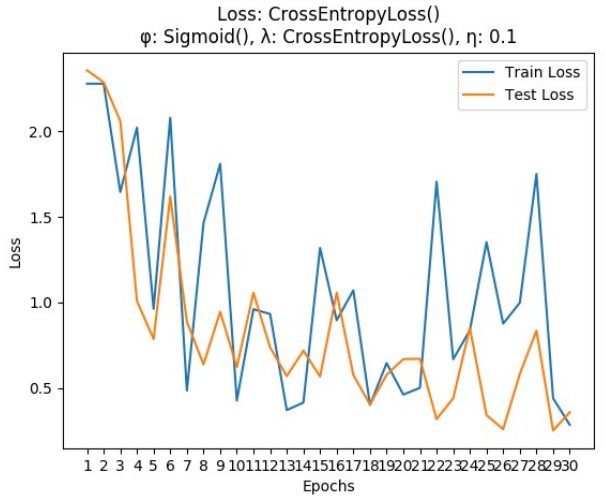
ϕ: Sigmoid (), λ: CrossEntropyLoss(), η: 0.01



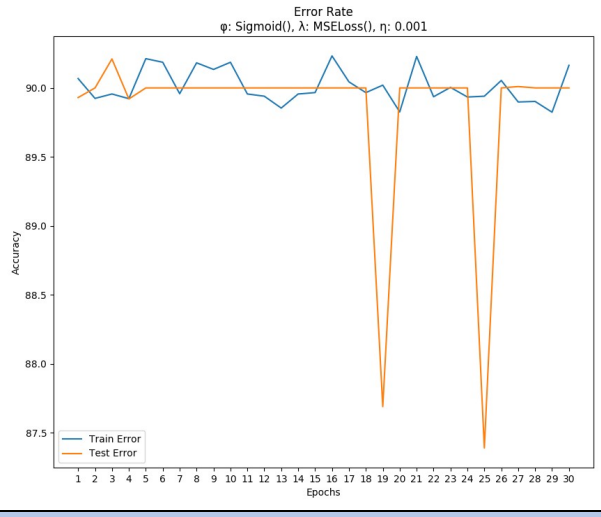


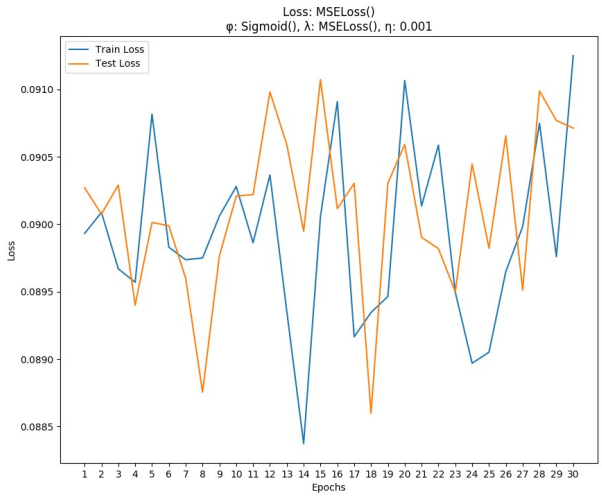
ϕ: Sigmoid (), λ: CrossEntropyLoss(), η: 0.1



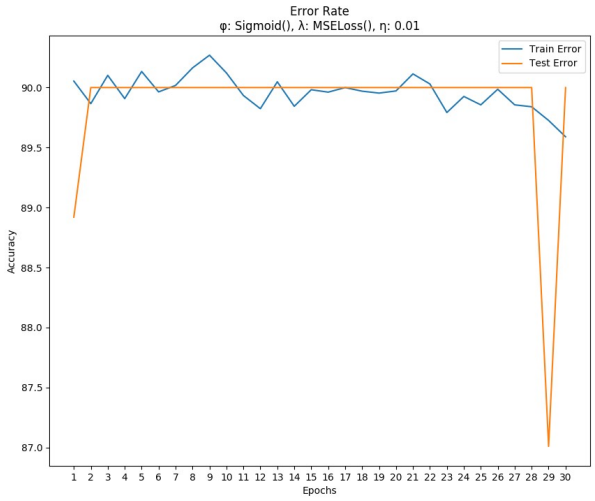


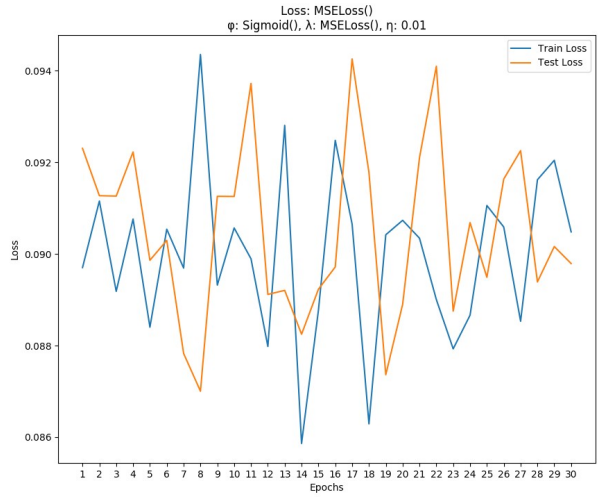
ϕ: Sigmoid(), λ: MSELoss(), η: 0.001





ϕ: Sigmoid (), λ: MSELoss(), η: 0.01





ϕ: Sigmoid (), λ: MSELoss(), η: 0.1