Implementing Convolutional Neural Networks in Cyber Security

University of New Hampshire
College of Engineering and Physical Sciences

Landon Buell[1], Qiaoyan Yu[2]
[1]Department of Physics and Astronomy
[2]Department of Electrical and Computer Engineering
University of New Hampshire, Durham, New Hampshire, USA

Introduction

The Multilayer Perceptron (MLP) is a simple Neural Network architecture that is based off of stacked layers of neurons that interact via linear transformations [2,3]. The network produces output by accepting a set of numerical data in the form of a vector, and passing though the layers with matrix multiplications and vector additions (Eqn. 1) [5]. In this experiment we will explore how a simulated cyber attack on an MLP can affect it's ability to perform. To do this, we insert an attack function into the forward pass mechanism which activates only when a trigger condition is met (Eqn. 2). By simulating this type of event for three kinds of attacks we can closer analyze how an attack on a network alters it's output characteristics. We can also use this information to identify results that may change and thus be indicative of an attack, what characters do not change, and thus may allow an attack to go unnoticed and cause further damage. This research will demonstrate initial concepts that Network designers can use to better prepare proactive and reliable defense mechanisms against attacks.

Mathematical Model & Scikit Learn Implementation

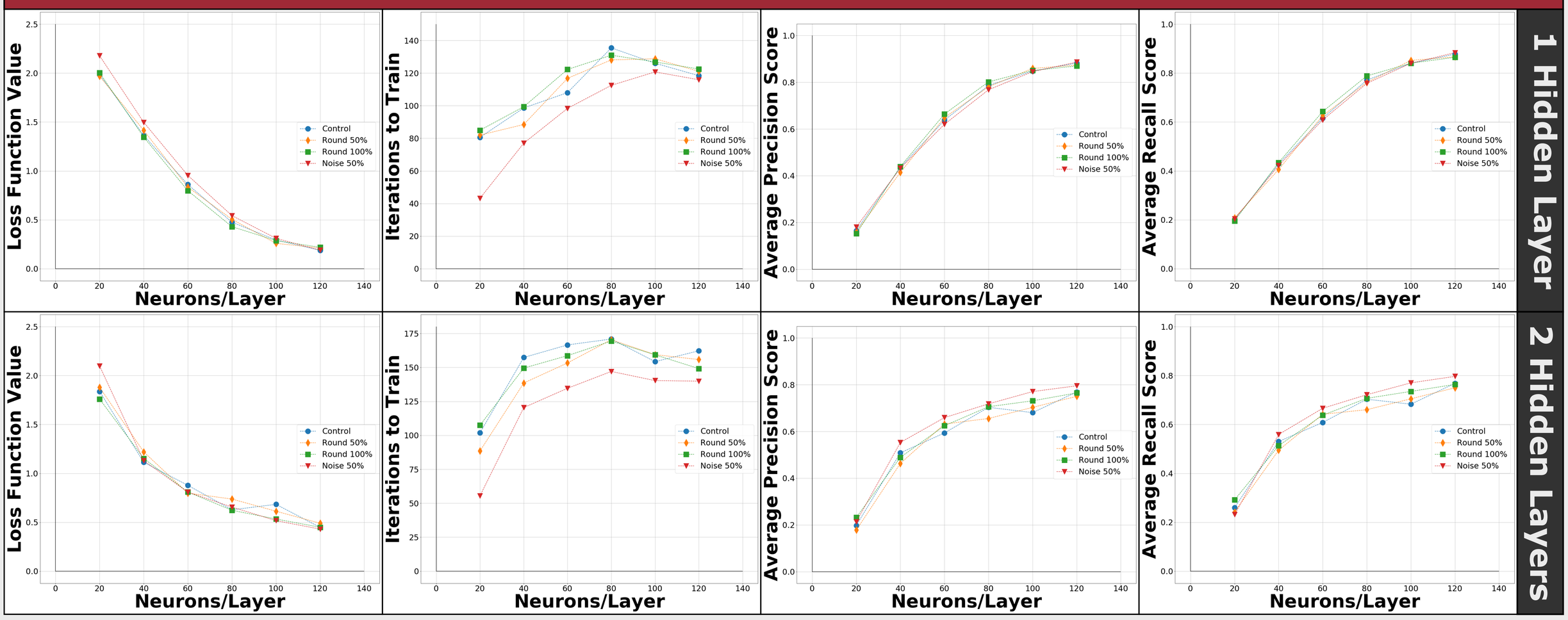
$$\vec{x}^{(l+1)} = f\left[\hat{W}^{(l)}\vec{x}^{(l)} + \vec{b}^{(l)}\right] \qquad (1)$$

$$\vec{x}^{(l+1)} = f\left[A\left(\hat{W}^{(l)}\vec{x}^{(l)}\right) + \vec{b}^{(l)}\right] \qquad (2)$$

The forward propagation system of a Multilayer Perceptron is governed by the repeated matrix-vector equation (eqn. 1). Due to this being the ultimate decision-making process, it is a reasonable target for a cyber attack. This attack will act as an additional operation in this forward pass system (eqn. 2). By inserting an *attack function* into the forward pass method, we can mimic a potential attack based on four arguments: The layer activation vector, $x^{(l)}$, the layer weighting matrix, $W^{(l)}$, an attack variant, V , and a trigger condition, *T*. If the trigger condition argument is *false*, then no attack takes place, and the activation are returned as normal in equation (1). Otherwise, the attack function activates and changes the values produced.

Python's open-source machine learning module *scikit learn* (sklearn by convention), provides a Class called "MLPClassifier" which stands for "Multilayer Perceptron Classifier" [1,7]. This allows for easy implementation of a conventional MLP, with error handling and easy control of various hyperparameters in this project.

Experiment Results



The Attack Function

Three different variants of attack functions will be compared to a "control" model that serves as a performance baseline. The variants tested are visualized above, and explained below.

. **Rounding Layer Activations to 0 decimals of accuracy, 50% of the time ("Round 50%")**
This operations mimics any sort of attack that seeks to generally reduce the numerical accuracy of mathematical operations. Each floating-point number is rounded to only the nearest integer value. In each layer advancement, the trigger True/False condition is randomly generated with 50/50 probability.

. **Rounding Layer Activations to 0 decimals of accuracy, 100% of the time (Round 100%)**
The same as above, however in each layer, the trigger True/False condition is always set to return *true*. This means that at every layer advancement, the numerical accuracy is also rounded to the nearest integer value.

. **Adding Gaussian noise to Layer Activations , 50% of the time (Noise 50%)**
This operations mimics any sort of attack that seeks to introduce randomly generated noise. In this case, we have drawn from a Gaussian distribution with $\mu = 0$ and $\sigma = 0$. This generated noise is in the form of a vecotr and is added to the activations. Gaussian noise is a common method to loosly model added distortions [4].

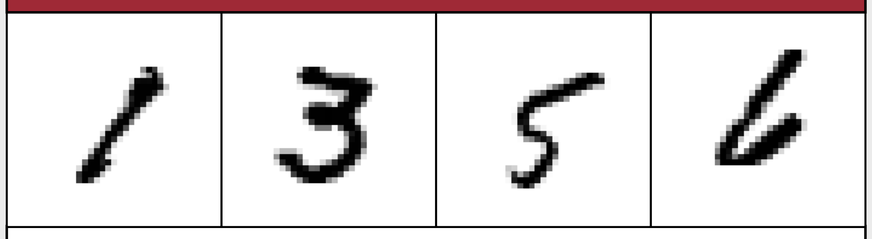
Neuron Densities & Architecture

For each of the three functions, we test 12 variations of neuron densities and layer numbers. For neuron density we use the values 20, 40, 60, 80, 100 and 120 neurons per layer and this is executed for one hidden layer and again for two hidden layers. The first row of plots above indicates results for single layer models, and the second row is for double layer models. To further ensure validity of any results, each of the 12 variations has 100 models were trained under that particular density. Each one is given the exact same set of data and hyper parameters—the only difference being the initial starting values in the weighting matrices. This means that every data point plotted in the figures above is a uniform average across 100 models subject to the same conditions.

Experimental Conclusions

[I have these written down on paper, but need to formally organize them before including them in this DRAFT (v1)]

Dataset



The MNIST (Modified NIST) data set was used in all model variations for this experiment. This is a common choice among validating neural network algorithms [2]. It consists of 70,000 handwritten figures that have been labeled as digits between 0 and 9. Each image consists of 28 x 28 pixels, with entries from 0 - 255 corresponding to the brightness of that pixel. When flattened and transposed, a single image produces a (784 x 1) feature vector, which is the network's input layer in all cases. The first 12,000 images of this data set were used to train each model, and the next 4,000 images were used to test each model. Some examples from this dataset are pictured above.

References

[1] Buitinck et al. API design for machine learning software: experiences from the scikit-learn project, 2013
[2] Géron Aurélien. Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly, 2017.
[3] Goodfellow, Ian, et al. Deep Learning. MIT Press, 2017.
[4] James, Gareth, et al. An Introduction to Statistical Learning with Applications in R. Springer, 2017.
[5] Loy, James. Neural Network Projects with Python: The Ultimate Guide to Using Python to Explore the True Power of Neural Networks through Six Projects. Packt Publishing, 2019.
[6] McCulloch, Warren S., and Walter Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity." The Bulletin of Mathematical Biophysics, vol. 5, no. 4, 1943, pp. 115–133.
[7] Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

This Research was funded by *[source here]*