CS 156a Bonus Exercise

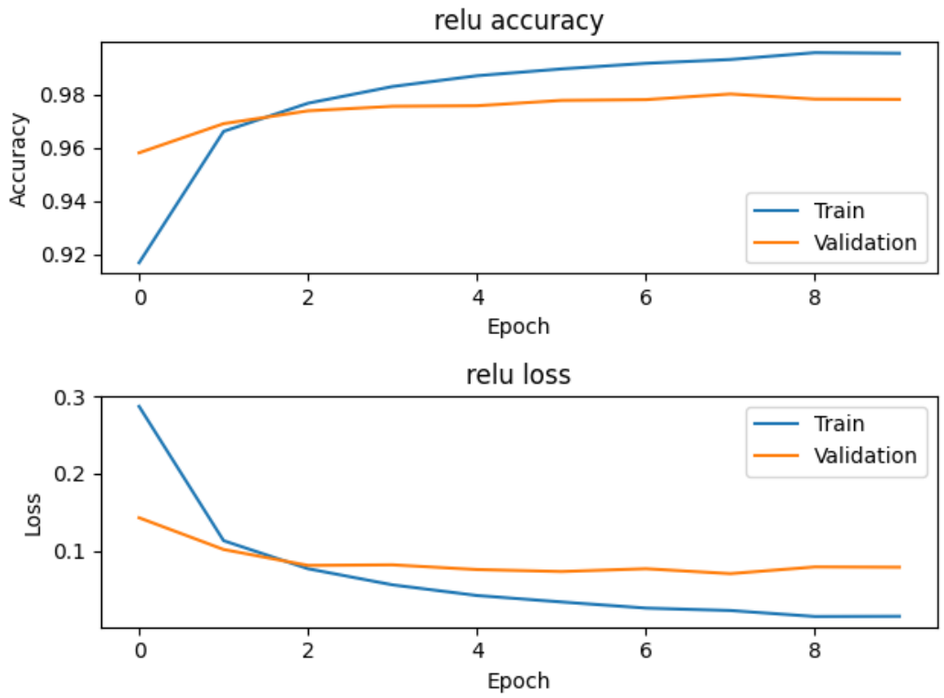
Cason Shepard

January 18, 2023

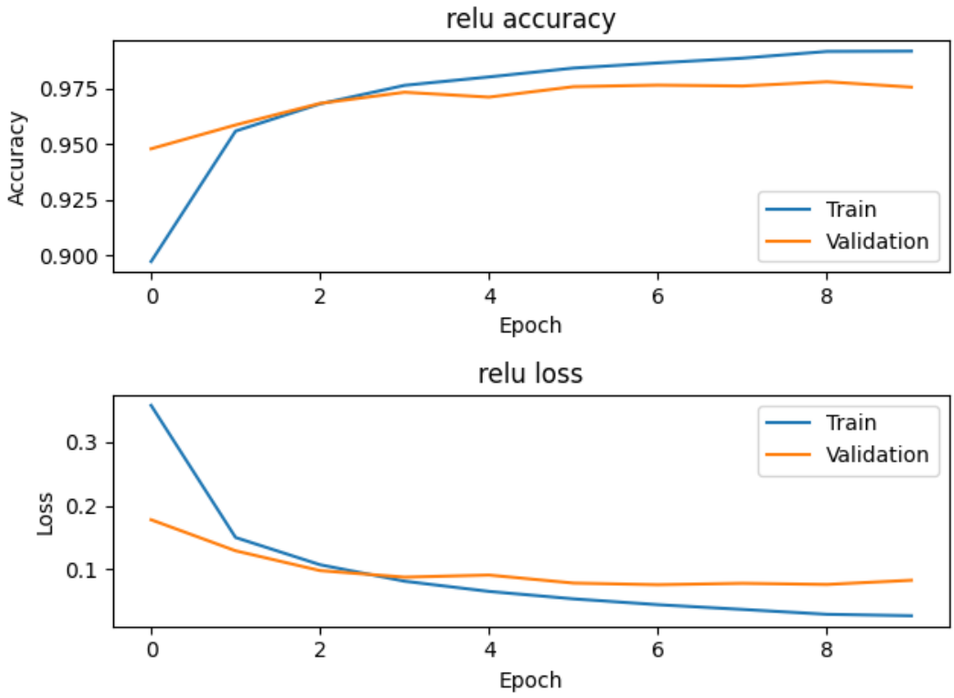
1 Classifying digits with NN's

1.3

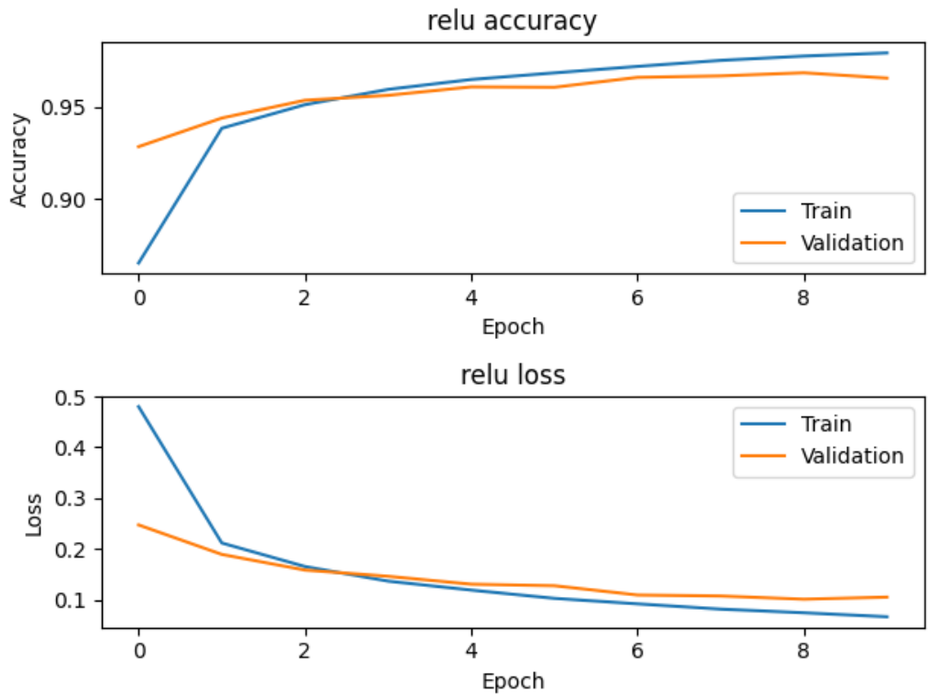
When The number of hidden units is varied, the following accuracy and loss graphs for the training program are generated.



This is the original accuracy/loss graphs using hidden layers of 200 and 100 units.
Total trainable parameters: 178,110



This is the accuracy/loss graphs using hidden layers of 100 and 50 units.
Total trainable parameters: 84,060

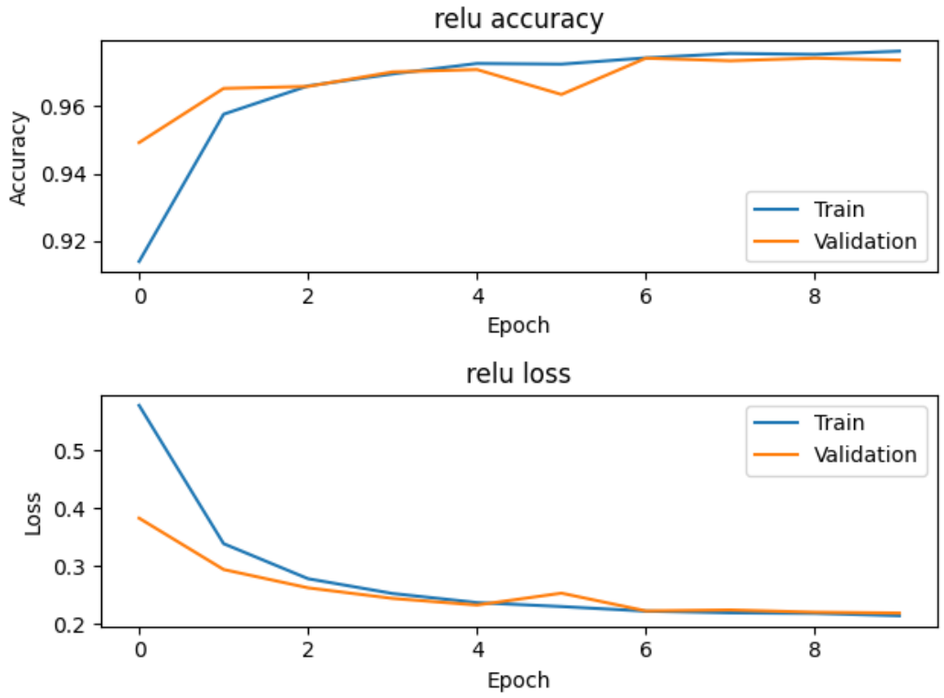


This is the accuracy/loss graphs using hidden layers of 50 and 25 units.
Total trainable parameters: 40,785

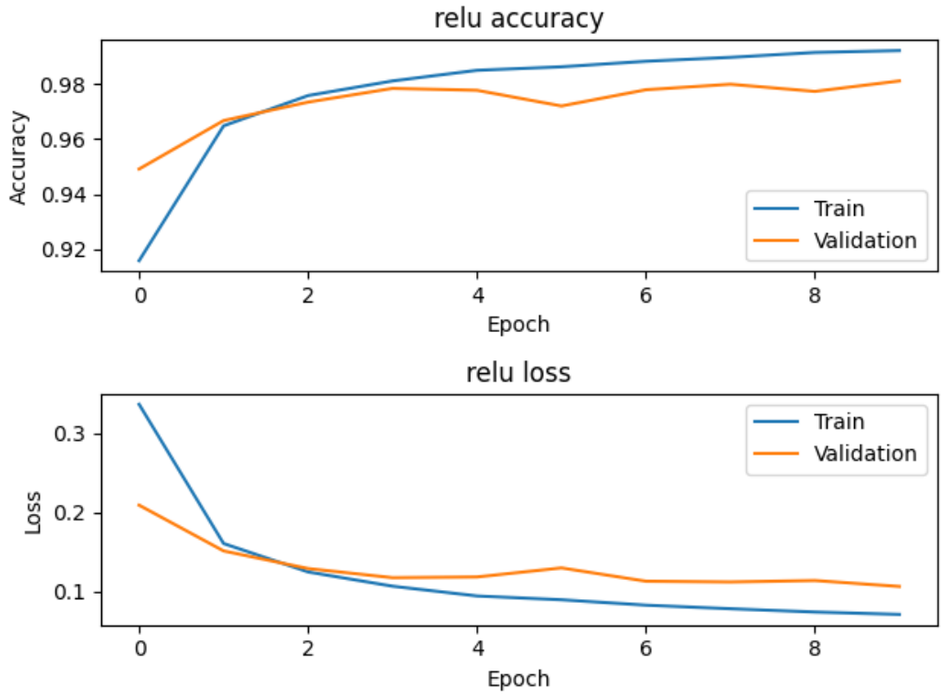
I estimate that the smallest number of parameters to achieve 97% **accuracy is around 80,000.**

1.4

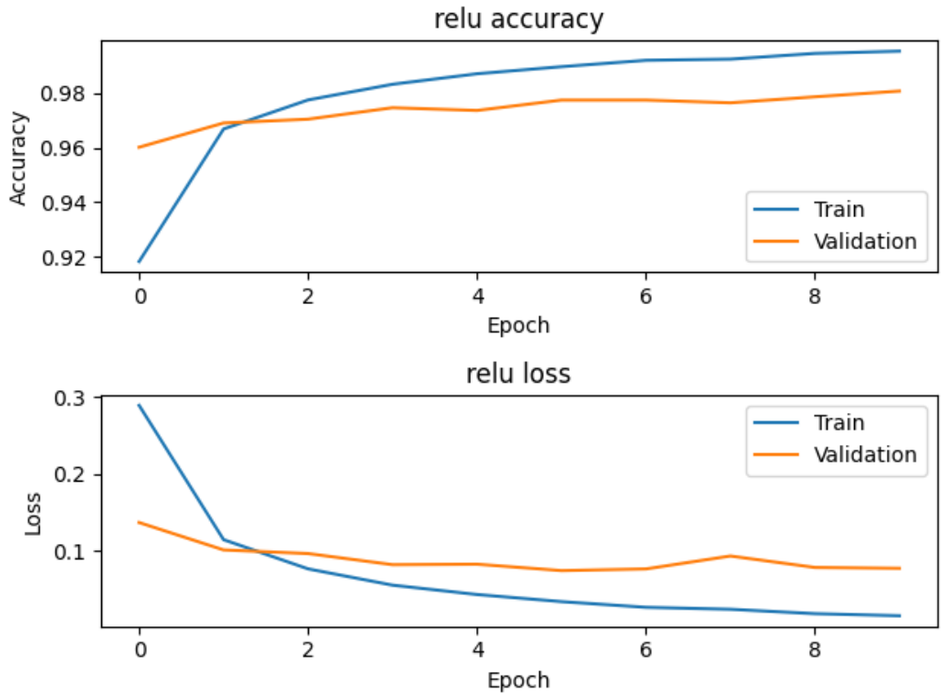
In the following graphs, the REGULARIZATION parameter was varied and the *ReLU* accuracy/loss was plotted.



This is the accuracy/loss plot for $\lambda = 0.001$.



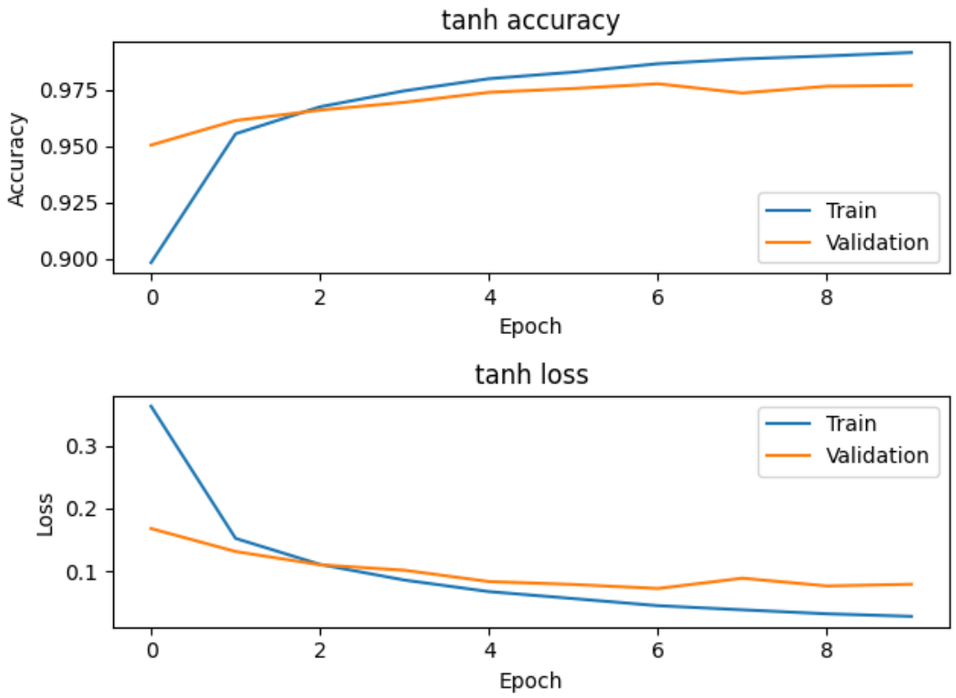
This is the accuracy/loss plot for $\lambda = 0.0001$. **This graph is most similar to the neural network I implemented above with hidden layers of 100 and 50 units respectively.**



This is the accuracy/loss plot for $\lambda = 0.000001$.

1.5

In the following plots, the *tanh* activation function was used.



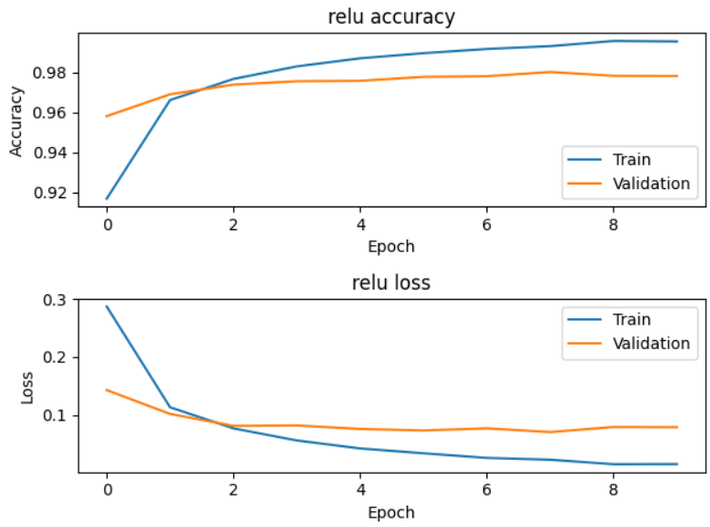
In this plot, the *tanh* activation function was used with the original 100 and 50 unit hidden neural network layers. **Comparing this to the *ReLU* function used over 100 and 50 unit hidden layers, we can see that the *tanh* function converges faster than *ReLU*.**

1.6

The highest validation accuracy i was able to achieve was with an activation of *tanh*, $\lambda = 10^{-7}$, and only a single hidden layer of 200 units. Validation accuracy was 0.9807.

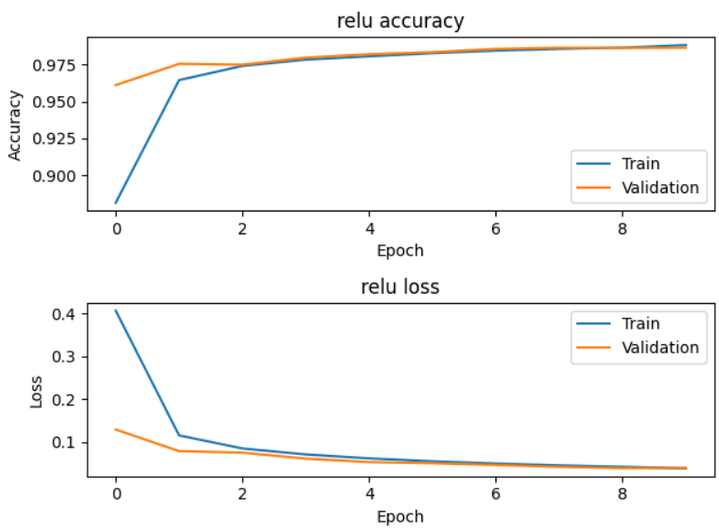
1.7

Dense Neural Net



Total Parameters: 178,110
Validation loss: 0.0760
Validation accuracy: 0.9790

Convolutional Neural Net



Total Parameters: 7240
Validation loss: 0.0391
Validation accuracy: 0.9863

The convolutional neural net had a higher validation accuracy and a lower validation loss than the dense neural network. Furthermore, the convolutional network had less parameters.

2 Generative Adversarial Network

2.2

2.2

1) $L^D = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}[\log D(x)] - \frac{1}{2}\mathbb{E}_{z\sim p_{latent}}[\log(1-D(G(x)))]$

$L^D = -\frac{1}{2}E_{data}[\log D(x)] - \frac{1}{2}E_{model}[\log(1-D(x))]$

$\frac{\delta}{\delta D(x)}\mathbb{E}_{x\sim p}\log D(x) = \int \frac{1}{D(x)}p(x)$

Does p(x) represent $\mathbb{E}$?

$\frac{\delta L^D}{\delta D(x)} = -\frac{1}{2}\int \frac{1}{D(x)}p(x) + \frac{1}{2}\int \frac{1}{1-D(x)}p(x)$

derivative assuming p(x) constant

$2L' = \int \frac{1}{1-D(x)}p_d(x) - \int \frac{1}{D(x)}p_m(x)$

$= p(x)_d \log(1-D(x)) - p(x)_m \log(Dx)$

$L' = \frac{1}{2}p_{data}\log\left(\frac{1}{1-D(x)}\right) + \frac{1}{2}p_{model}\log\left(\frac{1}{D(x)}\right)$

2)

$0 = \frac{1}{2}p_{data}\log\left(\frac{1}{1-D(x)}\right) + \frac{1}{2}p_{model}\log\left(\frac{1}{D(x)}\right)$

$e^{p_{data}\log(1-D(x))} = e^{p_{model}\log\left(\frac{1}{D(x)}\right)}$

$(1-D(x))e^{p_d} = \frac{1}{D(x)}e^{p_m}$

$D(x)(1-D(x)) = e^{p_m - p_d}$

$-D^2(x) + D(x) = e^{p_m - p_d}$

$D^2(x) - D(x) + e^{p_m - p_d} = 0$

$x^2 - x + e^{p_m - p_d}$

$D(x) = \frac{1 \pm \sqrt{1 - 4e^{p_m - p_d}}}{2}$

I do not get the correct answer

$D(x) = \frac{p_{data}}{p_{model} + p_{data}}$

3) This makes sense, as the discriminator should be determined as a combination of the target data and the model of this data. As far as why D(x) is formulated this way, I am unsure.

2.4

The training process seemed to be very efficient, as epochs 1-10 generated very blurry images, but 20-50 began to produce more clear numbers. Past those, the generated images didn't seem to change too much as the epoch number grew. There seemed to be an accuracy threshold that was reached

and then the accuracy did not increase more after that.

Mode collapse is when the generator begins to output the same thing many times. In this situation, the discriminator should be able to recognize this and adjust to reject these repeated outputs. The collapse happens when the generator learns to optimize around the discriminator, so the discriminator stops affecting the output.

I think the GAN provided avoids this by alternating between training of the discriminator and generator. That way the discriminator is forced to change after each generator change, thus, avoiding mode collapse.

3 Feedback

I thought this mini project was enjoyable. Part 1 I ran locally in Windows Powershell, while Part 2 I ran on a Google Colab Notebook. I enjoyed using Powershell to run the scripts and edit them. I wish we could do more of this. Part 1 was slightly repetitive and it got tricky to decipher the axis on the graphs when trying to see if two graphs were similar. I also wish we got more explanation of what the given GAN was doing.