End-to-End Radio Traffic Sequence Recognition with Recurrent Neural Networks

Timothy J. O'Shea
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Arlington, VA
Email: oshea@vt.edu

Seth Hitefield
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Arlington, VA
Email: hitefield@vt.edu

Johnathan Corgan
Corgan Labs,
San Jose, CA
Email: johnathan@corganlabs.com

***Abstract*—We investigate sequence machine learning techniques on raw radio signal time-series data. By applying deep recurrent neural networks we learn to discriminate between several application layer traffic types on top of a constant envelope modulation without using an expert demodulation algorithm. We show that complex protocol sequences can be learned and used for both classification and generation tasks using this approach.**

I. Introduction

Traffic analysis and deep packet inspection are important tools in ensuring quality of service (QoS), network security, and proper billing and routing within wired and wireless networks. Systems and algorithms exist today to discern between different protocols and applications for these reason, but new methods provide great potential for improvement.

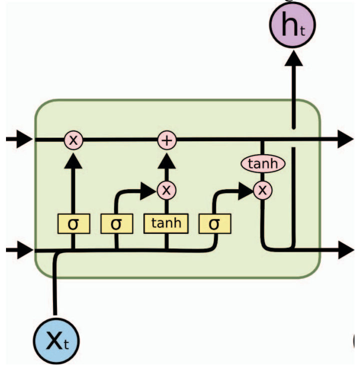
Current day techniques for this sort of traffic inspection typically involve explicit parsers written only for supported protocols and very prone to error. Any implementation of an additional wireless physical layer (PHY) requires explicit implementation of specific signal processing routines for that PHY, each of these adds complexity, implementation cost, potential vulnerabilities and narrow specificity to such a capability.

By applying machine learning to this task of identifying high level traffic protocols from a learning angle, we demonstrate that we can potentially shortcut many of these expert algorithm implementations and learn a data driven classifier which learns features able to extract high level protocol information from low layer PHY representation. In doing so we build a model which generalizes well and is less prone to security vulnerabilities, extending our prior work on determining lower level PHY classification we demonstrated in prior work [13].

A. Recurrent Networks in Natural Language

Recurrent neural network approaches to temporal sequence learning are not new but have seen a great resurgence in recent years in natural language learning and other sequence based domains due to improved methods and computational power. In many of these applications sequences of tokens such as characters [9], words or phonemes are encoded using recurrent neural networks especcially based on the the long short-term memory [3] (LSTM) or the Gated Recurrent Unit (GRU) [5]. The basic LSTM neuron unit and its transfer functions are shown in figure 1.

Fig. 1. Basic LSTM Unit Transfer Function Diagram from [10]



Many applications have employed recurrent networks for translation between sequences (such as languages) [7] based on embeddings, mapping from sequences to discrete classes [4], and many other tasks. The LSTM has been especially successful in many RNN models for sequencing voice and text. In many cases it has replaced Hidden Markov models (HMMs) as state of the art for these tasks.

B. Background on Radio Sequence Motivations

In radio communications, the radio transmitter and receiver are comprised of a hierarchy of sequence translation routines [1]. These translate between sequences of protocol data bits, forward error corrected encoded bits, randomized and whitened bits, framed bits, and modulated and encoded symbols to traverse the radio channel.

Rather than implementing expert algorithms for each of these, we can attempt to learn these sequence translations by presenting data to a machine learning algorithm.

II. Supervised Traffic Type Learning

In our network we train multiple LSTM and fully-connected layers on a succession of slices of raw modulated radio I/Q data to perform supervised classification into one of 11 different protocol traffic classes.

A. Dataset Generation

We generate a hybrid real-synthetic dataset by recording real network application protocols from traffic traversing our test virtual machine using wireshark while considering each of the network traffic behaviors below.

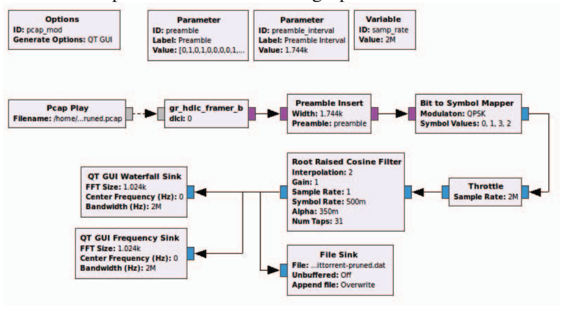
- **Streaming**
  - Video Streamin (via ABC video)
  - Video Streaming (via Youtube)
  - Music Streaming (via Spotify)
- **Utilities**
  - Apt-get
  - ICMP Response Test (Ping)
  - Version Control (git)
  - Internet Relay Chat (IRC)
- **Downloading/Browsing**
  - Bit-Torrent
  - Web browsing
  - File transfar protocol (FTP)
  - HTTP Download

Wireshark and *tcpdump* were used to capture network traffic and generate traces of each network protocol used later in for training and classification. While these utilities can filter specific network traffic/protocols, we consider all of the traffic traversing to the host including domain name look ups and other unrelated idle traffic, as would be realistic in most scenarios.

Packetized network traffic is then replayed onto a continuously modulation synthetic software modem and recorded for the I/Q sample data in the dataset. The transmitter uses GNU Radio [2] flow-graph with High-Level Data Link Control (HDLC) framing and a Quadrature Phase Shift Keying (QPSK) modulation. We omit error correction or randomization for now, as shown in figure II-A.

When no packet is present to transmit, HDLC emits a cosntant idle flag of 0x7E, and we additionally introduce a constant preamble every 1744 bits to allow for PHY synchronization by a receiver.

Fig. 2. Packet Capture Transmitter Flowgraph in GNU Radio


Options
ID: pcap_mod
Generate Options: QT GUI

Parameter
ID: preamble
Label: Preamble
Value: [0,1,0,1,0,0,0,0,1,...

Parameter
ID: preamble_interval
Label: Preamble Interval
Value: 1.744k

Variable
ID: samp_rate
Value: 2M

Pcap Play
Filename: /home/...runed.pcap

gr_hdlc_framer_b
dlci: 0

Preamble Insert
Width: 1.744k
Preamble: preamble

Bit to Symbol Mapper
Modulaton: QPSK
Symbol Values: 0, 1, 3, 2

Root Raised Cosine Filter
Interpolation: 2
Gain: 1
Sample Rate: 1
Symbol Rate: 500m
Alpha: 350m
Num Taps: 31

Throttle
Sample Rate: 2M

QT GUI Waterfall Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 2M

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 2M

File Sink
File: ...ittorrent-pruned.dat
Unbuffered: Off
Append file: Overwrite

B. Model Data Ingest

When training a RNN model, we must decide how to aggregate the time-series into states. Among others, two considerations here are how to slice a sequence into time steps and how to partition the data into regions of training and test data.

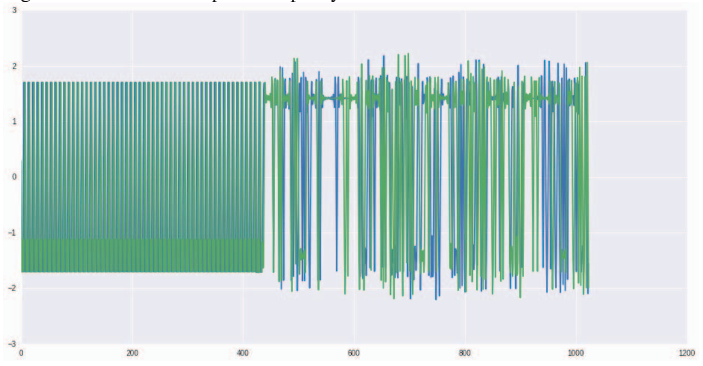
In the first case, consider a time series $x(n)$ where we wish to create examples from linear subsequences. In this case, we extract N windows of size L at a stride of M to form a three dimensional example vector. In this case, the dimensions are expressed in the form of a real-valued tensor of shape Nx2xL, where the first dimensions is over window, the second is over the I/Q dimension, and the third is over time within each window. Each tensor example is then formed from $L + (N-1)*M$ complex samples in the original time-series. Since an optimal slicing is not known offhand for either task, we will use this notation throughout to refer to our input tensor shape tested during training. We perform this slicing using python-numpy and ingest tensor data into Keras [8] and Theano [16] for model training.

For our supervised network-task classification model, we use one-hot target labels for each example where 1xK output values are all zero except where the target index k is of the example class, where it is set to 1.0. This is commonly used along with a SoftMAX output activation layer to help in training for class prediction, and we use it the same way here.

In the case of a generative regression model, we use the same N time-step windows as out input tensor data, while using an N+1'th time-step of real-valued samples as our example target.

Lastly, as a pre-processing step, we consider whether to input I/Q samples, R/θ samples, R-only, or θ only from our sample representation, where R,θ represent the polar form of the I/Q sample. We do this to consider capturing the circular relationship between in-phase and quadrature components which is thrown away when treating them as real valued separate channels.

Fig. 3. 1024 time samples of Spotify class



C. Discriminative Model Training

In our discriminative classifier we train a network to decide which traffic is present as a K-class supervised learning problem. We implement both a CLDNN [11], a network formed by a sequence of convolutional layers, LSTM layers, and fully-connected layers, as well as a LSTM and fully connected layers. The architecture for the latter is shown in 4.

Since few benchmark data sets exist in this domain, we publish our data sets on radioml.com and describe out approach for comparison.

1) Noiseless Training with Overlap: We begin with the simplest case to ensure the learning with this model is possible. We use the raw modulated signal, at very high signal to noise ratio (SNR), with no effects of frequency timing offset. Our examples are each 128-symbol to simplify the task complexity. We do not re-use examples between training and test sets, but we do allow some edge overlap between training and test sets.

We select an input tensor shape of Nx2x128 where we search over a range of N values to optimize time steps for performance, shown in figure 5

Fig. 4. LSTM256 Recurrent Network Structure

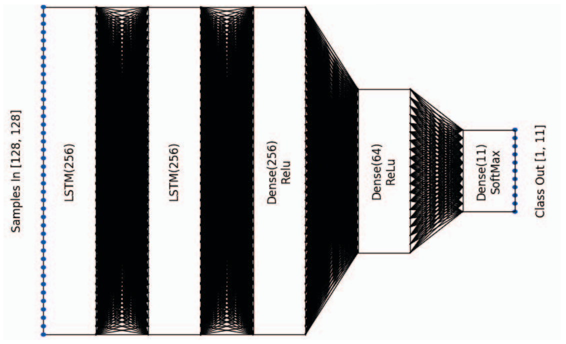


Fig. 5. Performance of classifier vs RNN sequence length

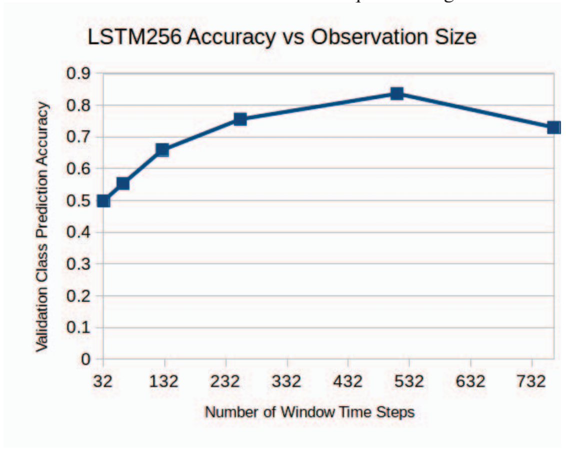
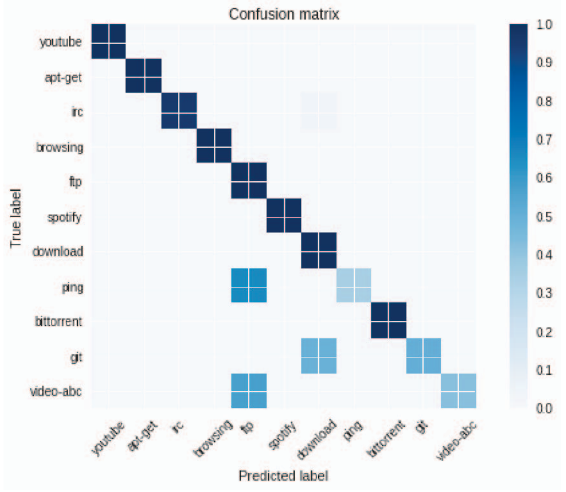


Fig. 6. Best LSTM256 confusion with RNN length of 512 time-steps

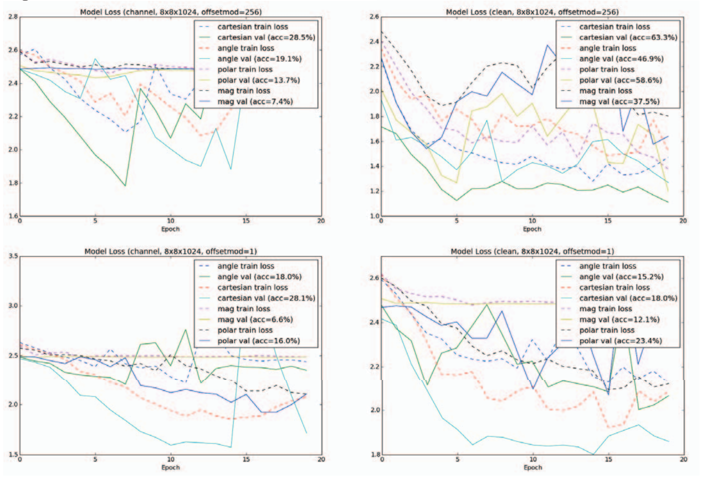


We find our best accuracy performance to be obtained when using 512 time-steps of 2x128 samples into the LSTM. Details of different sequence length evaluation are detailed in table I. For a sequence length of 512, we obtain a confusion matrix of our best performance classification accuracy in figure 6, with an overall accuracy of 84%. It is important to note that some error is inherent in the data set especially for low rate application protocols which may not have defining packets available in any given small time window.

2) Training with Channels and no Overlap: To fully differentiate training and test sets, we need to fully remove overlap between examples drawn from each. In this section, we partition the original time series into hard partitions of 250,000 samples, each assigned to either training or test, and then draw examples from within these bounds for training and test. This ensure that we are learning generalizable sequence features rather than specific window examples which may be used to recall one class. We consider two forms of the input signals, one, ”clean”, which represents the high-SNR signal without frequency or timing offsets, and one, ”channel”, which applies the channel effects of Gaussian noise, random frequency offset, and random timing offset. The latter has a signal to noise ratio around 20dB. Lastly, we relax the effect of 128-symbol aligned example offsets. We consider two values for ”offset_modulo”: 1, where we may be begin on any offset, and 256, where we begin 128-symbol (256 -sample) aligned to consider the effect of random alignment on classification.

Since numerous architectures exist to evaluate on this task, and searching over them is a laborious and compute-time intensive task, we introduce a tool, still in very early form called dist_hyperas [18], to help in searching for optimal architecture hyper-parameters over a number of distributed GPU instances.

Fig. 7. Trade Search 1: Data Dimensions



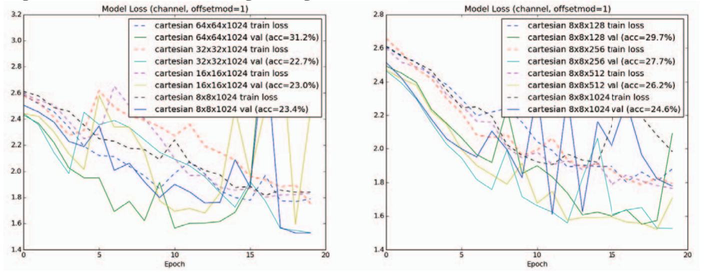
In our first trade, we evaluate the performance of input representation, channel effects and stride on our model using an 8x8x1024 input tensor shape. The loss curves and final accuracy for each model tested is shown in figure 7. In this case, we obtain our best performance with a channel using the Cartesian I/Q input representation, and the offset_modulo doesn’t seem to have a huge impact when a real channel is considered. (It has a much larger impact on performance with the clean signal). Best performance with a channel is around

TABLE I. PERFORMANCE MEASUREMENTS ON VARYING SEQUENCE LENGTHS

Sequence Length	Val. Loss	Val. Accuracy	$N_{samples}$	$N_{symbols}$	N_{bits}	Sec/Epoch
32	1.2126	0.498805	1120	140	280	5
64	1.0386	0.553546	2144	268	536	18
128	0.7179	0.65894	4192	524	1048	17
256	0.4586	0.75621	8288	1036	2072	29
512	0.2711	0.836535	16480	2060	4120	38
768	0.5328	0.730413	24672	3084	6168	27

28.5% while without a channel it is around 63.3%.

Fig. 8. Trade Search 2: Input Representation

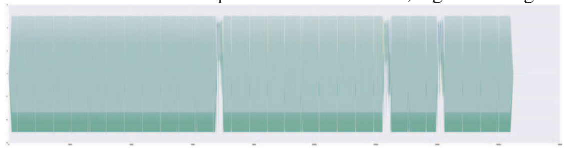


In our second trade, we consider only Cartesian I/Q inputs and an offset_mod of 1. In this case we trade the sequence length (number of time-steps) against the size and stride of the window used. The results are shown in figure 8. In this case, we seem to obtain out best performance with a window size of L=64 and a sequence length of N=1024 giving a classification accuracy of around 31.2% with realistic channel and sampling conditions. We are still investigating larger models and additional hyper-parameter combinations but large LSTM architectures require large memory footprints currently, near/at the limitations of our Titan X, and training takes significant compute-time. In the future we hope to find smarter ways to live within these limitations.

We believe some additional performance could be gained from architecture searches, but also from improved fundamental techniques described below to help cope with channel variation.

D. Generative Model Training

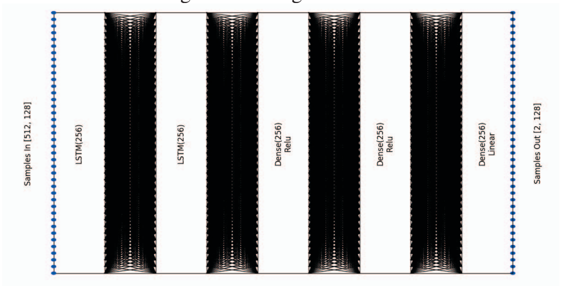
Fig. 9. Prediction of IRC sequence: Left half seed, right half regression



We employ a first order generative model shown in figure 10 to predict the next time-step window given N previous time-step windows as a regression task and train using mean squared error (MSE) loss.

In figure 9 we show a generative test where the first half is ground truth from an IRC sequence example and the second half of samples is predicted from our recurrent generative model.

Fig. 10. Best LSTM256 generative regression network



Visually comparing the predicted samples to those from the baseline example, we can see it correctly predicts both HDLC idle pattern and equal-width framing patterns as well as some semblance of data bursts occurring asynchronously. This is encouraging given it is a completely naive model with no expert knowledge of the modulation, preamble structure, the HDLC protocol, or the application on top.

In future work we would like to investigate the use of generative adversarial models [6] to greatly improve realism. Two other extremely promising recent approaches to time-series generation with many techniques to consider are also presented in [17] and [12].

III. CONCLUSION

We have shown in this work that recurrent neural network model can learn features on low level radio data to perform high level protocol recognition for both discriminative and generative tasks.

We have demonstrated baseline performance for both tasks which works well under ideal conditions, but have also showed that introducing realistic channel effects makes the task significantly more difficult reducing performance. We hope that in future work attention models to help reverse channel effects [15] and the channel regularization during training [14] can help improve performance.

These results have significant impact into showing what is possible in radio sequence and protocol recognition. By providing a robust learned method for protocol identification learning which is data and experience driven, future radio tasks such as spectrum allocation, QoS, scheduling and decision making can make intelligent decisions about how to prioritize and allocate resources within a larger multi-user cognitive radio system without relying on highly specialized expert metrics or parsers.

Acknowledgment

This research was supported in part by funding from the Defense Advanced Research Projects Agency's (DARPA) MTO Office under grant HR0011-16-1-0002. The views, opinions, and/or findings expressed solely are those of the authors.

References

[1] T. S. Rappaport *et al.*, *Wireless communications: Principles and practice*. Prentice Hall PTR New Jersey, 1996, vol. 2.

[2] E. Blossom, "Gnu radio: Tools for exploring the radio frequency spectrum," *Linux journal*, vol. 2004, no. 122, p. 4, 2004.

[3] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[4] J. Bayer, C. Osendorfer, and P. Van Der Smagt, "Learning sequence neighbourhood metrics," in *International Conference on Artificial Neural Networks*, Springer, 2012, pp. 531–538.

[5] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *ArXiv preprint arXiv:1412.3555*, 2014.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[7] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[8] F. Chollet, *Keras*, https://github.com/fchollet/keras, 2015.

[9] A. Karpathy, *Char-rnn: Multi-layer recurrent neural networks (lstm, gru, rnn) for character-level language models in torch*, 2015.

[10] C. Olah, "Understanding lstm networks," *Net: Http://colah. github. io/posts/2015-08-Understanding-LSTMs*, 2015.

[11] T. N. Sainath *et al.*, "Learning the speech front-end with raw waveform cldnns," in *Proc. Interspeech*, 2015.

[12] S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, *et al.*, "Wavenet: A generative model for raw audio," *ArXiv preprint arXiv:1609.03499*, 2016.

[13] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," *ArXiv preprint arXiv:1602.04105*, 2016.

[14] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," *ArXiv preprint arXiv:1608.06409*, 2016.

[15] T. J. O'Shea, L. Pemula, D. Batra, and T. C. Clancy, "Radio transformer networks: Attention models for learning to synchronize in wireless systems," *ArXiv preprint arXiv:1605.00716*, 2016.

[16] Theano Development Team, "Theano: a Python framework for fast computation of mathematical expressions," *ArXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688.

[17] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," *ArXiv preprint arXiv:1609.05473*, 2016.

[18] *Github dist_hyperas project*, https://github.com/osh/dist_hyperas, 2016.