Modeling Attack Functions with a Multilayer Perceptron Network

Landon Buell

19 May 2020

Introduction

In the modern world, neural networks have seen a great emergence due to progress in advanced numerical problem solving and the need for high- performance computing. Such widespread use makes a neural network a target for potential external attacks or other disruptions. Such attacks could present performance drops or security compromises to any system that relies on their functionality. Currently, few networks have systems in place that allow for the identification or correction of intended attacks.

To explore how Neural Networks perform when subject to a series of manipulations, we will create an image classification multilayer perceptron network. This feed-forward model is composed of layers of neurons, $\vec{x}^{(l)}$ connected by a system of weights, $\hat{W}^{(l)}$) and added with a column of intercepts $\vec{b}^{(l)}$ to pass data from one end to the other. This standard process is mathematically modeled by the matrix-vector equation:

$$\vec{x}^{(l+1)} = f\Big(\hat{W}^{(l)}\vec{x}^{(l)} + \vec{b}^{(l)}\Big) \tag{1}$$

Where f is an *activation function*, the superscript (l) and $(l+1)$ indicate a layer index, $\vec{x}$, $\hat{W}$, and $\vec{b}$ are the layer activation, weighting matrix, and bias vector respectively.

To model an attack function, we insert an extra function into this system that manipulates the expected outcome of the matrix-vector product. This has the effect of inhibiting the network from making an accurate decision based upon any input samples provided. The newly inserted function, called an *attack function* changes the feed-forward model, eqn. (1) such that:

$$\vec{x}^{(l+1)} = f\Big(A\big(\hat{W}^{(l)}\vec{x}^{(l)}\big) + \vec{b}^{(l)}\Big) \tag{2}$$

Where A is the attack function.

The attack function is also based on an *attack type* and a Boolean *trigger condition*. If the trigger condition is defined as *false*, then no attack occurs, thus eqn. (1) applies. If *true*, then the attack commences on that layer pass, and eqn. (1) is replaced with eqn. 2). In the latter sections of this paper we explore how different variants of the *attack type* is implemented, and how it affects the layer activations.

Network Sizes and Parameters

To ensure a sufficiently wide range of results, we applied all attack types to a variety of Network architectures. In all, 24 different MLP variants were tested on. The simplest way to vary the networks is the adjust the number of hidden layer in the model, and the number of neurons in each hidden layer. These are also referred to as *network depths* and *neuron density* respectively. We test four different depths, being 1, 2, 3 and 4 hidden layers and size different densities, being 20, 40, 60, 80, 100 and 120 neurons per layer. Combinations of these create the 24 network variants. Additionally, for each variant, we create 50 models and record metrics as averaged uniformly over them.

To ensure consistency across all models, several *hyperparameters* in the MLP have been set. This is done in scikit learn by providing arguments to the classifier when initializing the class instance.

- **Activation Function**: The *Rectified Linear Unit* Activation Function (ReLU), was used in all network instances. It is defined:

$$\mathrm{ReLU}(z) = \max\left(\{0, z\}\right) \tag{3}$$

  This is represented by f in equations (1) and (2).

- **Solver**: Stochastic Gradient Descent. This is an optimization method based on using iterations over the training data set to update parameters in a way such that it reduces the error given by the difference between a given output and an expected output in a training batch.

- **Batch Size**: A mini batch is a subset of training data used in each training iteration. For this experiemtn, a batch size of 100 samples was used. After each batch is passed through the network, the training set is then shuffled again and a new set of 100 samples are drawn at random for the next training iteration.

- **Maximum Iterations**: To prevent models from potentially being stuck in indefinite loops, a maximum number of 400 iterations per model instance was allowed. In practical training circumstances, this also prevents a network from being overfit on a single collection of data. If a model did not converge on a sufficient set of parameters, the training process halts regardless.

- **Tolerance**: The tol

Baseline Model

Before any modification to the MLP network, we need to establish a baseline or a control model. This model was run using the standard unmodified *Multilayer Perceptron Classifier* implementation provided by the Python Library *Scikit Learn* 0.22.1.

Rounding to the Nearest Integer Value

Adding Noise drawn from a Gaussian Distribution

Muting the Most Significant Bit

Concluding Remarks