Deep Learning for Cybersecurity

Detecting botnets using ANNs

Barton Rhodes

Acknowledgements

This talk is based on a paper: “Predicting Domain Generation Algorithms with Long Short Term Memory Networks” by Woodbridge et al (arXiv:1611.00791)

Predicting Domain Generation Algorithms with Long Short-Term Memory Networks

Jonathan Woodbridge, Hyrum S. Anderson, Anjum Ahuja, and Daniel Grant

{jwoodbridge,hyrum,aahuja,dgrant}@endgame.com
Endgame, Inc.
Arlington, VA 22201

***Abstract*—Various families of malware use domain generation algorithms (DGAs) to generate a large number of pseudo-random domain names to connect to a command and control (C2) server. In order to block DGA C2 traffic, security organizations must first discover the algorithm by reverse engineering malware samples, then generate a list of domains for a given seed. The domains are then either preregistered, sink-holed or published in a DNS blacklist. This process is not only tedious, but can be readily circumvented by malware authors. An alternative**

server from which it can update, upload gathered intelligence, or pursue other malicious activities. The malicious actor only needs to register a small number of these domains to be successful. However, all the domains must be sinkholed, registered, or blacklisted before they go into use in order to preemptively defeat such an attack. This defense becomes increasingly difficult as the rate of dynamically generated domains increases.

Additional thanks to Miles Rufat-Latre, Mark Sliva, and Joewie Koh for working with me on the PyTorch implementation and to Jason Mancuso for being an awesome gent

Code

The code for the demo is available at:

https://github.com/bmorphism/woodbridge-lstm (you'll probably want a GPU!)

These slides:

https://github.com/bmorphism/talks/tree/master/2017-11-14-deep-learning-cybersecurity

You may also be interested in my previous KerasR talk for Denver R User Group:

https://github.com/bmorphism/talks/tree/master/2017-08-10-keras

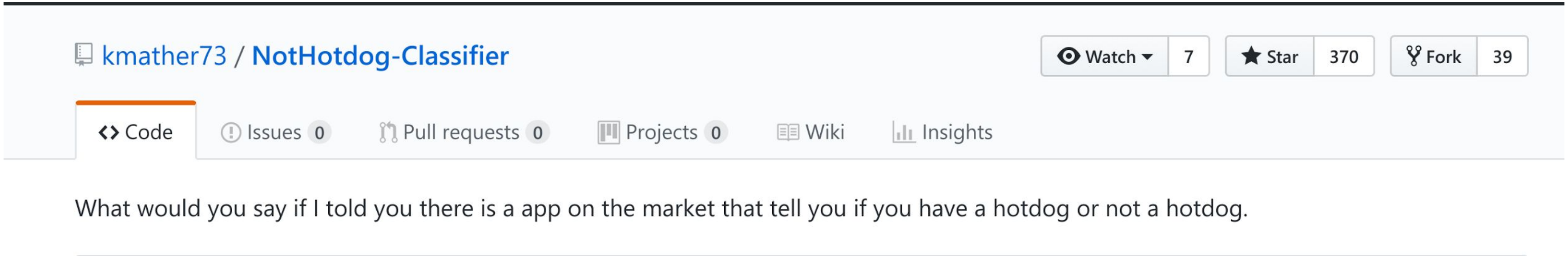
Problem

- Various families of malware use domain generation algorithms (DGAs) to generate a large number of pseudo-random domain names deterministically
- Register a command and control (C2) server on a small portion of the domain names
- Have botnet try all possible domain names until the C2 is found
- To block DGA C2 traffic, security organizations must first discover the algorithm by reverse engineering malware samples, then generate a list of domains for a given seed
- Tedious and largely ineffective using traditional methods

Solution - supervised learning!

Given examples of domains generated by algorithms and common non-botnet domains, learn how to identify one from another.

Hold-out test set to evaluate model performance.



Previous solutions

- rely on observing statistical properties, such as distributions of bigrams
- in the case of classical ML, like Random Forest, rely on hand-crafted features
- require large windows or don't generalize as well

Examples:

- Hidden Markov Models
- Jaccard's similarity on n-grams
- Logistic Regression on bigrams
- Random Forest classifier

Why Neural Networks are better

Universal Approximation Theorem:

In the mathematical theory of artificial neural networks, the **universal approximation theorem** states[1] that a feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions on compact subsets of $\mathbf{R}^n$, under mild assumptions on the activation function. (Source: Wikipedia)

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

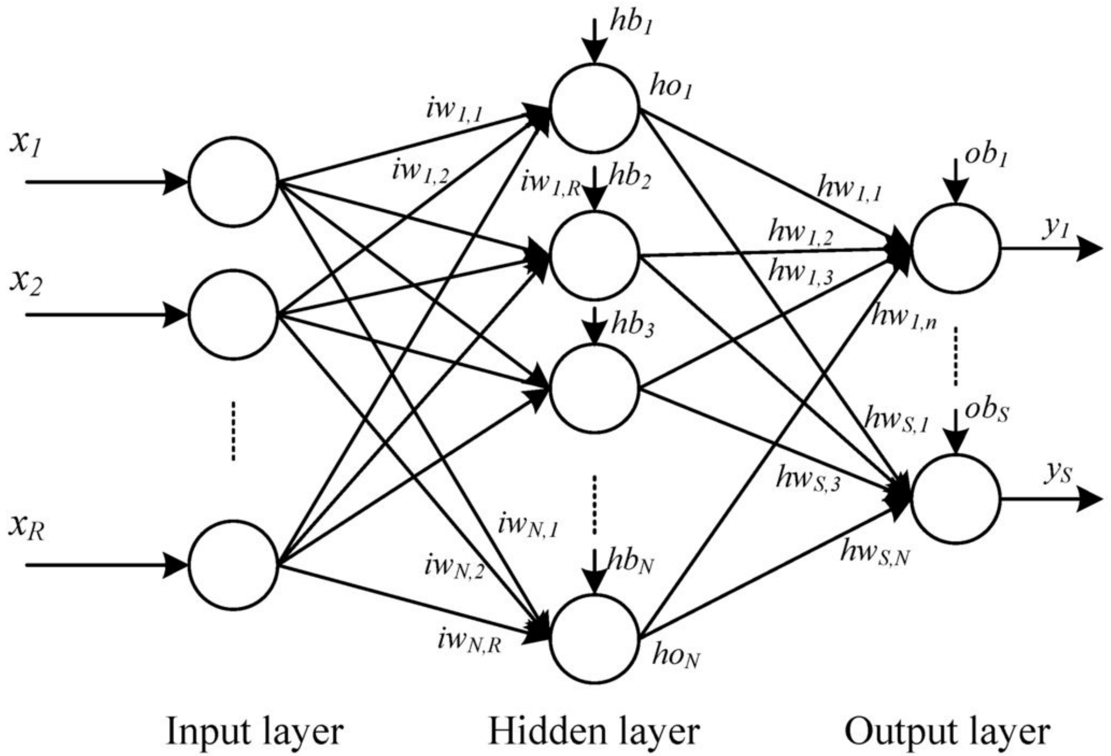
Feature engineering

In particular, the manually crafted features of the random forest DGA classifier include the following:

- length of domain name,
- entropy of character distribution in domain name,
- vowel to consonant ratio,
- Alexa 1M n-gram frequency distribution co-occurrence count, where $n = 3, 4$ or 5,
- *n-gram normality score*, and
- *meaningful characters ratio*.

What is a neural network?

http://deeplearning.ai
http://fast.ai



What is keras?

https://keras.io/

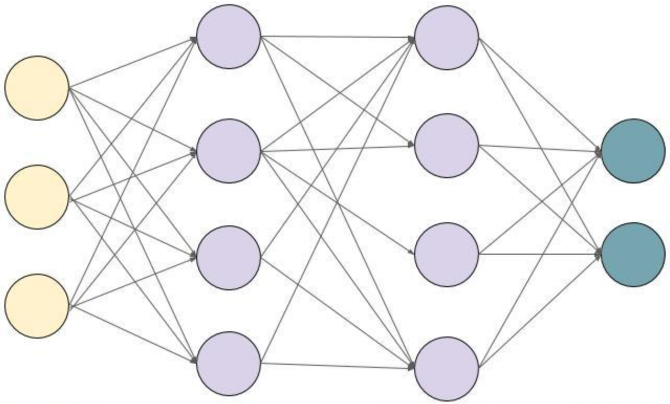
Core Layers

- Dense
- Activation
- Dropout
- Flatten
- Reshape
- Permute
- RepeatVector
- Lambda
- ActivityRegularization
- Masking

Recurrent Layers

- RNN
- SimpleRNN
- GRU
- LSTM
- ConvLSTM2D
- SimpleRNNCell
- GRUCell
- LSTMCell
- StackedRNNCells
- CuDNNGRU
- CuDNNLSTM

Simple example

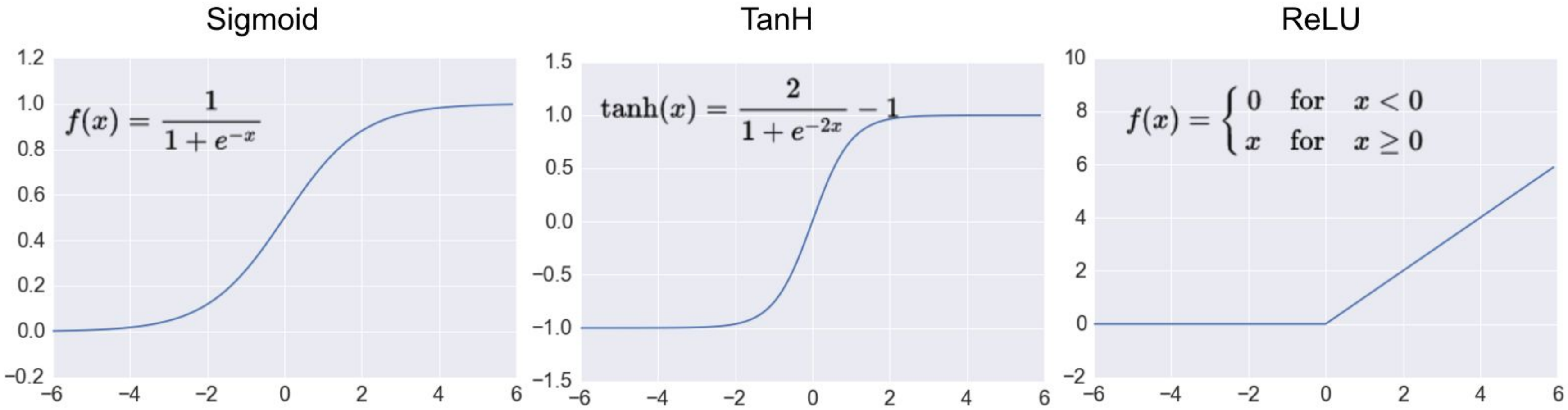



```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(512, activation='relu',
input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(nClasses,
activation='softmax'))
```

https://keras.io/#getting-started-30-seconds-to-keras

A word about activation functions



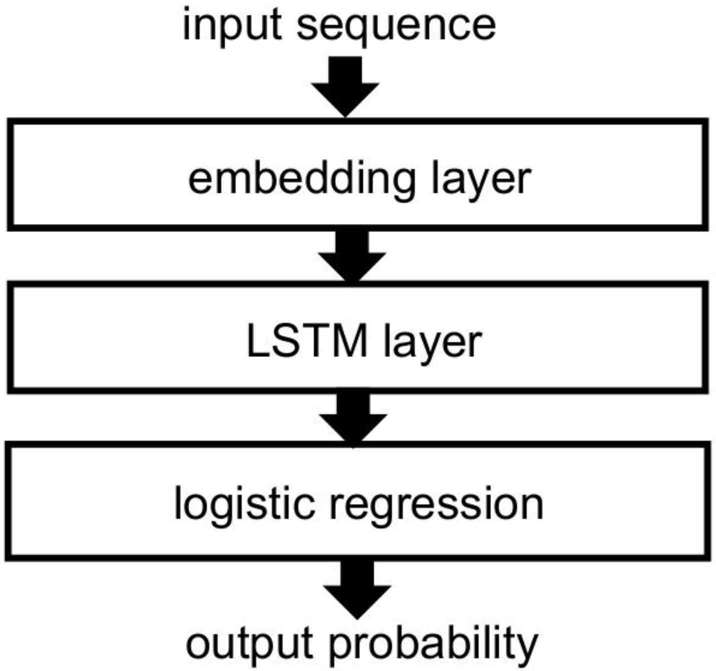
https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

Data

- Alexa Top 1M domains
- OSINT Domain-Generation Algorithms feed (as well as domain generation snippets published with the paper’s code)

For the DGA (malicious) domains, 750k examples across 30 botnet classes.

Architecture



Embedding

representation of the *semantics* of a word, efficiently encoding semantic information that might be relevant to the task at hand

$$q_{\text{mathematician}} = \left[\overbrace{2.3}^{\text{can run}}, \overbrace{9.4}^{\text{likes coffee}}, \overbrace{-5.5}^{\text{majored in Physics}}, \dots \right]$$

$$q_{\text{physicist}} = \left[\overbrace{2.5}^{\text{can run}}, \overbrace{9.1}^{\text{likes coffee}}, \overbrace{6.4}^{\text{majored in Physics}}, \dots \right]$$

http://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

Visualizing features is difficult

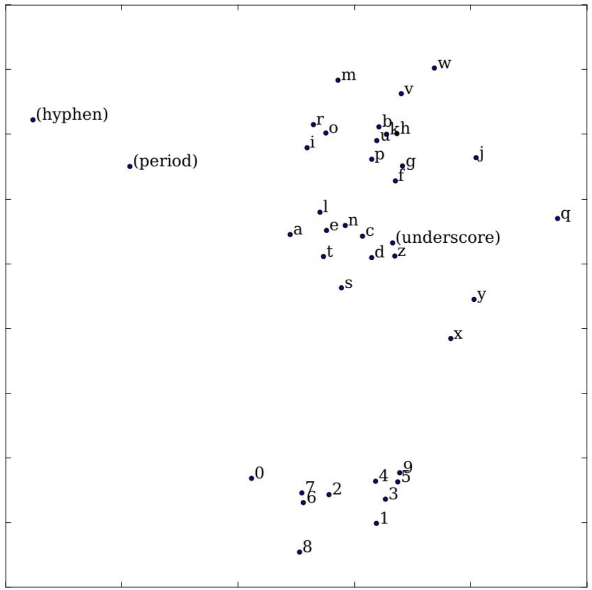
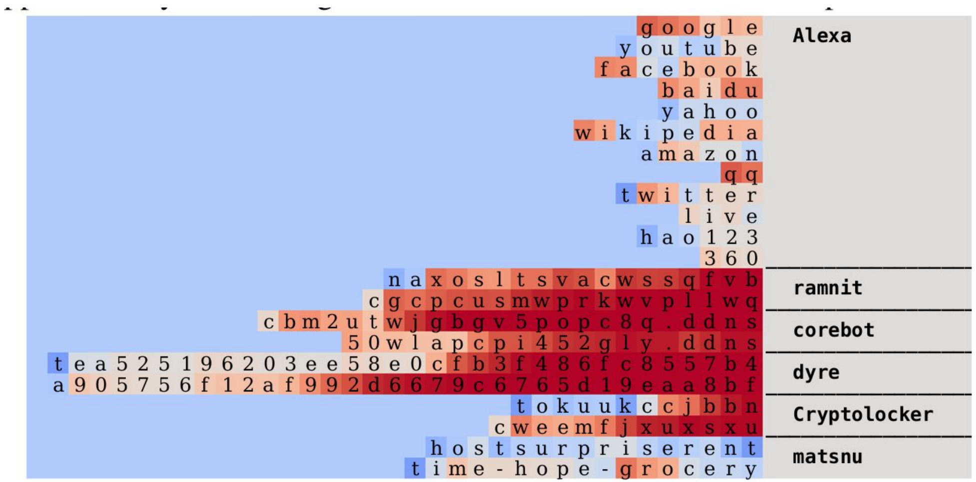
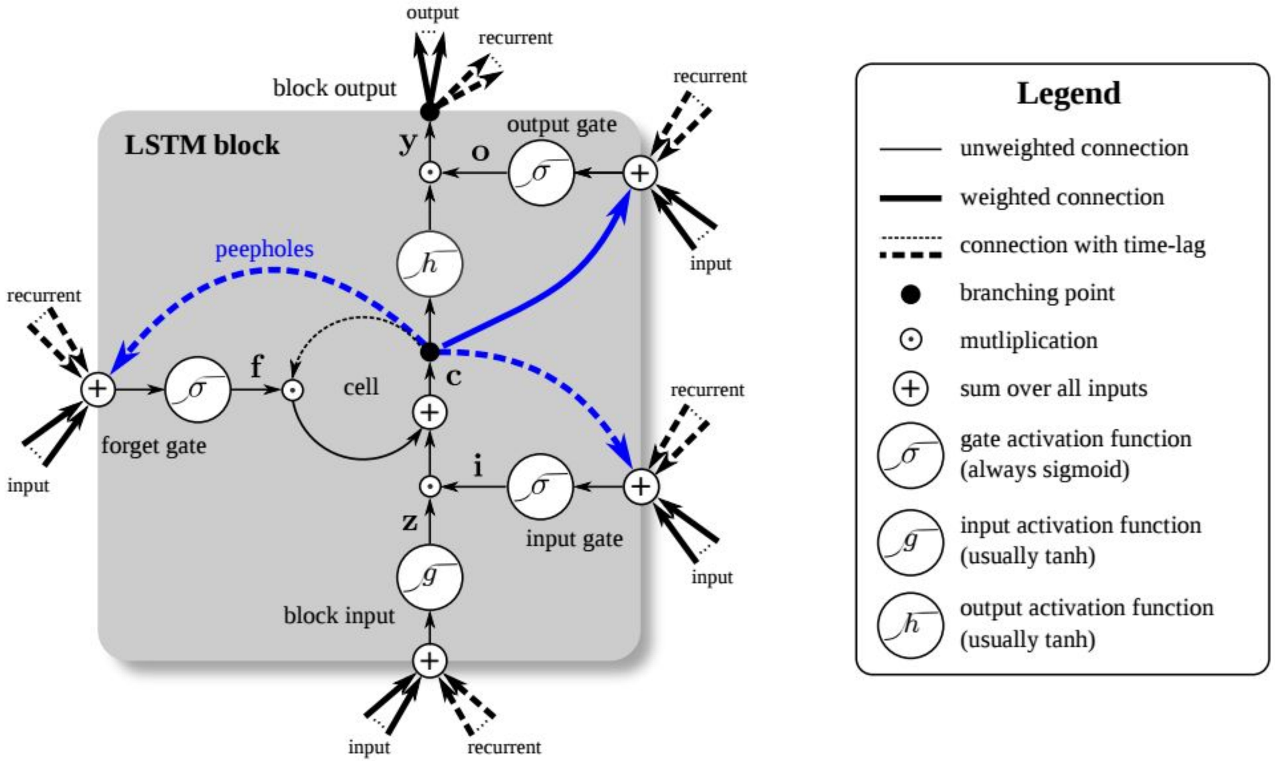


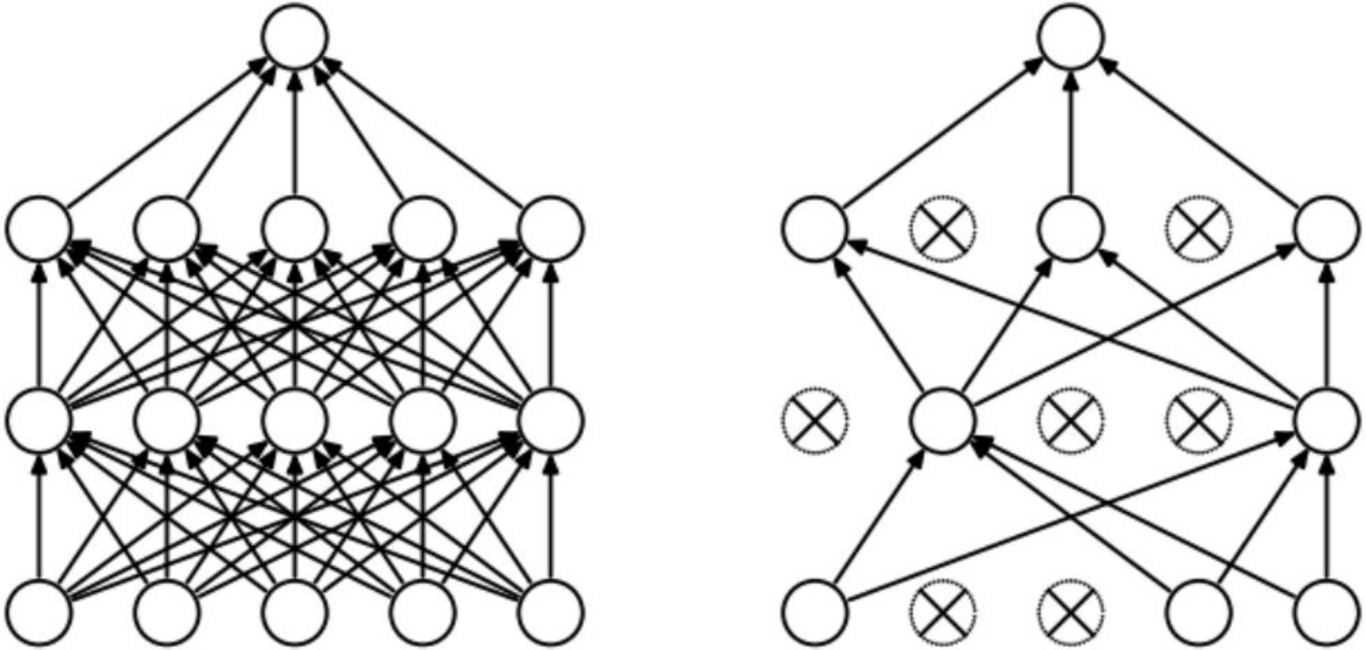
Fig. 8: Two-dimensional linear projection (PCA) of the embedded character vectors learned by the LSTM binary classifier. Note that the model groups characters by similar effect on the LSTM layer's states and the subsequent model loss.



LSTM


output
recurrent
block output
LSTM block
y
o
output gate
recurrent
input
peepholes
recurrent
f
cell
c
forget gate
input
i
z
input gate
recurrent
input
block input
input
recurrent
Legend
unweighted connection
weighted connection
connection with time-lag
branching point
mutliplication
sum over all inputs
gate activation function
(always sigmoid)
input activation function
(usually tanh)
output activation function
(usually tanh)

Dropout - prevent overfitting



(a) Standard Neural Net

(b) After applying dropout.

Keras

```
from keras.preprocessing import pad_sequences
from keras.models import Sequential
from keras.layers.core import Dense
from keras.layers.core import Dropout
from keras.layers.core import Activation
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM

model=Sequential()
model.add(Embedding(max_features,
                    128,
                    input_length=75))
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop')

# Pad sequence where sequences are case
# insensitive characters encoded to
# integers from 0 to number of valid
# characters
X_train=sequence.pad_sequences(X_train,
                               maxlen=75)

# Train where y_train is 0-1
model.fit(X_train, y_train,
          batch_size=batch_size, nb_epoch=1)
```

Fig. 2: Binary LSTM Code

```
from keras.preprocessing import pad_sequences
from keras.models import Sequential
from keras.layers.core import Dense
from keras.layers.core import Dropout
from keras.layers.core import Activation
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM

model=Sequential()
model.add(Embedding(max_features,
                    128,
                    input_length=75))
model.add(LSTM(128))
model.add(Dropout(0.5))
# nb_classes is the number of classes in
# the training set
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop')

# Pad sequence where sequences are case
# insensitive characters encoded to
# integers from 0 to number of valid
# characters
X_train=sequence.pad_sequences(X_train,
                               maxlen=75)

# Train where y_train is one-hot encoded for
# each class
model.fit(X_train, y_train,
          batch_size=batch_size, nb_epoch=1)
```

Fig. 3: Multiclass LSTM Code

Metrics

$$\text{Precision} = \frac{\sum \text{True Positive}}{\sum \text{True Positive} + \sum \text{False Positive}}$$

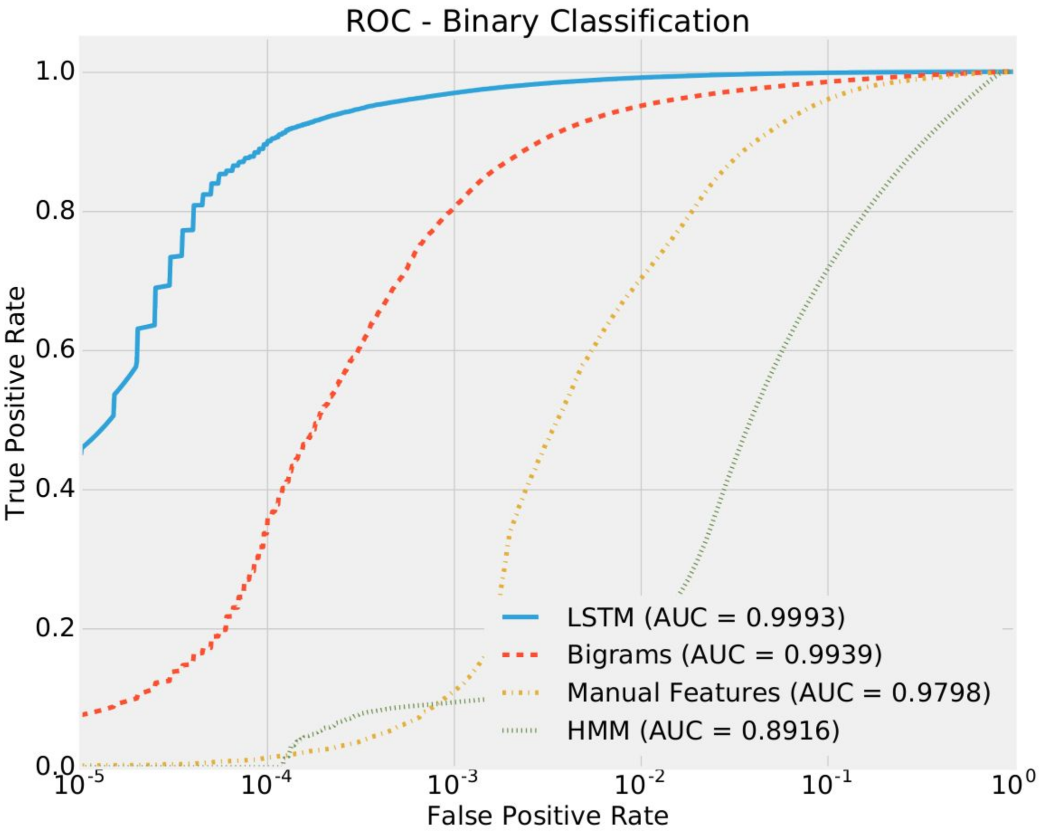
$$\text{Recall} = \frac{\sum \text{True Positive}}{\sum \text{True Positive} + \sum \text{False Negative}}$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{TPR} = \frac{\sum \text{True Positive}}{\sum \text{True Positive} + \sum \text{False Negative}}$$

$$\text{FPR} = \frac{\sum \text{False Positive}}{\sum \text{False Positive} + \sum \text{True Negative}}$$

Results



ROC - Binary Classification
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
LSTM (AUC = 0.9993)
Bigrams (AUC = 0.9939)
Manual Features (AUC = 0.9798)
HMM (AUC = 0.8916)

TABLE II: Precision, Recall and F_1 Score for Binary Classifiers

Domain Type	**Precision**				**Recall**				**F_1 Score**				**Support**
	HMM	Features	Bigram	LSTM	HMM	Features	Bigram	LSTM	HMM	Features	Bigram	LSTM	
`Alexa`	0.8300	0.9400	0.9700	0.9900	1.0000	1.0000	1.0000	1.0000	0.9100	0.9700	0.9900	0.9900	300064
`Cryptolocker`	1.0000	1.0000	1.0000	1.0000	0.9000	0.9800	0.9700	0.9900	0.9500	0.9900	0.9900	0.9900	1799
`P2P Gameover Zeus`	1.0000	1.0000	1.0000	1.0000	0.9900	1.0000	1.0000	1.0000	0.9900	1.0000	1.0000	1.0000	298
`Post Tovar GOZ`	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	19863
`Volatile Cedar / Explosive`	0.0000	1.0000	1.0000	1.0000	0.0000	0.4600	0.4900	0.9900	0.0000	0.6300	0.6600	1.0000	294
`banjori`	1.0000	1.0000	1.0000	1.0000	0.5900	0.9400	1.0000	1.0000	0.7400	0.9700	1.0000	1.0000	121678
`bedep`	1.0000	1.0000	1.0000	1.0000	0.8100	1.0000	1.0000	1.0000	0.8900	1.0000	1.0000	1.0000	53
`beebone`	0.0000	1.0000	1.0000	1.0000	0.0000	1.0000	0.9700	1.0000	0.0000	1.0000	0.9900	1.0000	65
`corebot`	1.0000	1.0000	1.0000	1.0000	0.5900	1.0000	1.0000	0.9600	0.7400	1.0000	1.0000	0.9800	81
`cryptowall`	1.0000	1.0000	1.0000	1.0000	0.1100	0.0600	0.1400	0.1200	0.1900	0.1100	0.2500	0.2100	29
`dircrypt`	1.0000	1.0000	1.0000	1.0000	0.9100	0.9200	0.9600	0.9600	0.9500	0.9600	0.9800	0.9800	150
`dyre`	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9900	1.0000	1.0000	1.0000	0.9900	1.0000	2389
`fobber`	1.0000	1.0000	1.0000	1.0000	0.8900	0.9600	0.9700	0.9700	0.9400	0.9800	0.9800	0.9900	181
`geodo`	1.0000	1.0000	1.0000	1.0000	0.9100	1.0000	0.9900	0.9900	0.9500	1.0000	1.0000	1.0000	173
`hesperbot`	1.0000	1.0000	1.0000	1.0000	0.8300	0.7700	0.8500	0.9700	0.9100	0.8700	0.9200	0.9800	58
`matsnu`	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	14
`murofet`	1.0000	1.0000	1.0000	1.0000	0.9200	1.0000	0.9900	1.0000	0.9600	1.0000	1.0000	1.0000	4292
`necurs`	1.0000	1.0000	1.0000	1.0000	0.8800	0.8400	0.9400	0.9600	0.9400	0.9100	0.9700	0.9800	1232
`nymaim`	1.0000	1.0000	1.0000	1.0000	0.8000	0.5600	0.7300	0.8000	0.8900	0.7200	0.8500	0.8900	1815
`pushdo`	1.0000	1.0000	1.0000	1.0000	0.6600	0.4700	0.5600	0.6000	0.7900	0.6400	0.7200	0.7500	507
`pykspa`	1.0000	1.0000	1.0000	1.0000	0.7200	0.5400	0.7700	0.9000	0.8400	0.7000	0.8700	0.9500	4250
`qakbot`	1.0000	1.0000	1.0000	1.0000	0.9100	0.9600	0.9600	0.9800	0.9500	0.9800	0.9800	0.9900	1517
`ramnit`	1.0000	1.0000	1.0000	1.0000	0.8800	0.9100	0.9400	0.9600	0.9400	0.9500	0.9700	0.9800	27439
`ranbyus`	1.0000	1.0000	1.0000	1.0000	0.9000	1.0000	0.9800	0.9800	0.9500	1.0000	0.9900	0.9900	2625
`shifu`	1.0000	1.0000	1.0000	1.0000	0.7200	0.2100	0.6600	0.7700	0.8400	0.3500	0.8000	0.8700	697
`shiotob/urlzone/bebloh`	1.0000	1.0000	1.0000	1.0000	0.9000	0.9700	0.9500	0.9800	0.9500	0.9900	0.9700	0.9900	3031
`simda`	1.0000	1.0000	1.0000	1.0000	0.5600	0.0800	0.4000	0.9200	0.7100	0.1400	0.5800	0.9600	4449
`suppobox`	1.0000	0.0000	1.0000	1.0000	0.0100	0.0000	0.0000	0.3200	0.0200	0.0000	0.0100	0.4800	298
`symmi`	0.0000	1.0000	1.0000	1.0000	0.0000	1.0000	0.7900	0.6900	0.0000	1.0000	0.8800	0.8200	18
`tempedreve`	1.0000	1.0000	1.0000	1.0000	0.7600	0.5700	0.8500	0.7700	0.8600	0.7300	0.9200	0.8700	74
`tinba`	1.0000	1.0000	1.0000	1.0000	0.8900	0.9800	0.9700	0.9900	0.9400	0.9900	0.9900	0.9900	18505
Micro Average	0.9008	0.9647	0.9826	**0.9942**	0.8815	0.9639	0.9848	**0.9937**	0.8739	0.9593	0.9851	**0.9906**	16708
Macro Average	0.8655	0.9335	0.9668	**0.9674**	0.6787	0.7477	0.8006	**0.8571**	0.7335	0.7929	0.8468	**0.8913**	16708

TABLE IV: Precision, Recall and F_1 Score for Multiclass Classifiers

	Precision			**Recall**			**F_1 Score**			
Domain Type	Features	Bigram	LSTM	Features	Bigram	LSTM	Features	Bigram	LSTM	Support
`Alexa`	0.914	0.980	**0.990**	0.960	0.990	**1.000**	0.940	0.988	**0.990**	199978
`Cryptolocker`	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1189
`P2P Gameover Zeus`	0.000	**0.343**	0.327	0.000	**0.288**	0.217	0.000	**0.308**	0.247	196
`Post Tovar GOZ`	0.941	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.970	**1.000**	**1.000**	13185
`Volatile Cedar / Explosive`	0.000	**1.000**	0.987	0.000	**1.000**	0.980	0.000	**1.000**	0.980	200
`banjori`	0.900	0.990	**1.000**	0.938	**1.000**	**1.000**	0.920	**1.000**	**1.000**	81281
`bedep`	0.000	0.000	**0.943**	0.000	0.000	**0.107**	0.000	0.000	**0.187**	34
`beebone`	**1.000**	**1.000**	**1.000**	0.560	**1.000**	**1.000**	0.713	**1.000**	**1.000**	42
`corebot`	0.000	**1.000**	**1.000**	0.000	0.980	**0.990**	0.000	0.990	**0.993**	54
`cryptowall`	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	15
`dircrypt`	0.000	**0.083**	0.000	0.000	**0.010**	0.000	0.000	**0.020**	0.000	100
`dyre`	0.985	0.988	**1.000**	**1.000**	0.988	**1.000**	0.991	0.988	**1.000**	1600
`fobber`	0.000	0.000	**0.177**	0.000	0.000	**0.023**	0.000	0.000	**0.040**	121
`geodo`	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	114
`hesperbot`	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	36
`matsnu`	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	9
`murofet`	**0.883**	0.643	0.783	0.066	0.542	**0.700**	0.122	0.590	**0.737**	2845
`necurs`	0.000	0.000	**0.643**	0.000	0.000	**0.093**	0.000	0.000	**0.160**	827
`nymaim`	0.000	0.390	**0.477**	0.000	0.113	**0.190**	0.000	0.175	**0.267**	1222
`pushdo`	0.000	0.770	**0.853**	0.000	0.588	**0.640**	0.000	0.665	**0.730**	339
`pykspa`	0.000	0.788	**0.910**	0.000	0.593	**0.713**	0.000	0.675	**0.800**	2827
`qakbot`	0.000	**0.590**	0.590	0.000	0.232	**0.387**	0.000	0.338	**0.463**	993
`ramnit`	0.566	0.637	**0.770**	0.654	0.763	**0.850**	0.605	0.690	**0.810**	18308
`ranbyus`	0.439	0.000	**0.450**	0.000	0.000	**0.517**	0.001	0.000	**0.460**	1736
`shifu`	0.000	0.037	**0.560**	0.000	0.003	**0.570**	0.000	0.007	**0.553**	465
`shiotob/urlzone/bebloh`	0.000	0.965	**0.973**	0.000	0.853	**0.907**	0.000	0.907	**0.940**	2016
`simda`	0.000	0.840	**0.930**	0.000	0.750	**0.977**	0.000	0.792	**0.950**	2955
`suppobox`	0.000	0.392	**0.833**	0.000	0.062	**0.517**	0.000	0.112	**0.627**	197
`symmi`	0.000	0.625	**0.913**	0.000	0.117	**0.857**	0.000	0.200	**0.883**	11
`tempedreve`	0.000	**0.043**	0.000	0.000	**0.010**	0.000	0.000	**0.018**	0.000	50
`tinba`	0.821	0.735	**0.910**	0.923	0.802	**0.990**	0.869	0.767	**0.950**	12332
Micro Average	0.851	0.933	**0.963**	0.888	0.944	**0.970**	0.867	0.940	**0.963**	11138
Macro Average	0.240	0.479	**0.614**	0.197	0.409	**0.523**	0.198	0.427	**0.541**	11138

Demo

Putting it all together in Keras

Conclusions

- by automating feature extraction and working with better representations in neural networks, one can accomplish state-of-the-art results on long-standing cybersecurity tasks
- Keras is a great library to quickly and easily reproduce a lot of papers (except ones you'll need PyTorch for https://medium.com/intuitionmachine/pytorch-dynamic-computational-graphs-and-modular-deep-learning-7e7f89f18d1)
- can be productionised to neutralize botnets, including new kinds, before they spread

Denver DL Study Group

Small meeting @ 2 pm at Denver Bicycle Café every Sunday.

Currently working through:

- Andrew Ng's Deep Learning Specialization on Coursera
- fast.ai Deep Learning for Coders (Part II)
- Passenger screening algorithm Kaggle competition

Contact me at @bmorphism on Twitter or over email at b@bmorphism.us to be added to our Slack.

Thanks!