Preliminary Implementation Report on Malware Classification using Back-Propagation based Artificial Neural Networks

Anurag Datta Roy
anuragdattaroy@gmail.com

Rajat Tripathi
rajat.kumar6@learner.manipal.edu

Suyash Shetty
suyash.shashikant@learner.manipal.edu

Manipal Institute of Technology
Department of Computer Science and Engineering

1 Architecture

Our preliminary, basic implementation model consists of a rudimentary multilayer perceptron, that is a feedforward artificial neural network that maps a set of input data onto a set of appropriate outputs. The model takes raw, preprocessed image pixels as input and tries to map them to their corresponding malware class.

The architecutre is 2-layer neural network, consisting of one input layer, one hidden layer and one output layer. The details of architecture are given below:

1.1 Data Preprocessing

For our first implementation attempt, we use our existing dataset as is. In the previous report [1], we describe in detail how we obtain the dataset and certain preprocessing techniques that we apply to make the data viable for use. From [1], we have two NumPy arrays containing raw image pixels and the corresponding image labels respectively. For this attempt, we perform no data augmentation techniques and work with only a meager number of 251 image representations of our malware binaries. Setting an arbitrary threshold h of 20736, we get images of size $144 * 144$, which are flattened into 1D arrays. The dataset is then split into train and test sets in $70\% - 30\%$ ratio, followed by the application of the train set mean subtraction on the entire dataset as a means of some basic preprocessing. As a result, we get a train dataset of $(175, 20736)$ on which on our primary model is trained and then tested on a test dataset of $(76, 20736)$.

1.2 Layers

As mentioned in the previous layer, the input layer has a dimension of $(175, 20736)$. The dimensions of the weights and biases of the first hidden layer are $(20736, 128)$ and (128) respectively. The output of this layer, $hidden = W_1 * X + b_1$, where W and b are the weights and biases respectively, has a dimension of $(175, 128)$. This output is then passed to a non-linear activation function, the Rectified Linear Unit (ReLU) [2], which performs the operation $hidden = max(0, hidden)$ followed by the passage through a *dropout* [3] layer which acts as a regularizer. The dropped-out hidden layer is again multiplied by the weights and biases of the final, output layer, giving the output layer as, $output = W_2 * hidden + b_2$ which contain the normalized probabilities of each image belonging to one of the 6 classes. Its dimension is $(175, 6)$.

2 Results

Table 1: Result of training

Step	Loss	Training Accuracy
0	215.675	0.430
100	1.623	1.00
200	0.725	1.00
300	0.00	0.99
400	0.00	1.00
500	0.00	0.99
600	0.103	1.00
700	1.104	1.00
800	0.00	1.00
900	0.00	1.00
1000	0.00	1.00
1100	0.00	1.00
1200	0.00	1.00
1300	0.00	1.00
1400	1.18	1.00
1500	0.00	1.00
1600	0.00	0.99
1700	0.00	1.00
1800	0.00	1.00
1900	0.00	1.00
2000	0.00	1.00

When evaluated on the test dataset, the model has an accuracy of 87%.

As we can see from the Table 1, the loss quickly decreases as we train the model and the accuracy increases. But due to the small size of our dataset, the model starts to overfit on our training dataset.

Future implementation models will deal with this overfitting by augmenting the dataset and performing certain data transformations such as *Principle Component Analysis* etc. Different neural network architectures will also be tested to find models which generalize well and suffer from very low overfitting.

References

[1] A. Datta Roy, R. Tripathi, and S. Shetty, "Artificial intelligence project progress report," 2016.

[2] G. E. Hinton and V. Nair, "Rectified linear units improve restricted boltzmann machines."

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.