Detecting Network Anomalies Using Machine Learning

Matthew Grubelic, Anthony Saldana, Luke Turbert, Chris Saliby, Andrew Rittenhouse

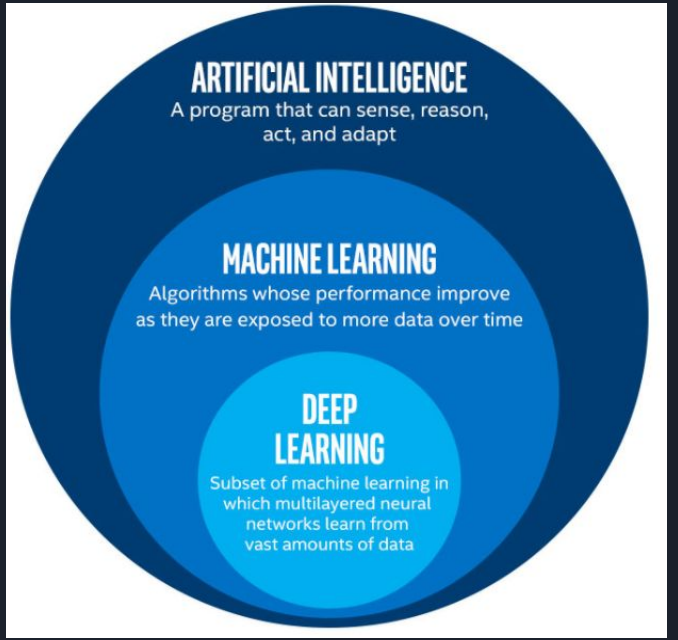
Brief Background

- The nature of cyber attacks are evolving and adapting in different ways
- Current Intrusion Detection Systems (IDSs) are beginning to show their age
  - IDSs such as “Snort” cannot account for certain attacks, such as “zero-day” attacks
- New cyber-security systems to identify and prevent intrusion attempts are being explored
  - Namely, with machine learning

What is Machine Learning

- Training a machine so that it can make predictions/assessments without being specifically programmed to do so.
  - Subset of Artificial Intelligence (AI)

- Two subsets of Machine Learning:
  - Supervised Learning:
    - Given (x,y) predict y
    - Ex: Sequence-to-sequence, SVM, etc.
  - Unsupervised Learning:
    - Given (x) find y
    - Ex: K-Means, DBSCAN, clustering, etc.



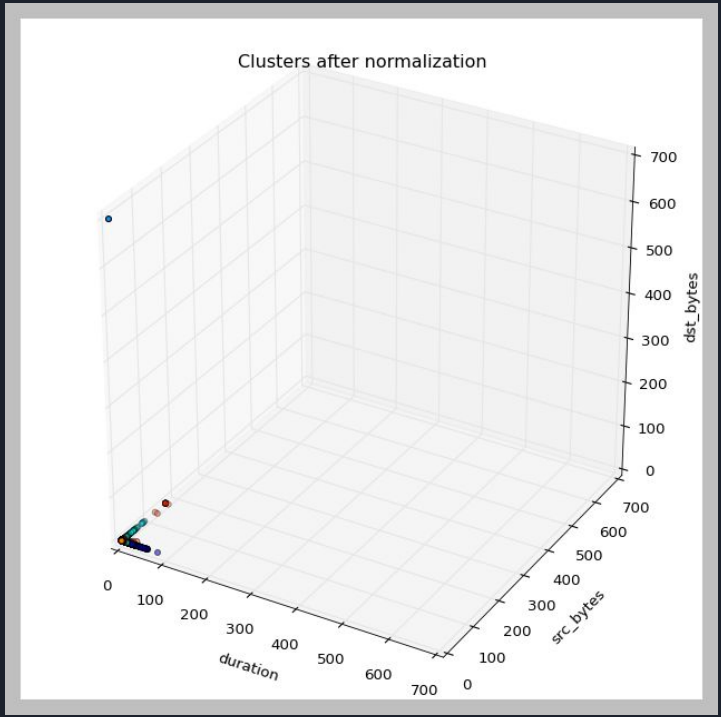
https://towardsdatascience.com/cousins-of-artificial-intelligence-dda4edc27b55

Our previous implementations

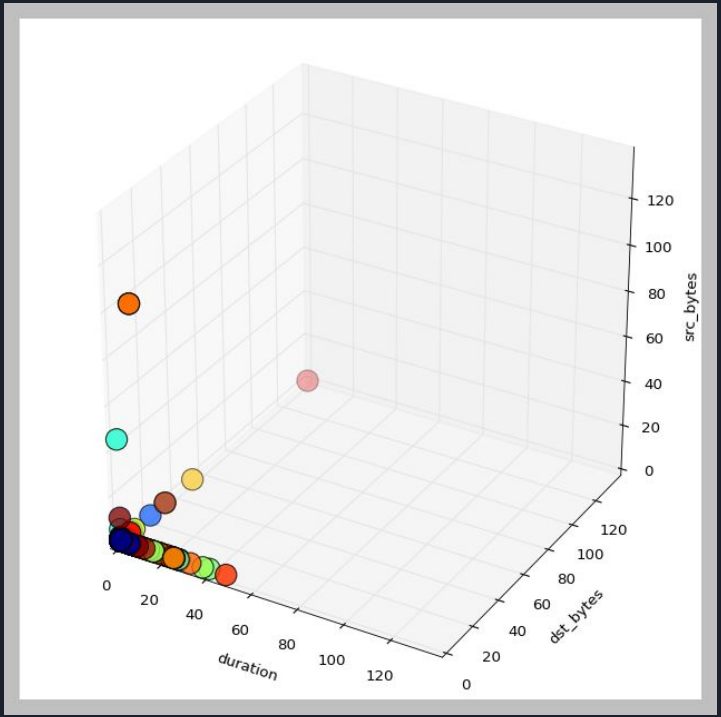
In the first semester, we implemented a K-Means and DBSCAN clustering algorithm on the KDD Cup 99 dataset

- The dataset: contains millions of labeled data entries with both normal and attack data
- The models:
  - K-Means : clustering algorithm that identifies outliers in the data
  - DBSCAN: density-based clustering algorithm, where outliers are found in low-density regions

K-Means results



DBSCAN results



After the Fall Semester

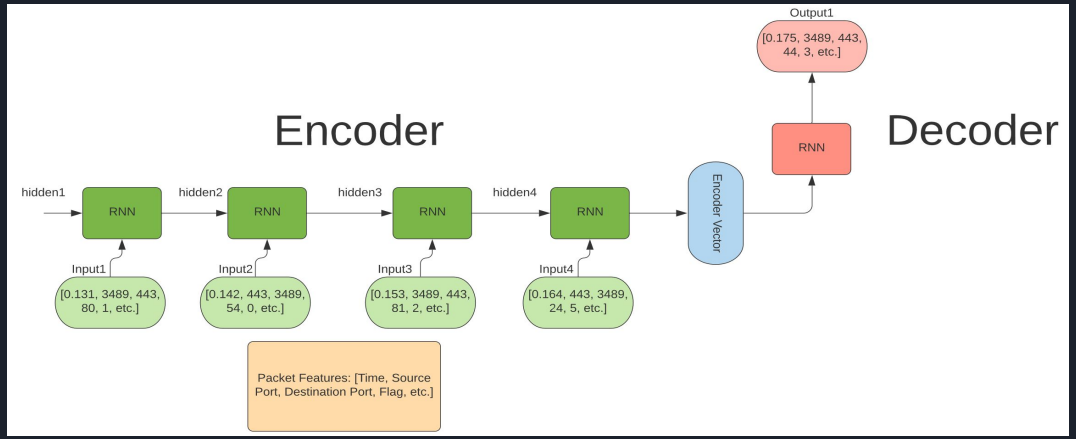
While we successfully implemented these clustering models on this dataset to identify anomalies, we moved away from it for two main reasons:

1. This implementation has been done extensively already
    a. It is often referred to as the “hello world” of anomaly detection in ML

2. We wanted to focus on a more niche application of a ML model in cyber-security

What is Sequence-to-Sequence (Seq2seq)

- Supervised Deep Learning model introduced by Google in 2014
- Works by mapping a sequence of inputs to a sequence of outputs
- Consists of two Recurrent Neural Networks (RNNs)
  - Encoder: converts input sequence into hidden vector
  - Decoder: uses output from the encoder and built in hidden states to generate a predicted outputs sequence

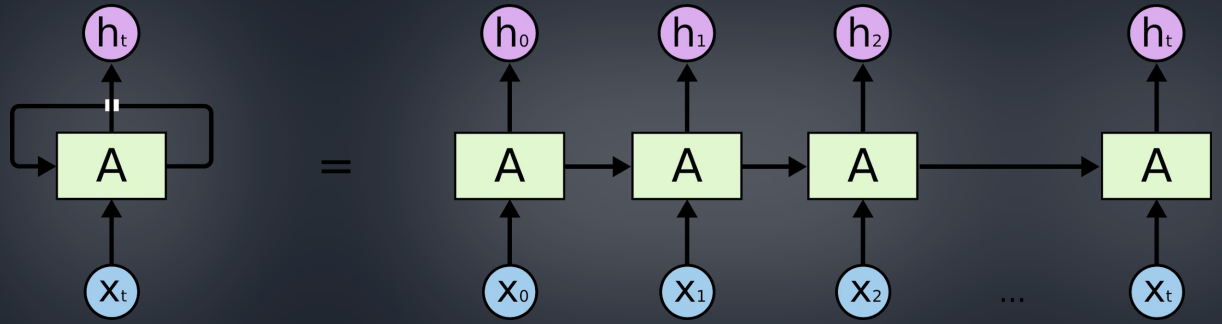


Our Seq2seq implementation

- Written in Python3 using the Pytorch library
- Adapted from a well known translation application using Pytorch
- Modified to use sequences of network packets (connections) instead of words (sentences)
- About 25 features considered - protocol, timestamp, ports, flags, etc
- Sequence formula: $T_n - T_1 < 1$ AND ($src\_ip_n = src\_ip_1$ OR $dest\_ip_n = src\_ip_1$)
    - Packets occur within 1 second, IP that initiated the connection is either src or dest
    - Min length = 4 packets, max length = 320 packets

Encoder

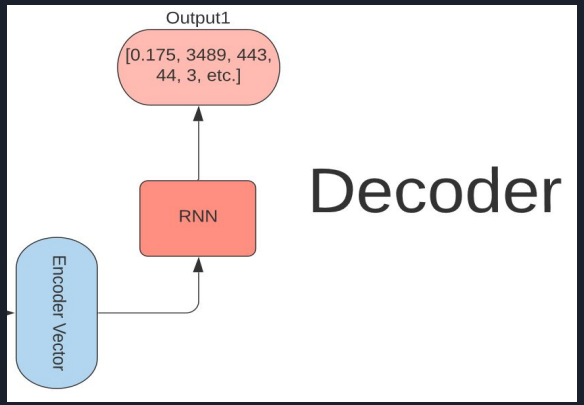
- Encoder's forward function reads one packet at a time and produces 2 things:
  - Output vector - continuously modified until the whole sequence is processed
  - Hidden state - condenses info about what has already been processed
- Output vector and hidden state used as input when the next packet is processed
- Final output - context vector to be passed to the decoder for prediction



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Decoder

- Works in similar fashion to the encoder
- Takes the output generated by the Encoder as input
- Produces a predicted output using the Encoder output and its hidden states



Training workflow/steps

Training Function

Step 1: We made a script to parse the DARPA 1999 Dataset into sequences of 4 to 320 packets. These sequences were saved into a ".csv" file and then inputted into the training script. We encoded each numeric column into z-score and each categorical one into a one-hot encoded vector so they can be used by the model as a feature.

Step 2: We parsed the sequences of packets into pairs of "input" and "target". The "input" is the first 3 packets of the sequence and the "target" is the rest of the packets in the sequence. The "input" sequence will be used by the model to predict the "target" sequence.

Step 3: The input sequence is then fed into the Encoder, so that each input can be condensed to a single vector representation with the use of a GRU. The encoder outputs this vector and a "hidden state" vector that is then inputted into the Decoder. The Decoder produces a prediction of the target packet sequence.

Step 4: The predicted output is then compared to the actual target packet sequence and the loss between the two values is calculated. The model then learns from this value and uses it to change its parameters to make the loss value lower, increasing accurracy of predictions.

Steps 3 and 4 happen continuously until all iterations of the sequences end. We then save the parameter values from the last iteration into a file for use in the test function.

Testing Function

Steps 1 to 3 from the training function are repeated in the testing function. However, the saved parameter values from the training function are used as the model parameters for the testing function.

↓

Step 1: The predicted output is then compared to the actual target packet and the loss between both values is calculated. The model then determines if the sequence of packets is an anomaly or not based on a specified threshold value for the loss.

↓

Step 2: The total number of anomalies detected is then outputted to the screen, along with the overall RMSE score for the normal and attack sequences.

DARPA Dataset Parser

```
    # loop to parse generate sequences
    for x in range(pos, pos + 320):
        # if packet does not have the same source or destination IP as the source of the first one in the sequence
        if (input_data[x][2] != src1 and input_data[x][3] != src1 and len(sequence)>3):
            break
        # if the current one did not happen within 1 second of the first one in the sequence
        elif ((input_data[x][1]-t1)>1 and len(sequence)>3):
            break
        else:
            row = input_data[x]
            sequence.append(row)  # Append the sequence with the current packet
            num_packets = num_packets + 1  # Update the number of packets
```

```
Number of packets in sequence:  4
Sequence:
      No.    timestamp          src_ip          dst_ip protocol  length  src_port  dst_port
0  12772  1379.713178    135.8.60.182  172.16.114.148      TCP      60      2871        21
1  12773  1379.713816  172.16.114.148    135.8.60.182      TCP      60        21      2871
2  12774  1379.713999    135.8.60.182  172.16.114.148      TCP      60      2871        21
3  12775  1379.714079  172.16.114.148    135.8.60.182      TCP      60        21      2871
Sequence Number:  368
============================================================
```

Training/Testing Encoder

```
3     def __init__(self, input_size, hidden_size):
4         super(Encoder, self).__init__()
5         self.hidden_size = hidden_size
6         self.gru = nn.GRU(input_size, hidden_size) # Applies Gated Recurrent Unit (GRU) to input sequence
7
8     def forward(self, input_token, hidden_state):
9         input_token = input_token.unsqueeze(0).unsqueeze(0)
10        output_vector, hidden_state = self.gru(input_token, hidden_state)
11        return output_vector, hidden_state
```

Training/Testing Decoder

```
3     def __init__(self, hidden_size, output_size):
4         super(Decoder, self).__init__()
5         self.hidden_size = hidden_size
6         self.gru = nn.GRU(output_size, hidden_size) # Applies GRU
7         self.out = nn.Linear(hidden_size, output_size)
8         self.relu = nn.LeakyReLU()
9
10    def forward(self, input_token, hidden_state):
11        input_token = input_token.unsqueeze(0).unsqueeze(0)
12        output_vector = self.relu(input_token)
13        output_vector, hidden_state = self.gru(output_vector, hidden_state)
14        output_vector = self.out(output_vector[0])
15        return output_vector, hidden_state
```

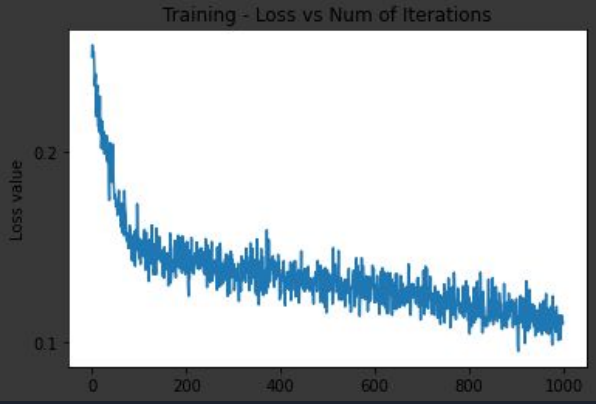
Training the Model and Output

```
hidden_size = 256
encoder1 = Encoder(input_size=25,hidden_size=hidden_size).to(device)
decoder1 = Decoder(hidden_size=hidden_size,output_size=25).to(device)

train_iterations(encoder1, decoder1, n_iterations=100000, print_every=1000, plot_every=100, learning_rate=0.0001)
```

```
18m 54sec (- 0m 23sec) (98000 98%) Current Loss Value ----> 0.1106
19m 6sec (- 0m 11sec) (99000 99%) Current Loss Value ----> 0.1115
19m 17sec (- 0m 0sec) (100000 100%) Current Loss Value ----> 0.1100
================================================================
-------> MODEL HAS FINISHED TRAINING <-------

Plotting Model......
<Figure size 432x288 with 0 Axes>
```



Testing Model on Data and the Output

```
count_anomalies, rmse_score, normal_rmse_score, atk_rmse_score = test_model(encoder1, decoder1, print_every = 1000, threshold = 0.5)

print('Potentially anomalous sequences: ', count_anomalies)
print('Overall RMSE score: ', rmse_score)
print('Normal RMSE score: ', normal_rmse_score)
print('Attack RMSE score: ', atk_rmse_score)
print("---> Finished Testing Function <---")
```

```
Percent complete: 92%
Percent complete: 94%
Percent complete: 96%
Percent complete: 97%
Percent complete: 99%
Potentially anomalous sequences:  2080
Overall RMSE score:  0.340393348431479 34
Normal RMSE score:  0.33410887812161344
Attack RMSE score:  0.5301994586892161
---> Finished Testing Function <---
```

Results

- We trained our model on random packet sequences in our training dataset and reached an average loss value of 0.1114 after training on 100k samples
- Using this trained model, we were able to detect 2,080 potentially anomalous sequences within our test set, containing attack data

```
Potentially anomalous sequences:  2080
Overall RMSE score:  0.340393348431479 34
Normal RMSE score:  0.33410887812161344
Attack RMSE score:  0.5301994586892161
---> Finished Testing Function <---
```