Predicting Malware Presence with Neural Networks

Final Data Mining Project Report

Personal Information Scrubbed

Introduction

*The Competition*

In Spring of 2019, Microsoft posted to Kaggle a data set they compiled of millions of entries of systems they recorded from surveys. With it, they offered a $10,000 prize to the person that submitted the model most capable of predicting which machines were affected by malware.

*Goal and Motivation*

Though our team was more interested in the challenge than any pretense of prize money, by the time our team began work on this project in earnest, the competition was over, with the most accurate model achieving only 71% accuracy. However, that did mean that contestants had begun sharing the models they had used, and that became a great tool. With that advantage, despite our amateurism, we set out to build a model that would achieve accuracies near the best.

0 FORMALITY

Last thing before jumping in: Initial specifications, Resources, and Related Work established here.

0.1 Acknowledgements

It is prudent at the outset to make acknowledgements, as the information gathered by other contestants in this contest was instrumental as early as the outset of the project, and many steps toward our final model. Official citations are used and refences provided in the final section.

Chris Deotte: Data Scientist & Researcher at UCSD

He provided excellent summaries of the models he implemented, comparing performance, and thus helped us choose which to go with.

YouHan Lee: PhD student at KAIST

His breakdown of the raw data was second to none and eliminated much of the busy work of the discovery process.

Dr. Xintao Wu: Professor in CSCE at U of A

Most importantly, the opportunity to work on this project would not have been possible without our course professor. His imminently appreciable excitement of Data Mining was the fuel necessary to cope with all the difficult concepts presented in the course. Our most fundamental understanding of all that will be presented was provided by Dr. Wu.

0.2 Hardware

Preprocessing and Modelling, Training and Testing were performed on two different machines, here-so listed respectfully:

System 1

|  |  |
| --- | --- |
| CPU | Intel i5-6600K @ 3.5GHz |
| GPU1 | NVIDIA RTX 2060 6GB GDDR6 |
| GPU2 | NVIDIA GTX 970 4GB GDDR5 |
| RAM | 16 GB DDR4 @ 2133 MHz |
| STORAGE | Samsung 970 SSD PRO NVMe 256GB |
| OS | Windows 10 |

System 2

|  |  |
| --- | --- |
| CPU | Intel i9-9900K 3.6-5GHz |
| GPU | NVIDIA GTX 1080Ti 11264MB GDDR5X |
| RAM | 16GB DDR4 @ 3000 MHz |
| STORAGE | Samsung 970 PRO SSD NVMe 1TB |
| OS | Arch Linux |

0.3 Software

All our code was written in Python, and we used the Pandas library for preprocessing, and the Keras library for modelling and testing. Graphs were compiled by TensorBoard.

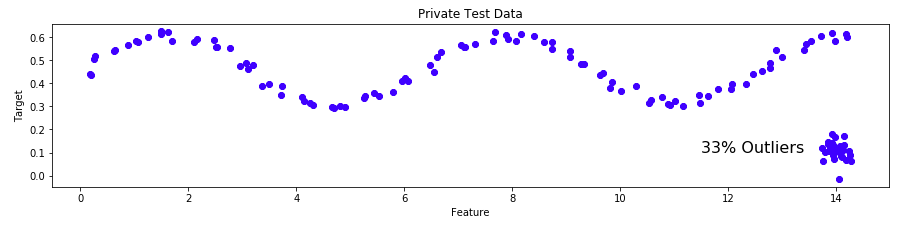
Design and Implementation

1 Data

Millions of entries are provided publicly in a training set and a testing set, with the latter missing the label entries. Kaggle additionally holds a private data set that your model predicts upon submission. The public training data are labeled half true and half false. Microsoft also provided an incomplete description of the 84 attributes, which can be roughly grouped into specifications on hardware, software, user preferences, and user location. All but a handful are assuredly nominal, with the others being discrete. Many attributes were very sparse (>10% missing values) with the following being the most notable:

|  |  |
| --- | --- |
| **Field** | **Amount Missing** |
| PuaMode | 99.97% |
| Census\_ProcessorClass | 99.59% |
| DefaultBrowsersIdentifier | 95.14% |
| Census\_IsFlightingInternal | 83.04% |

Of note, 33% of the private testing data set contains outliers, which account for the relatively low accuracy of even the best models submitted (~70%). Our project would likely have greatly benefited from more time spent exploring the data or reading articles from other users that had, especially in making the data size more manageable, as well as guiding our approach to the model.



**[Image provided by Chris Deotte and found in this article:** [**https://www.kaggle.com/c/microsoft-malware-prediction/discussion/84745**](https://www.kaggle.com/c/microsoft-malware-prediction/discussion/84745)**]**

2 PREPROCESSING

Our hastened approach to preprocessing was to treat every attribute as nominal. While there were a handful discrete attributes (e.g. Screen Size, System Storage Capacity), it was hard to justify the comparable nature of continuous data types in the context of malware detection. While this allowed us to handle missing values efficiently, by treating them as a category until themselves, it would be nigh-impossible to one-hot encode everything, as there are 923,777 unique categories.

2.1 Statistical One-Hot Encoding

We mitigated much of this bloat by statistical one-hot encoding, which disregards underrepresented categories and those that do not appreciably match a presence of malware or lack-thereof. The workhorse equation of this method is provided by the when performing **the following** **test:**

**Or informally**, the occurrence across all entries that when a given category is present, our target label is also True.

The test is generalizedvia **Central Limit Theory** and subsequently rearranged by expanding the standard deviation operation **in the following**

which is finally organized into the comparison

**that is the central condition** around which our statistical one-hot encoding function is written. We provide the *-statistic* as the constant 0.5, while the is the number of instances the given category occurs and computed in the manner expressed above. When the condition is True, we include the category in our final encoded data-frame, otherwise the category is completely disregarded, as it is deemed non-effectual.

To wonderful effect, using the process, the one-hot-encoding process only 560 new Boolean attributes, as opposed to the nearly 1 million that the default one-hot encoding technique did. Although we expectedly did not have the hardware to test the accuracy of predicting based on default version due to the enormous size it would have created, mathematically and logically, our approach should have not costed us enough in accuracy to discount the greatly appreciated utility of a slimmer dataset.

2.2 Frequency Encoding

Four specific attributes, each containing the present versions of major software, were instead chosen to be Frequency Encoded: the ratio of the occurrence of a given category to that attribute’s most common category’s occurrence. This has the effect of transforming categorical attributes into discrete continuous ones; Fast and informative. The aforementioned specific attributes were chosen for two reasons:

1. The most common versions would be most commonly targeted by malware producers. Often the high userbase of a version is enticing enough to offset the higher likelihood that the version will entail higher security as well.
2. The prediction model would entwine its consideration of the safety of these versions with that popularity.

3 MODELING

We were immediately anxious when the models we build for the course’s practice project – decision trees, Bayesian models, k-nearest neighbors, and k-means – ended up yielded middling results, barely better than a guess. As those constituted the majority of our understanding of data mining, we quickly began studying other contest submissions. Quickly, it became clear to us that the front runners for effective models in this competition were all using Neural Networks and Light Gradient Boosting Machines. Knowing little about either, and with only enough time to implement one, we treated the choice as a toss-up, and went with the buzzword.

In some ways, Neural Networks are much simpler than one imagines of such an impressive name, and in other ways just as complicated as expected. The sheer number of parameters involved is quite intimidating. How many layers and of what type? What cost function? What are your activation layers? These were all the questions we expected going in. We didn’t know it was important to choose an approach an appropriate number of epochs, batch size, a fitting optimizer, etc.

3.1 Sequential Neural Network

The simplest form of a neural network is a sequential one, a stacking of layers from start to finish, each receiving some inputs from the previous and transforming the data in some way before providing that data as the inputs to the subsequent layer.

Given this would be the first neural network we would ever construct (that did not consist of purely the default parameters of a popular python data mining library’s model), we wanted to limit the factors we to tweak in order to obtain a focused understanding of the core elements.

The final model and each of its parameters is detailed below.

3.2 Loss Function.

Perhaps tried and true is applicable even to the most advanced topics, because Binary Cross Entropy seemed to be the perfect fit from the instant we read about it and proved to be just that in the long run. Being the utility function which our bot would be desperately attempting to maximize, it was surprising how simple a process this choice was, especially in comparison to many others.

Cross entropy is logarithmic and thus encourages rapid learning early on while discouraging the network from second guessing itself once it gains some insight. It also supremely highlights areas that need great improvement. Cross entropy comes in a couple varieties, but Binary our only option given our label is a binary value.

3.3 Optimization

Adam was the best performing optimizer after trying each listed in Kera’s documentation. We would have settled for highest accuracy, but it was also the quickest to run by a wide margin. Such is explained due to its expertise in handling big, noisy, and sparse data. It is a stochastic optimizer, ergo randomly distributed an unable to be precisely predicted, which synergizes with a couple of our layer choices going forward.

3.4 Layers

Our final model entailed a succession of 4 layers that repeats itself once. One trainable layer followed by three that are optimizers. They are discussed below, respectively.

3.4.1 Dense

Dense layers are the fundamental version of a neural network, represented in this simple math equation:

With the weights being the values trained by the matrix, and the bias being an option for avoiding overfitting. One we would not end up taking.

Thus, our only parameter here is the number of inputs, which we perhaps arbitrarily set to a nice, even 100.

3.4.2 Dropout

Our earliest Neural Network models had the frustrating issue of not improving is loss or accuracy between epochs – a poignant opportunity to recall the dangers of overfitting and local minimums.

Dropout layers are one way to avoid such pitfalls. They are set to have some probability (in our case, ) to set an input vector’s neuron to be zero. The effect is best represented in the following visual:



By doing so, the responsibility of various neurons is rescaled, giving an opportunity for the overconfident neurons to stop overshadowing other ones, or the ill performing get some room to improve.

3.4.3 Batch Normalization

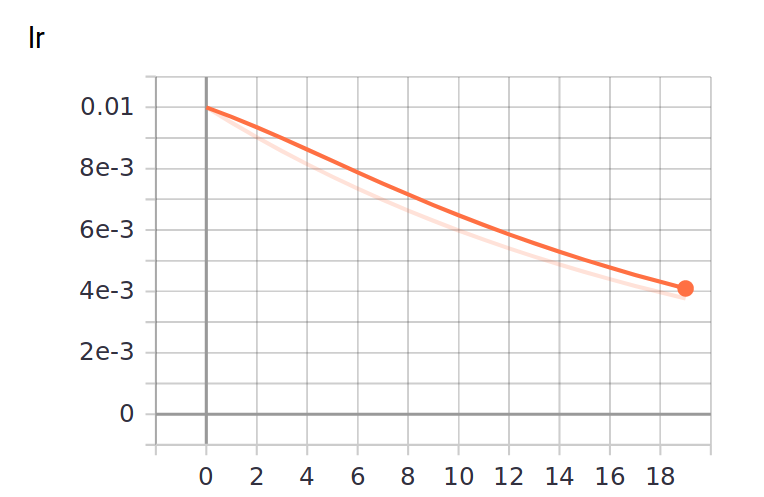
In order to combat wild deviations, we employ Batch Normalization. It takes the output of our dropout layer and subtracts the batch mean, then dividing it by the batch standard deviation. While dropout creates an immense value shift, this reigns it in, while having no effect on how often it occurs. The overall effect witnessed was path (read: number of epochs) to our peak accuracy than before.

3.4.4 Activation

Our first activation setup was to continue reusing Sigmoid until capping off with Softmax, which seemed to make enough sense given the reliability of Sigmoid and that Softmax would nicely round our prediction values up to a 1 or down to a 0. Au contrair, doing so makes confident predictions much less clear than simple guesses.

We introduced a lot of chaos to our data with Dropout and liked where Batch Normalization left it, so we replaced the original Sigmoid’s duty with ReLU, instead giving it the job of final activation. In that place, Sigmoid encourages greater change about the guessing region than ReLU, but not as drastic as Softmax. It would allow the model to fix errors, while not second-guessing itself forever.

In the following graph recording during testing, we achieved a steadily declining learning rate:



*Learning Rate over Epoch Number during training*

3.5 Arrangement

In summary of the above, the final layer setup was as follows:

1. Dense(100)
2. Dropout(0.4)
3. Batch Normalization
4. ReLU Activation
5. Dense(100)
6. Dropout(0.4)
7. Batch Normalization
8. ReLU Activation
9. Dense(1)
10. Sigmoid Activation

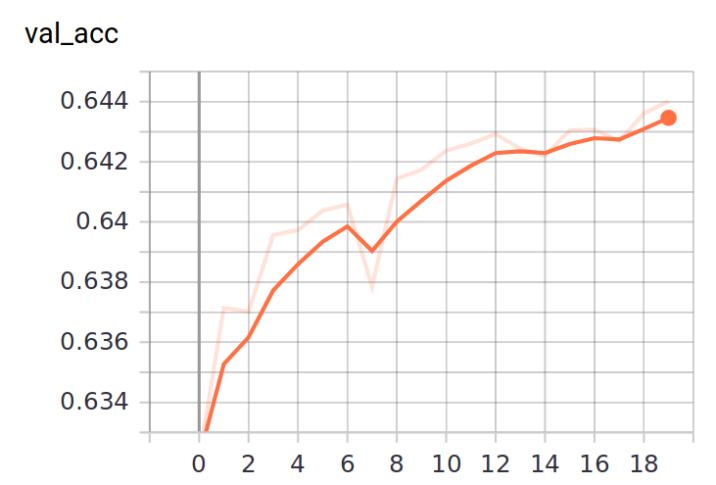
While our 4-layer combination alone did not show any improvement to our accuracy over all previously attempted models, the single repetition made all the difference. More layers are usually better, so that much seemed intuitive enough, but we struggle to explain why a second repeat failed to show any improvement whatsoever.

5 FITTING

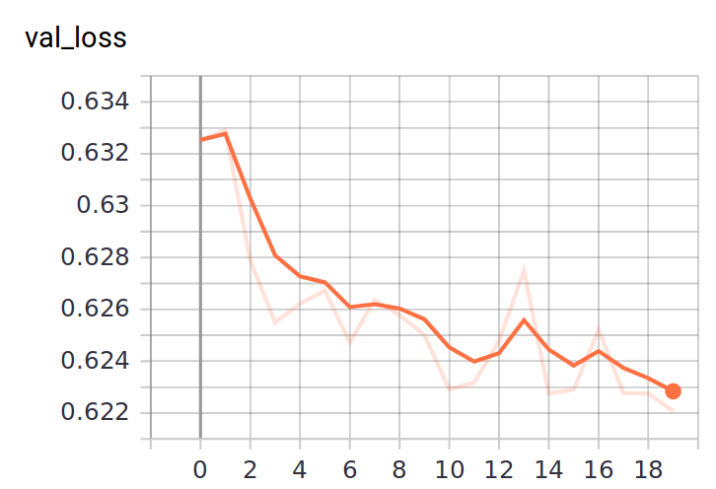
Our machines were unable to work on the full data set in timeframes that allowed us to incrementally improve upon it, so we trained our models random selections of 2 million entries. We original wrote this selection code for a Random Forest Model but did not find time to complete it.

6 TESTING

We tested our models we created using a batch size of 32, as 16 took too long, and set the epochs to continue until the loss and accuracy values failed to improve between one to the next. While our initial models would only run for a single epoch with a middling ~53% accuracy, we are happy to show that our final model clearly circumvents the issues that plagued the first drafts:



*Accuracy over Epoch Number during testing*



*Loss Value over Epoch Number during testing*

Each largely mirror one another as expected, and even as late as epoch 15, the model escapes a local minimum.

7 RESULTS

Our submission to Kaggle achieved the rank of 1730 with accuracy on the public data and private data is shown in the image below:

https://cdn.discordapp.com/attachments/448951591483998219/575905921700462592/SF_2019-05-08_114345_PM.png

Private Score 0.62064; Public Score: 0.67291

We are satisfied with our result. We achieved an accuracy only lower than the highest achieved in the contest. We fared no better against the private data set with its aforementioned full third of its contents being outliers.

SUMMARY

We met our goals of gaining a fundamental understanding of neural networks and producing a model that came close to the accuracy of the highest ranked submissions. Our experimentation with various layers and parameters yielded a sequential network with singly repeat sequence of Dense(100), Dropout(0.4), Batch Normalization, and ReLU Activation, with a Sigmoid activation on the end.

By using Statistical One-Hot Encoding, we limited the size of our binarized data to a manageable degree. It was very rewarding to feel we had overcome the daunting size of the data, which was the most frightening aspect of the project at the outset.

Given more time, we would have loved to continue searching for the best Neural Network, tried out LGBM, or even Random Forest. With benefit of hindsight, we know to rigorously record the results of every model we try in the future.

Finally, I am happy to announce that I hate Python *slightly* less at the end of this project than I did when the semester began.

REFERENCES

The competition:

<https://www.kaggle.com/c/microsoft-malware-prediction>

On the Data:

<https://www.kaggle.com/c/microsoft-malware-prediction/discussion/84745>

<https://www.kaggle.com/youhanlee/my-eda-i-want-to-see-all>

On Annealing:

<https://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/>

On Adam:

<https://arxiv.org/abs/1412.6980v8>

On Dropout:

<https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/>

On Batch Normalization:

<https://www.dlology.com/blog/one-simple-trick-to-train-keras-model-faster-with-batch-normalization/>

General Understanding of NN Models:

https://www.depends-on-the-definition.com/guide-to-multi-label-classification-with-neural-networks/

<https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi>

On Optimization:

<https://arxiv.org/abs/1412.6980v8>

<https://openreview.net/forum?id=ryQu7f-RZ>

On Encoding:

<https://www.kaggle.com/cdeotte/neural-network-malware-0-67>

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00