Analysis of Short-Run Performance of Regularization Methods when Combined with Different Activation Functions

Miguel Garcia
University of California, San Diego
mig053@ucsd.edu

Abstract

In this paper, I use Convolutional Neural Networks to envision which method to prevent overfitting works best with different activation functions in the short run. Though there has been many articles and reports about methods to prevent overfitting, I thought that they would vary in performance depending on which activation functions they are used with. The dataset is the MNIST handwritten digit database which contains 60,000 training samples and 10,000 testing samples. I train the models with 12 epochs each as more epochs are not necessary since the model converges to a reasonably high accuracy with just a few epochs and change the hyperparameters for every analysis I run. Finally, I graph the training and testing loss and accuracy curves for a 10% validation split from the training samples.

1 Introduction

It is important to prevent overfitting in any deep neural network model; making sure it performs well on data that isn't included as part of the training is essentially the goal; it makes the model useful and applicable when trying to use it for different tasks. In the recent years, many researchers have started to come up with interesting methods that help with preventing overfitting. One thing I realized from all these overfitting techniques is that there really is not a way to say that one of these is better than the others; in fact, all of them perform differently when we vary the parameters in the neural network. My motivation for this came from one of my visits to the recently renovated mall called UTC in La Jolla. When you are about to leave with your car, the license plate gets recognized and you do not need to insert the ticket in

the machine which makes the flow of traffic in/outside of the mall faster. This method of license plate would work best if we assume they used training samples from American license plates; this would work perfectly with any license plate that looks similar to the ones used in the train data. What happens when we start bringing in variety is the question that needs to get answered. If we bring license plates from our neighboring country (Mexico) and for neighboring states which have different ways of writing the font or have different types of characters, we start having a problem. This is what can be denoted as overfitting; when the classifier network only has good performance on the training data samples rather than the testing ones. From the many possible ways to prevent overfitting, it is necessary to make sure the right one is used depending on how the network is structured and what activation function it is currently using. What I want to do in this paper is analyze which methods to prevent overfitting are the most suitable for each activation function that can help achieve convergence of test and train error in the short run using a few epochs and a 10% validation split and hyperparameter tuning. I also attached my code in the submission.

2 Dataset

2.1 Data Sets Used

In this section, you can see the summary of the datasets used.

The data set used for this paper is the MNIST [1]

The MNIST dataset is a database of handwritten digits containing 60,000 examples with 10,000 test examples. It can also be normalized and centered based on fixed-size preference. Dimensions are 28x28. I normalized the dataset before training.

2.2 Data Examples

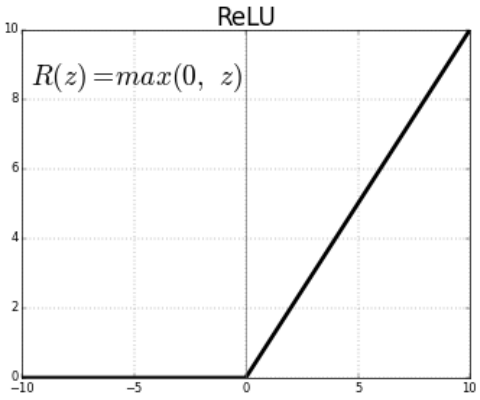
Examples from the dataset are shown below.

3 Methods

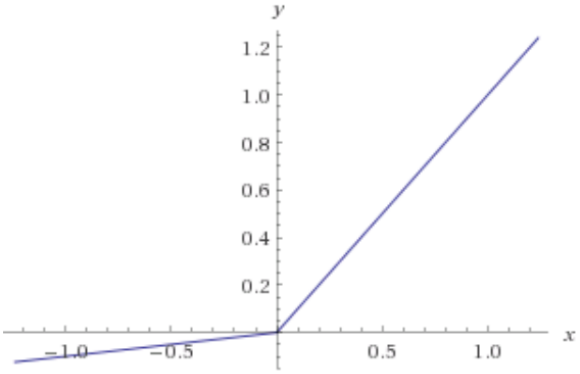
In this section, you can see the summary of the activation functions used, methods to prevent overfitting, hyperparameters, and details on the network used.

3.1 Activation Functions

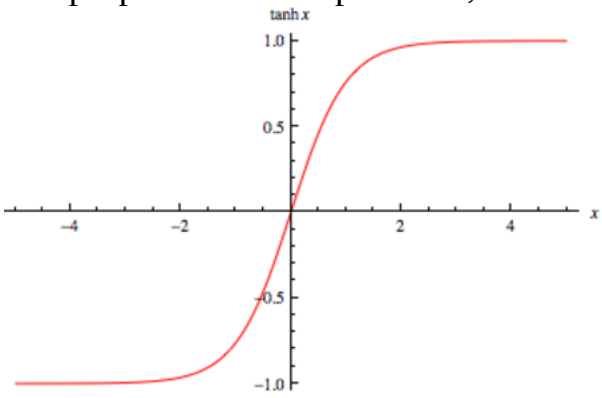
First, I use ReLU activation function, a linear function that converts all negative outputs to zero. It is one of the most common functions used.



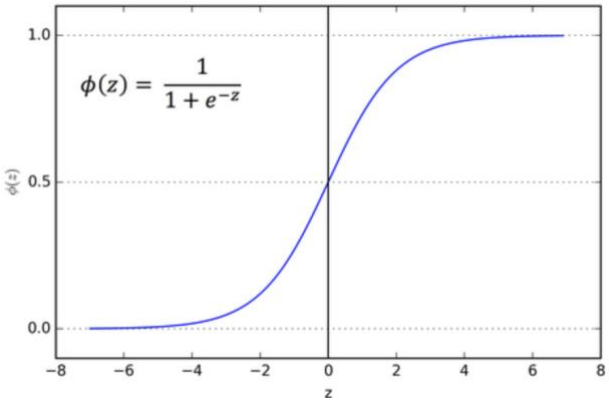
The other activation function I used is the Leaky ReLU which does the same thing as the ReLU function except it has a predetermined slope for negative values (usually y = .01x when x < 0)



The next activation function is the tanh which behaves similar to the logistic sigmoid. Its range is from -1 to 1. This function is typically used for cases where we have 2 classes but for the purpose of this experiment, we will attempt using it.



The last activation function is the sigmoid. Its range is from 0 to 1. It works as a probabilistic model that predicts within that range, one thing to have in mind when using this for a neural network is that it could cause a neural network to get stuck and prevent it from converging.



3.2 Regularization Methods to Prevent Overfitting

First method used is Dropout, this method reduces the interdependent learning from the connected neurons in the network, which basically drops nodes that are not necessary. It also takes longer to converge but it makes sure to learn only the necessary features that are useful for random subsets of other neurons within the network.

Second method is Batch Normalization, this one reduces the covariance shift and helps each layer learn more about itself independently from the other neurons within the network.

Third and fourth methods are a little similar to each other. In L1 it is:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

In L2:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

This helps with the values of the weight vector as it makes the model simpler after regularizing with either of these; it is like a compression of the weights. L2 is usually preferred but I am going to test both.

Finally, I also test a combination of some of these regularization methods: Dropout with Batch Normalization, L2 with Batch Normalization, and L1 with Batch Normalization.

3.3 Architecture of the Networks

For the first layer, I flatten the feature maps; for the second and third layers, I pass them through 2 convolutional layers with different activation function per trial of the experiment, then for the fourth layer I use the standard softmax function. The parameters that I will make sure remain unchanged for the purpose of the experiment are: for the standard optimizer, I use Adam when training, and for the loss I use sparse categorical cross entropy.

3.4 Open Source Imports and Programs Used

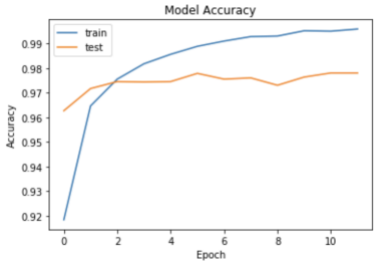
I made use of TensorFlow [2] combined with Keras [3] in order to build the neural network and to apply the regularization algorithms. I also extracted the dataset using these. I wrote all my code in Python 3.0 using Jupyter Notebooks and I got some help from open source code within a website [4] in order to help me write the function for graphing of the results.

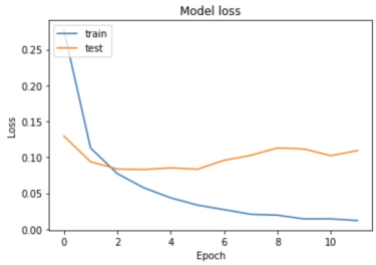
4 Experiment

4.1 No Regularization

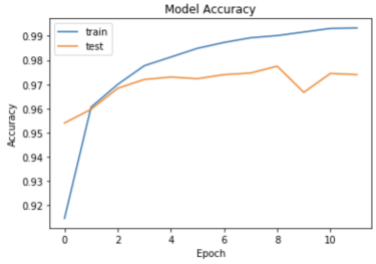
Below I include the curves for loss and accuracy for each respective activation function after 12 epochs.

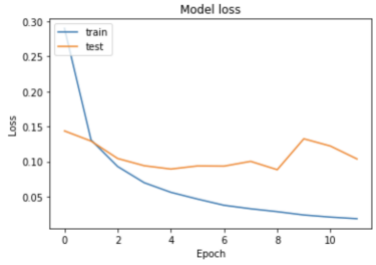
ReLU



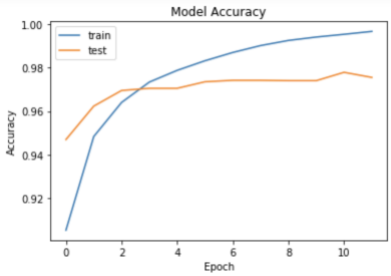


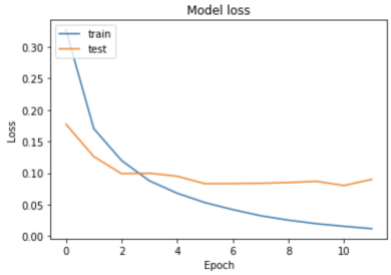
Leaky ReLU



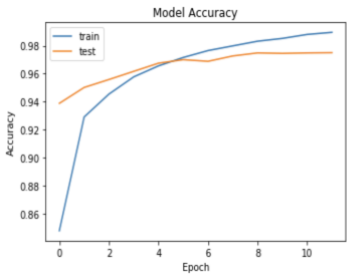


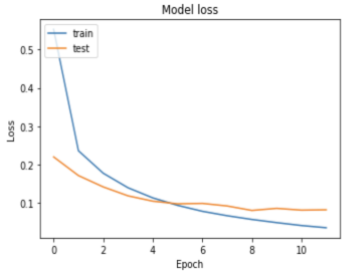
Tanh





Sigmoid

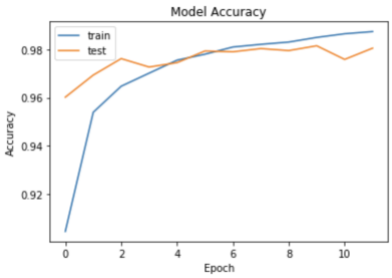


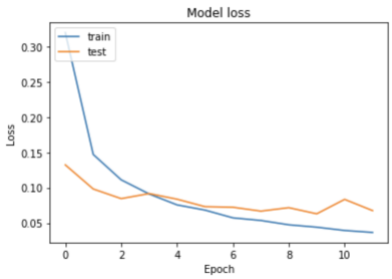


As we can observe, when no regularization technique is used, the best performance comes from the sigmoid, which converges faster than the other methods. Tanh had better performance than ReLU and Leaky ReLU, which seemed to remain stagnant. Let's see what happens when we start using regularization techniques on these

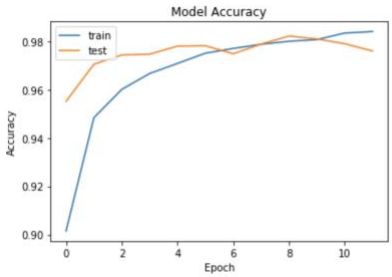
4.2 Dropout

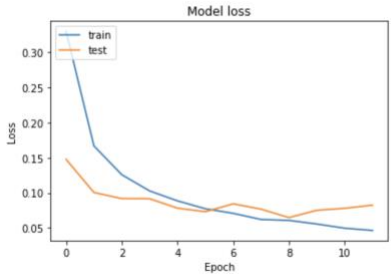
ReLu



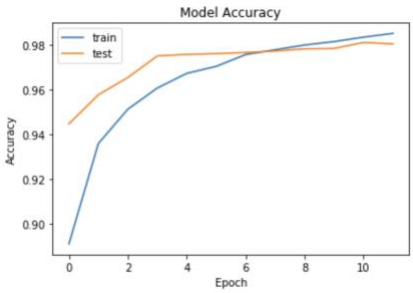


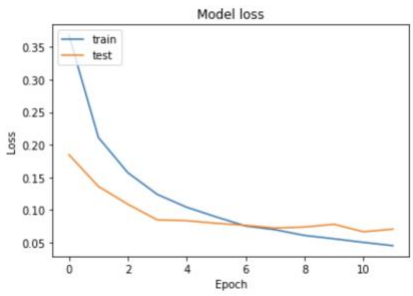
Leaky ReLU



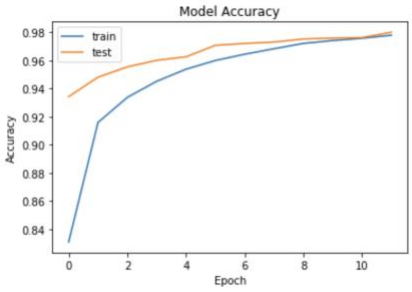


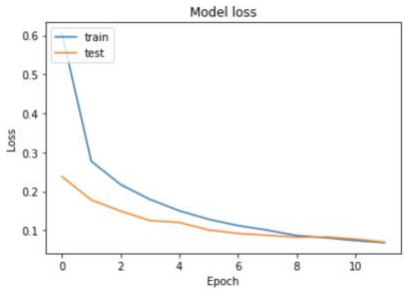
Tanh





Sigmoid



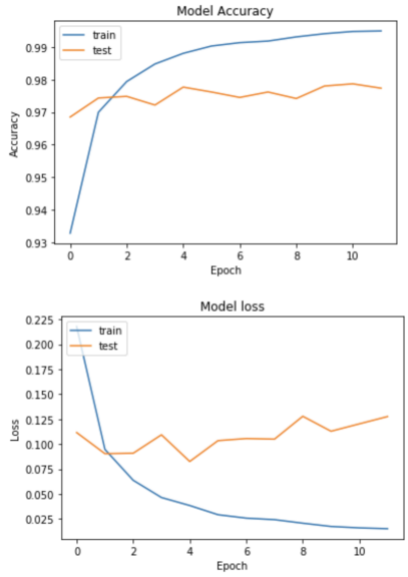


As it turns out, dropout did increase the convergence time for loss and accuracy. It did a lot of improvement in the different activation functions. ReLU and Leaky ReLU both had less fluctuation than before and also converged faster; as it turns out,

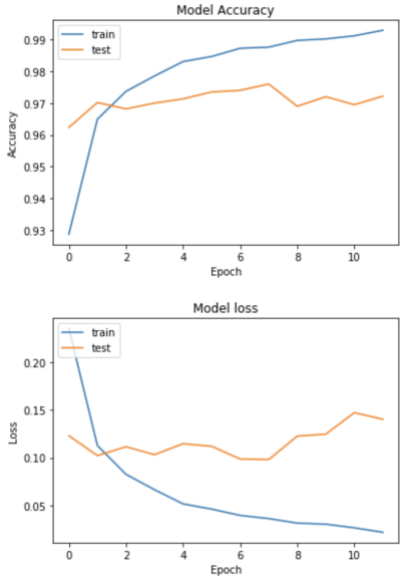
Dropout is indeed a good regularization method for almost any activation function. The Sigmoid function had a really good improvement in its performance, and it seems like overfitting is not even a problem at all for this activation function.

4.3 Batch Normalization

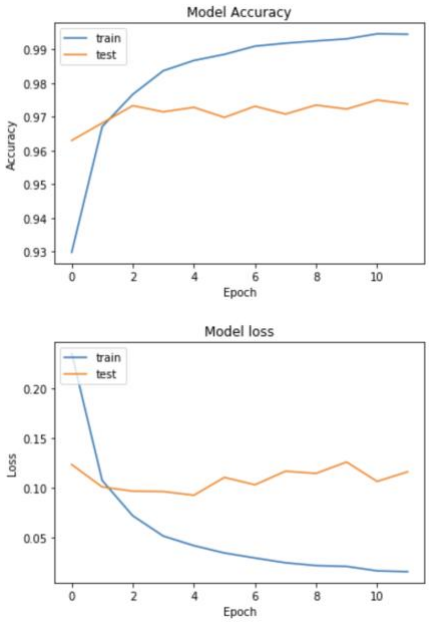
ReLU



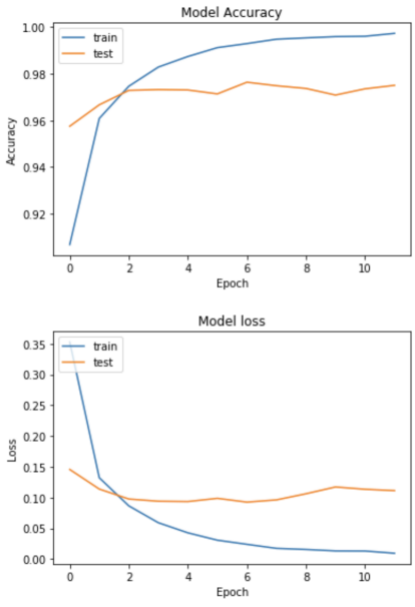
Leaky ReLU



Tanh



Sigmoid

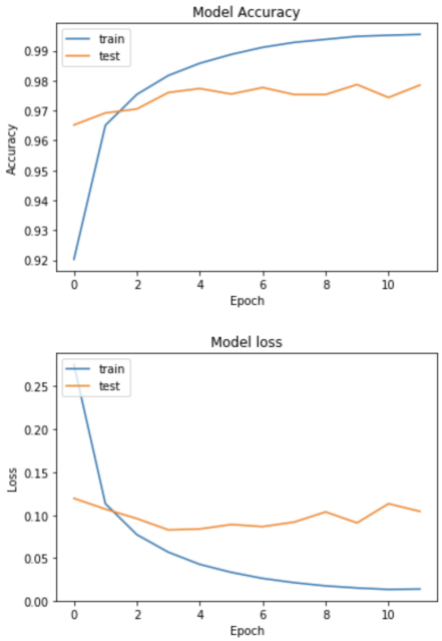


As you can observe, Batch Normalization had little to no improvement on both ReLU and Leaky ReLU. For tanh it seemed to give a lot of fluctuations and also little to no improvement on the loss and accuracy curves. For the Sigmoid function,

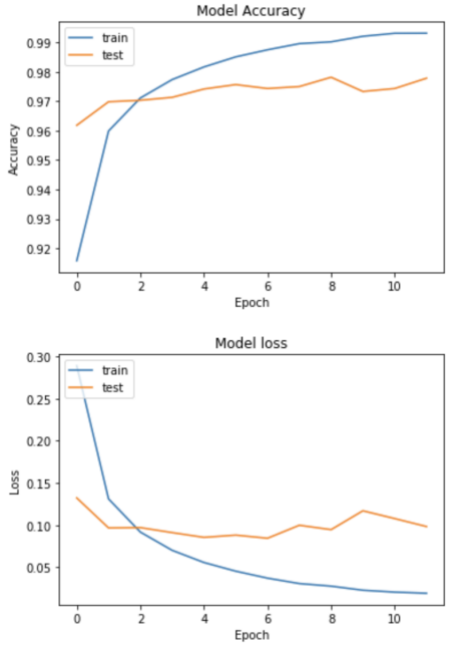
it just got worse. In general practice, Batch Normalization is not a really good way to prevent overfitting when it comes to hand written digit recognition.

4.4 L1 Regularization

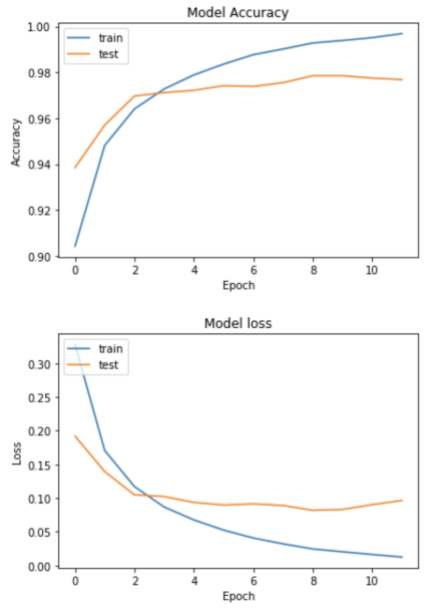
ReLu



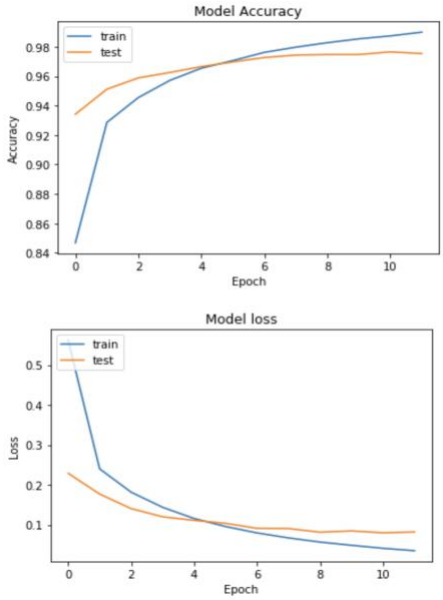
Leaky ReLU



Tanh



Sigmoid

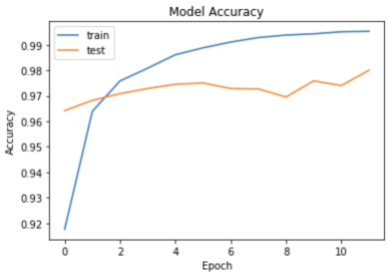


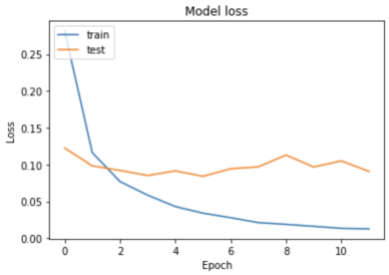
There was no noticeable improvement on both ReLU and Leaky ReLU except for the fact that it had a faster learning rate when using L1 regularization, but at the end, it did little to nothing. Similarly, for Tanh, it had little to no improvement. In the

Sigmoid function, the improvement was really small; L1 did not help it much but it did make the test error less during training.

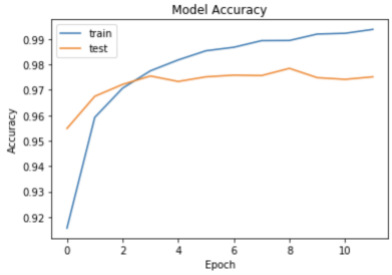
4.5 L2 Regularization

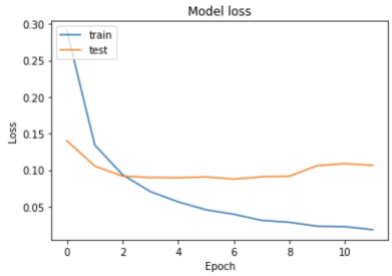
ReLU



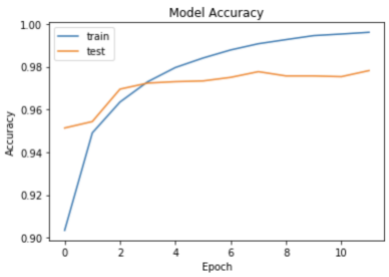


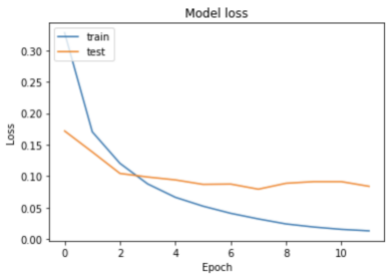
Leaky ReLU



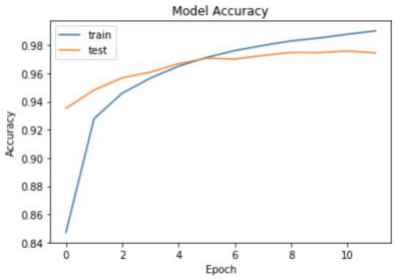


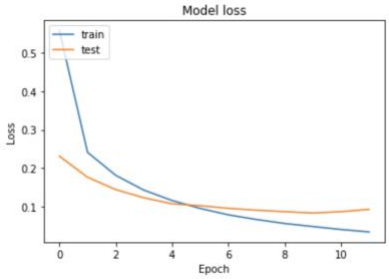
Tanh





Sigmoid



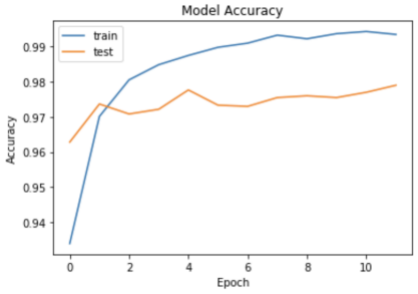


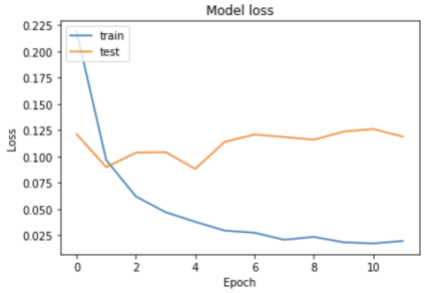
L2 regularization showed little to no improvement for both ReLU and Leaky ReLU. For Tanh it had an improvement but not a significant one. Sigmoid also did not have any significant or noticeable improvement. Overall, L2 works better with ReLU,

Tanh, and L1 works better with Leaky ReLU and Sigmoid; there is no distinction as if one of these is better than the other since they both perform differently given these activation functions.

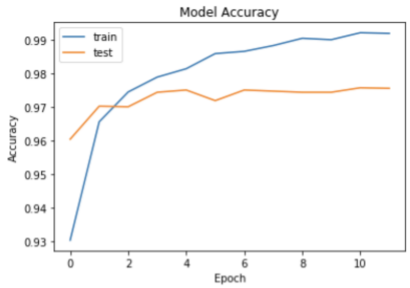
4.6 Batch Normalization and L2 Regularization

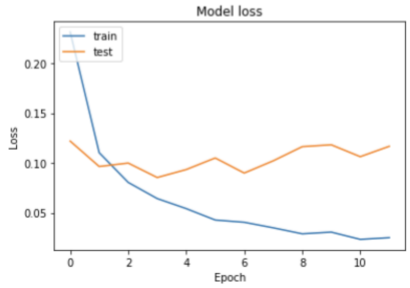
ReLU



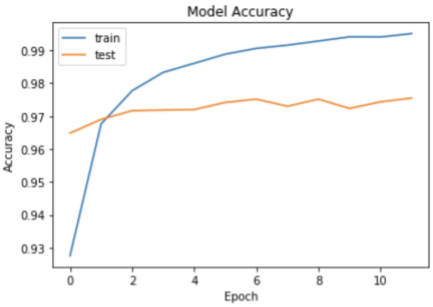


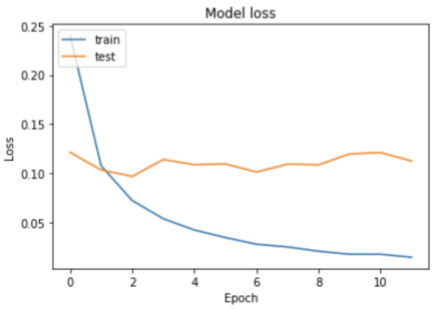
Leaky ReLU



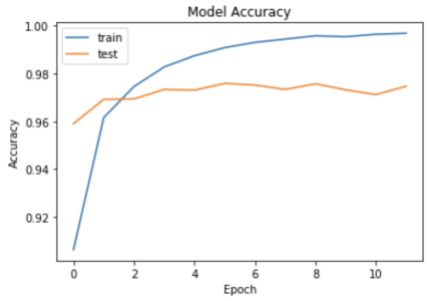


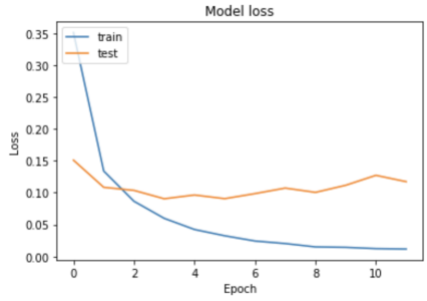
Tanh





Sigmoid

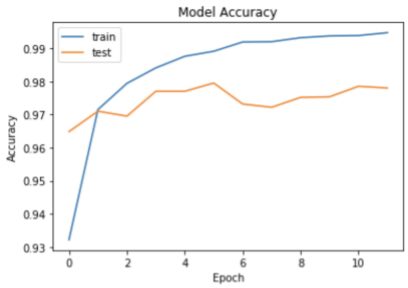


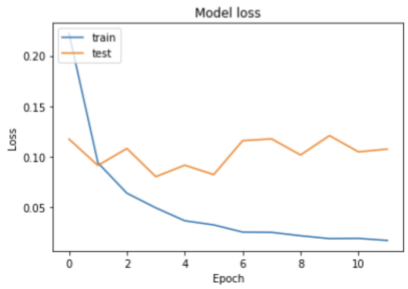


No visible improvement for ReLU and Leaky ReLU. Similarly for Tanh and Sigmoid; in general practice, combining Batch Normalization with L2 Regularization is not a good idea.

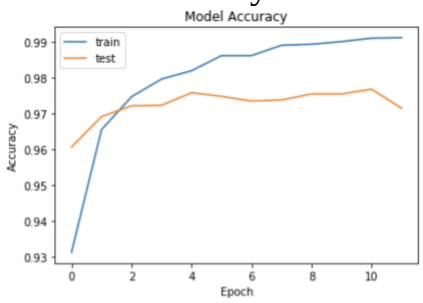
4.7 Batch Normalization and L1 Regularization

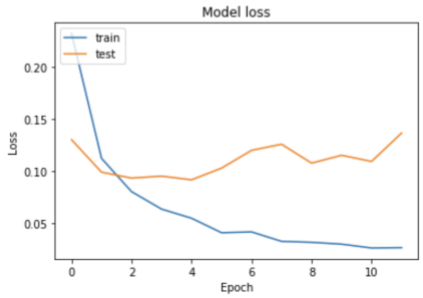
ReLU



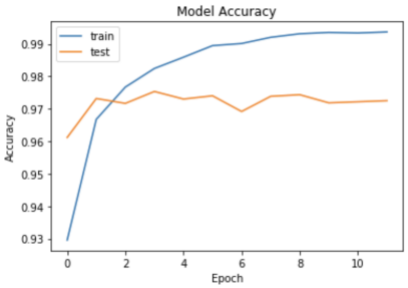


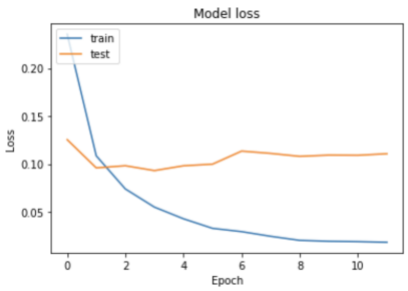
Leaky ReLU



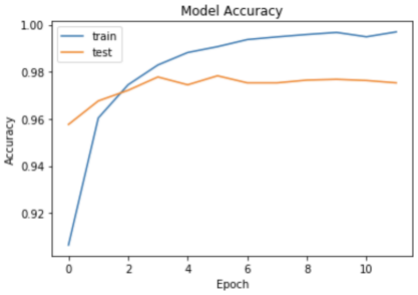


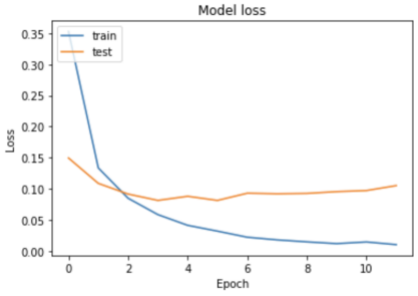
Tanh





Sigmoid

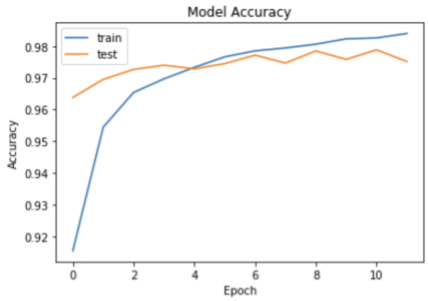


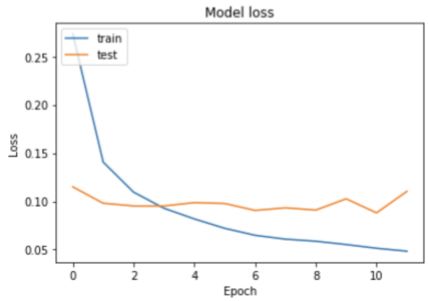


The same conclusion can be drawn as when using Batch Normalization with L2 Regularization. It is not a good idea to use either of these in practice.

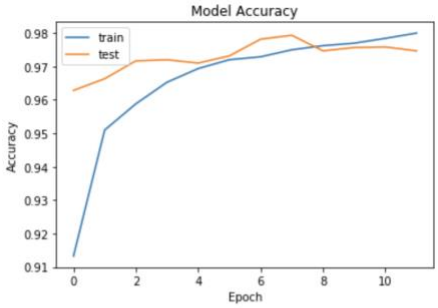
4.8 Batch Normalization and Dropout

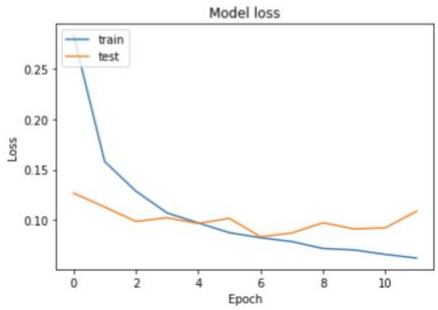
ReLU



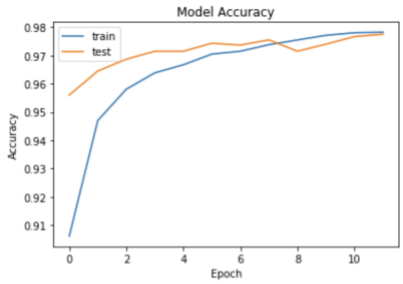


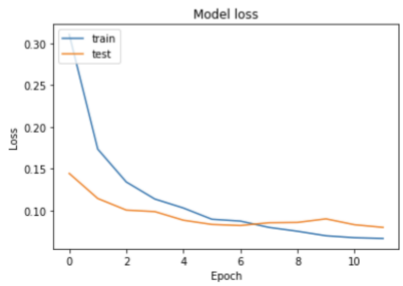
Leaky ReLU



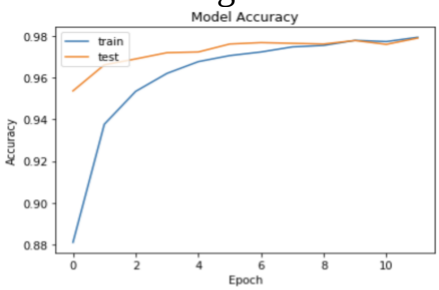


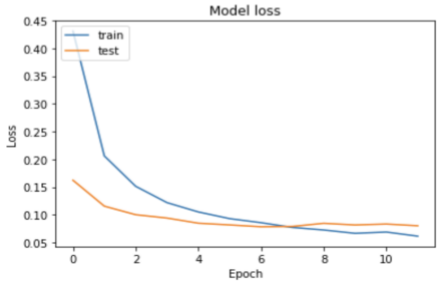
Tanh





Sigmoid





These are some interesting results. Overall, it did have better performance when compared to training with no regularization technique. It had almost perfect performance. In the next section I will compare all the regularization methods in order to draw some important conclusions.

4.9 Comparison Between all Methods Used

Below you are able to see a table of the lowest loss at the 12th epoch for all the experiments.

Name	ReLU	Leaky ReLU	Tanh	Sigmoid
No Reg	.1094	.1036	.0900	.0824
Dropout	**.0676**	**.0825**	**.0708**	**.0700**
L1	.1044	.0984	.0964	.0818
L2	.0908	.1063	.0838	.0930
Batch Norm	.1276	.1404	.1159	.1113
Batch L1	.1073	.1364	.1107	.1050
Batch L2	.1190	.1167	.1124	.1172
Batch Drop	.1103	.1087	.0797	.0804

When working with a large number of epochs, it is good in practice to use Dropout for all the different activation functions. The only thing that got closer to Dropout was Batch Normalization with Dropout and L2 and L1 themselves; these are still ideal but even still, when trying to prevent overfitting, it is important to know that Dropout works best in practice.

L1 and L2 have no notable difference between them as they both performed differently when used with different activation functions. Batch Normalization and its variations are not good in practice for hand written digit recognition unless it is combined with Dropout, which results in smooth testing curves that converge and do not fluctuate as much.

I attached my code along with this report so that you can see all the results in greater detail.

5 Conclusion

For short-run convergence and regularization with hyperparameter adjustment, it is easy to notice which of the common regularization methods works best and which converge faster and smoother for a set of testing samples. Indeed, all these regularization techniques work decently when tested with the actual data points from MNIST after all the training but the focus on this project is to see which regularization technique helps with short-run improvement and which technique avoids overfitting in the short-run best. From these experiments, it was clear that Dropout worked best as there were no fluctuations and all the curves seemed to converge smoothly and achieve a desirable accuracy and test error. Batch Normalization combined with Dropout also gave desirable results; yet not as effective as Dropout itself. In general practice, it is important to make sure we prevent overfitting since the very beginning of the training to avoid having a neural network that does not perform well on data outside the training one. One interesting discovery from this paper was that when it comes to hand written digit recognition, one cannot say that L1 is better than L2 and vice versa as they performed differently on different activation functions. Another interesting discovery is that Batch Normalization by itself and when combined with either L1 and L2, the performance does not improve, it just makes the neural network worse. It is always a good option to stick to Dropout since the performance of the neural network gets maximized when compared to other Regularization techniques/methods.

6 References

[1] Yann LeCun, Corinna Cortes, Christopher J.C. Burges (1998). The MNIST Database of handwritten digits. [http://yann.lecun.com/exdb/mnist/]. Courant Institute, NYU; Google Labs, NY; Microsoft Research, Redmond.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng.
TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from [tensorflow.org].

[3] Chollet, Francois (2015). Keras. [https://keras.io].

[4] Jason Brownlee (2016). Display Deep Learning Model Training History in Keras. [https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/]