# Applying Artificial Neural Networks to Graph Databases to identify Reconnaissance Tactics in UWF-ZeekData22

# Application of Graph Algorithms and Artificial Neural Networks with UWF-ZeekData22

# Abstract

**Novelty:**  Feature selection applied to a graph, deriving new features from the resulting graph, gaining a both a detailed and generalized perspective on the network activity and the network environment it is occurring in, and applying Artificial Neural Network algorithms to the resulting graphs to accurately identify specific types of cyberattacks.

Keywords: Machine Learning, Artificial Neural Networks, Graph Algorithms

# Introduction

The application of graph algorithms and artificial neural networks in conjunction with UWF-ZeekData22 data to process and analyze network activity and accurately identify a cyber attack tactic is the focus of this effort. Feature selection involved using attributes that measured a network connection based on the connection’s duration, connection’s data volume, and the unique ports utilized by the connection. The features selected were then aggregated for all connections between the source and destination network IP (Internet Protocol) as the key. A graph was generated from the aggregated results, using the source and destination IP as two vertices with a directed edge between source and destination, detecting and avoiding cycles during construction. Each edge is populated with the aggregated features (duration, bytes, and unique ports) and augmented by the number of aggregated connections. As the graph is formed, algorithms can be applied to derive additional features not in the original dataset. Each subgraph is traversed and edges are augmented with the number of vertices and unique ports found in the subgraph. Once processed in this way, each edge now contains not only detailed information regarding how two vertices were inner-related but general information about the subgraph the edge is contained within. This gives an edge to a detailed and general perspective of its potential role in a cyber attack. This data could then be used to train an artificial neural network to correctly identify specific cyber attack tactics with a high level of accuracy.

**Why are we doing this?**

Combining the results from the application of graph algorithms with Machine Learning (ML) algorithms seemed the next logical step in investigating the benefits of these two technologies. Specifically, the application of Artificial Neural Networks (ANN) was the next logical step in the creation of a system that could identify specific tactics involved in cybersecurity intrusion detection.

**Why are cyber attack studies important?**

Being able to study and detect ever more sophisticated and complicated cyber attacks is important in protecting critical infractures from multiple attack tactics. Being able to detect a wide range of attacks requires studying tactic signatures which involve numerous data attributes. Transforming large volumes of network traffic information into attributes that can be quickly evaluated to produce an identifiable attack tactic is the focus of this paper.

**Why graphing?**

A bit more general explanation on Graph databases and why they are important is needed here, before we go into UWF-ZeekData22

Using graph algorithms to organize the UWF-ZeekData22 allowed attributes to be associated collectively based on their relationships to other vertices (IP addresses) in the graph. This allowed the attributes to gain a relationship aspect that isn’t apparent when strictly using attribute values. Once the attributes were organized into a graph, aspects of the data could be evaluated. This resulted in derived information that enhanced the ability of the machine learning (ML) algorithm to detect an attack with a higher level of confidence.

**Why ANN?**

Artificial Neural Networks (ANN) allow you to incorporate disparate data points into a single, unified set of data to make decisions. Using ANN allowed incorporating data that originated from the vertex's inner connections as well as the overall subgraph’s information. This allowed the ANN to factor in the aspects that might otherwise might be ignored in other ML algorithms.

# Related Works

Related papers [2] [13] [14].

This paper extends the work started in [2] by applying ANN algorithms to the data as well as tailoring the data to the ANN algorithms. Although aggregation and subsequent graphing of the data occurred in [2], no attempt was made to further synthesize the data for use by ML algorithms. This effort focused on not only on ANN but on how to model the data to facilitate the use of ML algorithms. Although [13] took the approach of using nodes (vertices in this paper) to represent the data and edges to represent the relationships, we took the approach of storing aggregated relational data on the edge between the two vertices in a relationship. This quantified the relationship in a way that [13] did not. Fitness widgets from [13] were not implemented but a similar thought process was taken to create new data based on the resulting graph.

In addition, [14] makes mention of using provenance data as best practice for attack detection, describing interactions by the network entities, first by how the entities interact and second by events that occur over a period of time. We did not focus on the process or file, but rather on the interaction itself and scaled that interaction based on duration, volume of data, and volume of interactions.

This effort is unique in that the final constructed graph contains both detailed information about interactions between network endpoints as well as information regarding the overall environment in which those endpoints communicated.

# The UWF-ZeekData22 dataset

UWF-ZeekData22 dataset is a publicly available at datasets.uwf.edu and is a network dataset collected using Zeek and labeled using the MITRE ATT&CK framework [17]. Specifically the “Conn” dataset was used to build the graphs and train the ANN.

# Data Strategies

#### Feature Selection, aggregation, and representation:

The UWF-ZekaData22 dataset was reviewed to determine which attributes would be selected for use within graph based methodologies and algorithms. The process to obtain, aggregate, and represent the data in a graph was chosen to view the data spatially instead of just as raw data points. The process applied is detailed below:

#### Phase 1 - Aggregating data by source to destination connections

The purpose of this phase was to reduce the volume of data to a level that could be incorporated into a graph, using the edges to represent the volume of data for each selected attribute. The feature selection methodology used was to select from the attributes available, those that represented the volume (both connection and byte count) and duration of communications between two network nodes. The source IP and destination IP were selected for use as the key for aggregation. A count of the same source to destination was maintained and saved as well as the following attributes: bytes transferred, connection duration, and a unique list of ports used across the source to destination connection. This resulted in the following data points (features): sourceIP->destinationIP key, count of connections involving the source and destination, total number of bytes, total duration, and list of unique ports utilized. Additionally the tactic is retained, per edge, for use during training the neural network.

Aggregation was done by using linked hash maps in Java (key:value pairs), where key = srcIP,dst IP and values are kept in separate keyValue maps for each attribute (connections, bytes, duration, portList), and tactic (for use in training).

for each record in the input file

key = srcIP,dstIP

if key has not been encountered before

set key:connectionCount to 1

set key:bytes to record:bytes

set key:duration to record:duration

set key:portList to record:ports

set key:tactic to record:tactic (only one allowed per key)

else

increment key:connectionCount

add record:bytes to key:bytes

add record:duration to key:duration

add record:ports to key:portList

The portList is an accumulation of unique ports involved in all connections involving the key (srcIP->dstIP).

**Example input:**

| **srcIP** | **dstIP** | **bytes** | **duration** | **port** | **tactic** |
| --- | --- | --- | --- | --- | --- |
| 143.88.2.10 | 143.88.7.15 | 0 | 0.00879049 | 19315 | Reconnaissance |
| 143.88.2.10 | 143.88.7.15 | 0 | 0.00607013 | 5825 | Reconnaissance |
| 143.88.2.10 | 143.88.7.15 | 0 | 0.00658032 | 9968 | Reconnaissance |
| 143.88.2.10 | 143.88.7.1 | 0 | 0.00019192 | 3527 | Reconnaissance |

**Resulting aggregation:**

| **srcIP** | **dstIP** | **count** | **bytes** | **duration** | **port(s)** | **tactic** |
| --- | --- | --- | --- | --- | --- | --- |
| 143.88.2.10 | 143.88.7.15 | 3 | 0 | 0.02144094 | 19315,5825, 9968 | Reconnaissance |
| 143.88.2.10 | 143.88.7.1 | 1 | 0 | 0.00019192 | 3527 | Reconnaissance |

*Explain color coding.*

#### Phase 2 - Constructing a graph from the aggregated data

The purpose of this phase was to graphically represent the source and destinations as vertices in a directed, non-cyclic graph. Each edge between the connected vertices contained the connection count, total number of bytes across that connection, total duration of the connections, and unique ports utilized by the connections. While constructing the graph, if an edge would result in creating a cycle in the graph, that edge was removed from the graph.

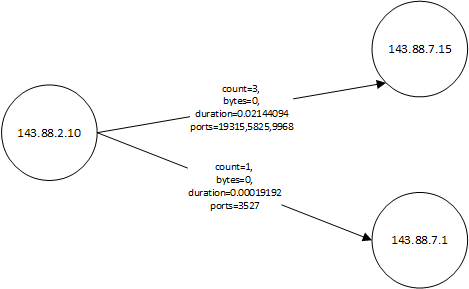
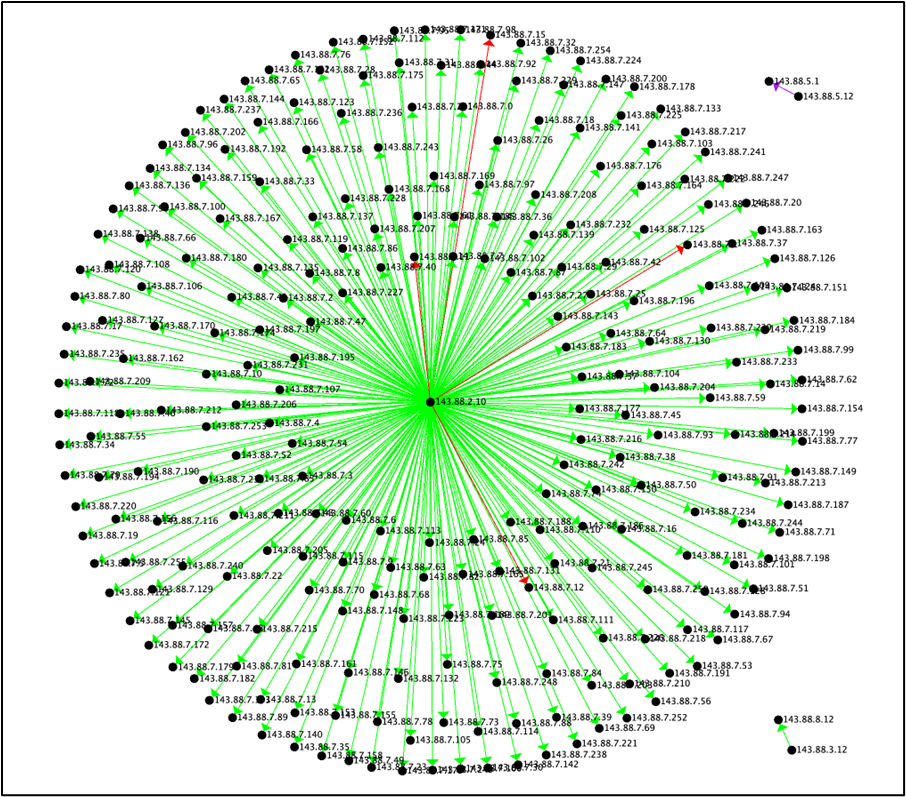


Figure x - Resulting graph from Phase 1 example

*Elaborate by using previous “live” graph from previous paper.*

**

*Graph of connection for a Recon ….*

#### Phase 3 - Locating vertices with an inDegree value of zero

The purpose of this phase was to locate all subgraphs created by creating the graph. The vertices and their edge in the subgraph are used to obtain the number of vertices contained within the subgraph and to create a subgraph level list of unique ports used within the subgraph. This information (graph vertice count and count of unique ports in the graph) is then distributed to each edge within the subgraph. This adds an additional dimension to the edge data, allowing each edge to represent not only the interaction between the vertices but also information about the subgraph that the edge is contained within.

for each vertex (V) in graph (G)

if inDegree(V) is zero

set verticeCount to 0

set portList to empty

for each edge (E) exiting vertex (V)

add 1 to verticeCount

for each port used by E

if portList does not contain port

add port to portList

set portCount to the number of ports in portList

for each edge (E) exiting vertex (V)

add verticeCount to edge (E)

add portCount to edge (E)

**Example input:**

| **srcIP** | **dstIP** | **count** | **dur** | **bytes** | **ports** | **tactic** |
| --- | --- | --- | --- | --- | --- | --- |
| 143.88.3.12 | 143.88.8.12 | 4 | 10.13 | 7723 | 631, 80 | Reconnaissance |
| 143.88.2.10 | 143.88.7.6 | 1024 | 0.41 | 0 | 80, 443 | Reconnaissance |
| 143.88.2.10 | 143.88.7.4 | 1024 | 0.37 | 0 | 80, 443 | Reconnaissance |
| 143.88.2.10 | 143.88.7.15 | 1024 | 0.28 | 0 | 80, 443, 1723 | Reconnaissance |

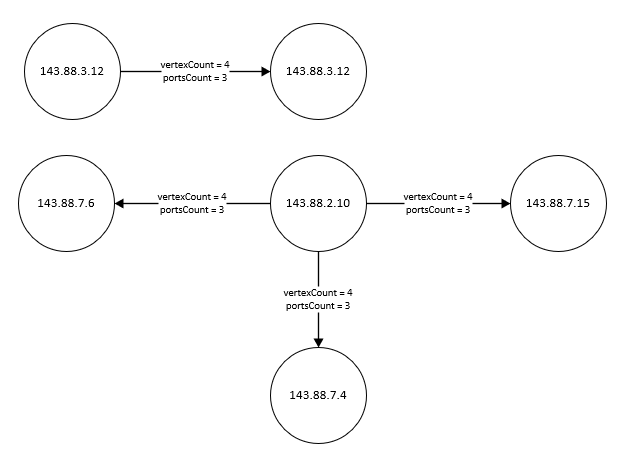
**Resulting information:**

| **srcIP** | **dstIP** | **count** | **dur** | **bytes** | **portsCount** | **vertexCount** | **tactic** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 143.88.3.12 | 143.88.8.12 | 4 | 10.13 | 7723 | 2 | 2 | Reconnaissance |
| 143.88.2.10 | 143.88.7.6 | 1024 | 0.41 | 0 | 3 | 4 | Reconnaissance |
| 143.88.2.10 | 143.88.7.4 | 1024 | 0.37 | 0 | 3 | 4 | Reconnaissance |
| 143.88.2.10 | 143.88.7.15 | 1024 | 0.28 | 0 | 3 | 4 | Reconnaissance |

*Explanation of the above information:*

*Line 1: IP address 143.88.3.12 created a graph with two vertices (143.88.8.12 and itself) and involved   
 two ports (631 and 80).*

*Line 2, 3, and 4: IP address 143.88.2.10 involved a total of four vertices (143.88.7.6, 143.88.7.4,  
 143.88.7.15, and itself) and involved a total of three unique ports (80, 443, and 1723).*



**Figure x - Resulting graph from phase 3 example**

#### Phase 4 - Normalizing Values

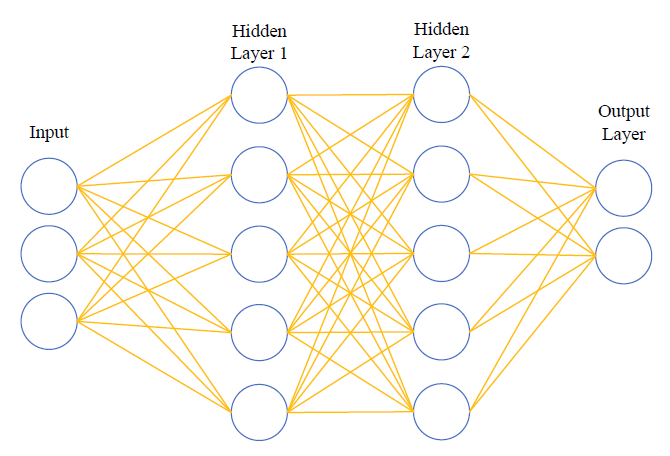
The purpose of this phase was to extract the data from the graph and normalize it in preparation for use by the neural network. The following attributes were extracted and normalized from each edge: src->dst key, subgraph vertex count, subgraph port count, edge connection count, edge byte count, edge duration. These attributes, along with the identified tactic, were then used to train the neural network.

## Artificial Neural Networks

A neural network, also known as an artificial neural network (ANN) was utilized to identify network activity. A neural network is modeled after the human brain and consists of a network of “neurons” that generally have a threshold that, once reached, will “fire” and propagate to the next layer of neurons in the network. These networks have two “visible” layers that consist of the input layer, consisting of input information, and an output layer, consisting of the possible outcomes. The input information is fed into one or more hidden layers in the neural network. The neurons in the hidden layers receive values which are multiplied by an individualized weight that is established during training. This weight is “learned” during training by multiple trial and error cycles, similar to a child trying something for the first time, failing, and then adjusting the approach and trying again. Additionally, a bias is also applied across all neurons in a layer to strengthen or weaken all neurons in a given layer.

Each layer in an ANN has a specific activation function applied to it. These functions control how that layer reaches the necessary value to activate the neurons in that layer. As part of the learning process, there is both forward and backward propagation of data. Forward propagation involves using the input data, passing it through the various layers and their activation functions. The results of this pass are compared to expected results and that information is passed backwards through the neurons, adjusting the weights of each in an attempt to fine tune the firing of each neuron, again similar to a child learning by trial and error. This forward and backward process is iteratively done, with each forward and backward cycle referred to as an epoch. This process occurs until the desired outcome is achieved or the number of learning cycles have been exhausted, again similar to a child giving up on a complex problem.

With each epoch, two values are calculated to determine the effectiveness of the training. First is accuracy, which is the number of correct predictions divided by the total number of predictions. Next is the loss function, which represents how wrong the model is. (4) And finally the learning rate, which is the rate at which the gradient is adjusted with each epoch. (4)



**Figure x - A Basic Neural Network**

## Specific Implementation of ANN

In this work, only two layers were implemented, each with their own activation function. The first activation function implemented was the Rectified Linear Unit (ReLU) function. This activation function is common in deep learning and provides a value of zero for any negative input values.

The second activation function implemented was the Softmax activation function. This activation function is used to convert values to a normalized probability distribution. As such, this activation function is used as the final layer of the neural network.

Additionally, this neural network implements multiple commonly used optimizers: Stochastic Gradient Descent (SGD), Adaptive Gradients (AdaGrad), Root Mean Squared Propagation (RMSProp), and Adam (a combination of AdaGrad and RMSProp).

Stochastic Gradient Descent (SGD) - Popular and common algorithm used in NNs. The normal gradient descent optimization algorithm is iterative and starts from a random point and moves down the slope with each iteration until the lowest point of the function is reached. SGD speeds up the process by selecting a random data point from the whole data set with each iteration, thus reducing the number of computations required.

Adaptive Gradients (AdaGrad) - Is an extension of the gradient descent optimization and allows the step size in each dimension used by the optimization algorithm to automatically adapt depending on the gradients seen during the course of the search [5].

Root Mean Squared Propagation (RMSProp) - Another popular optimization algorithm, first proposed by Geoff Hinton. This algorithm is an extension of gradient descent and AdaGrad and uses a decaying moving average and utilizes recently observed partial gradients [6].

Adam (a combination of AdaGrad and RMSProp) - A combination of AdaGrad, RMSProp, as well as momentum into one algorithm [8].

# Experiments

## Learning Results

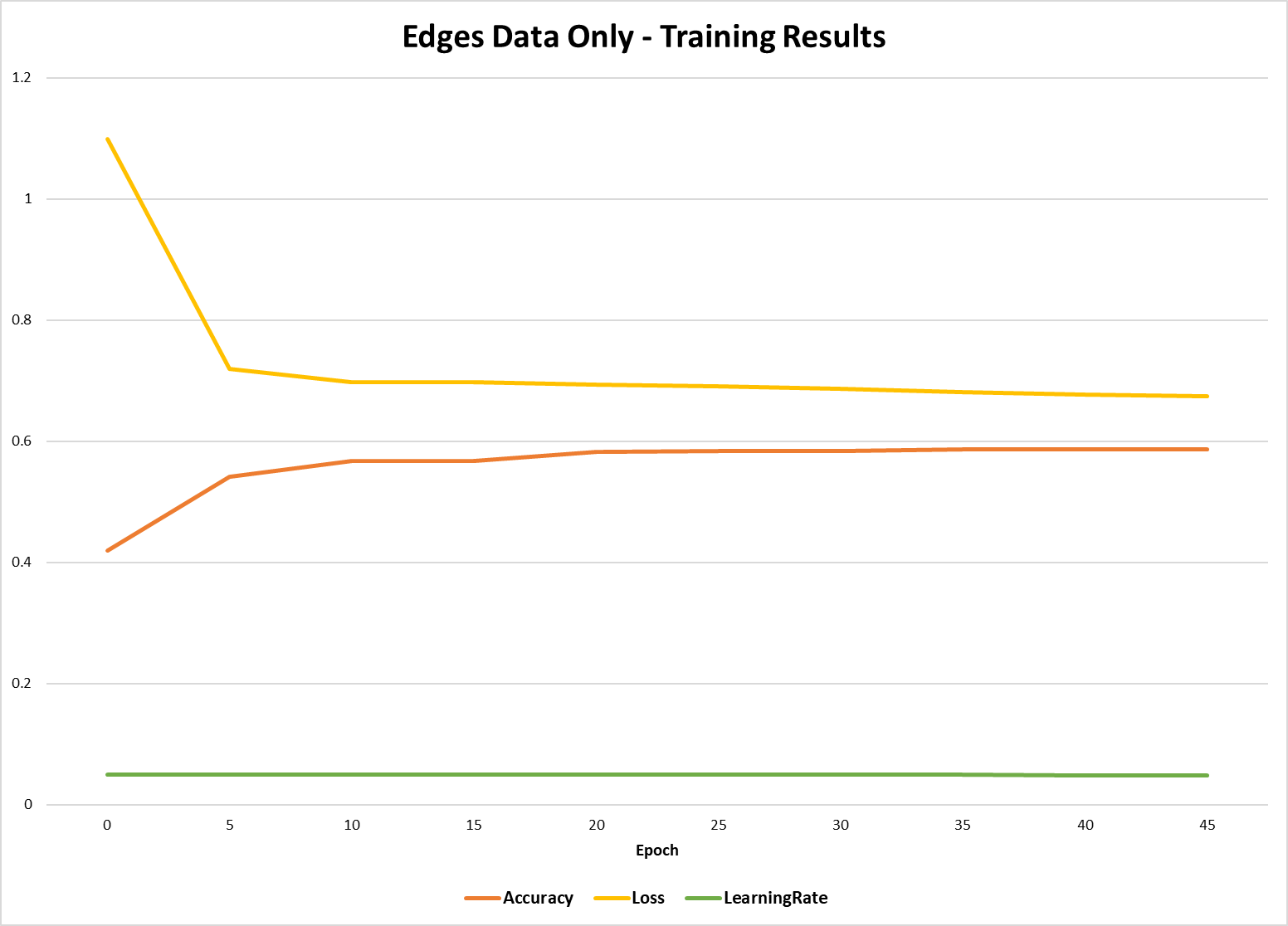
A growth approach was utilized, by adjusting feature selection, deriving new features from existing features, synthesizing new features from the resulting generated graph, or from normalization of values. The results from each technique were evaluated to determine if the technique applied to the data improved or degraded the training results of the ANN.

### Edge Data Only: Normalized edge data (count, bytes, duration):

This approach utilizes only the attributes associated with the edges between the vertices of the graph. No attempt was made to “visualize” the overall graph that the edge was a part of.

**Example data used:**

| **Connection Count** | **Bytes** | **Duration** |
| --- | --- | --- |
| 0.000 | 0.000 | 0.003 |
| 0.078 | 0.000 | 0.000 |
| 0.247 | 0.102 | 1.000 |
| 0.463 | 0.280 | 0.000 |
| 0.311 | 0.081 | 0.000 |



**Figure x - Edge Data Only - Training Results**

**Training results:**

| **Epoch** | **Accuracy** | **Loss** | **Learning Rate** |
| --- | --- | --- | --- |
| 0 | 0.42 | 1.099 | 0.05 |
| 5 | 0.542 | 0.719 | 0.05 |
| 10 | 0.567 | 0.697 | 0.05 |
| 15 | 0.567 | 0.697 | 0.05 |
| 20 | 0.582 | 0.694 | 0.05 |

#### Normalized Edge Data results and Lessons Learned:

The results of this approach were somewhat promising as accuracy increased as loss decreased but a high level of accuracy and a low level of loss seemed to falter at approximately the 5th epoch and only minimally improved but never reached above 60%. To improve the results, Technique 2 was created.

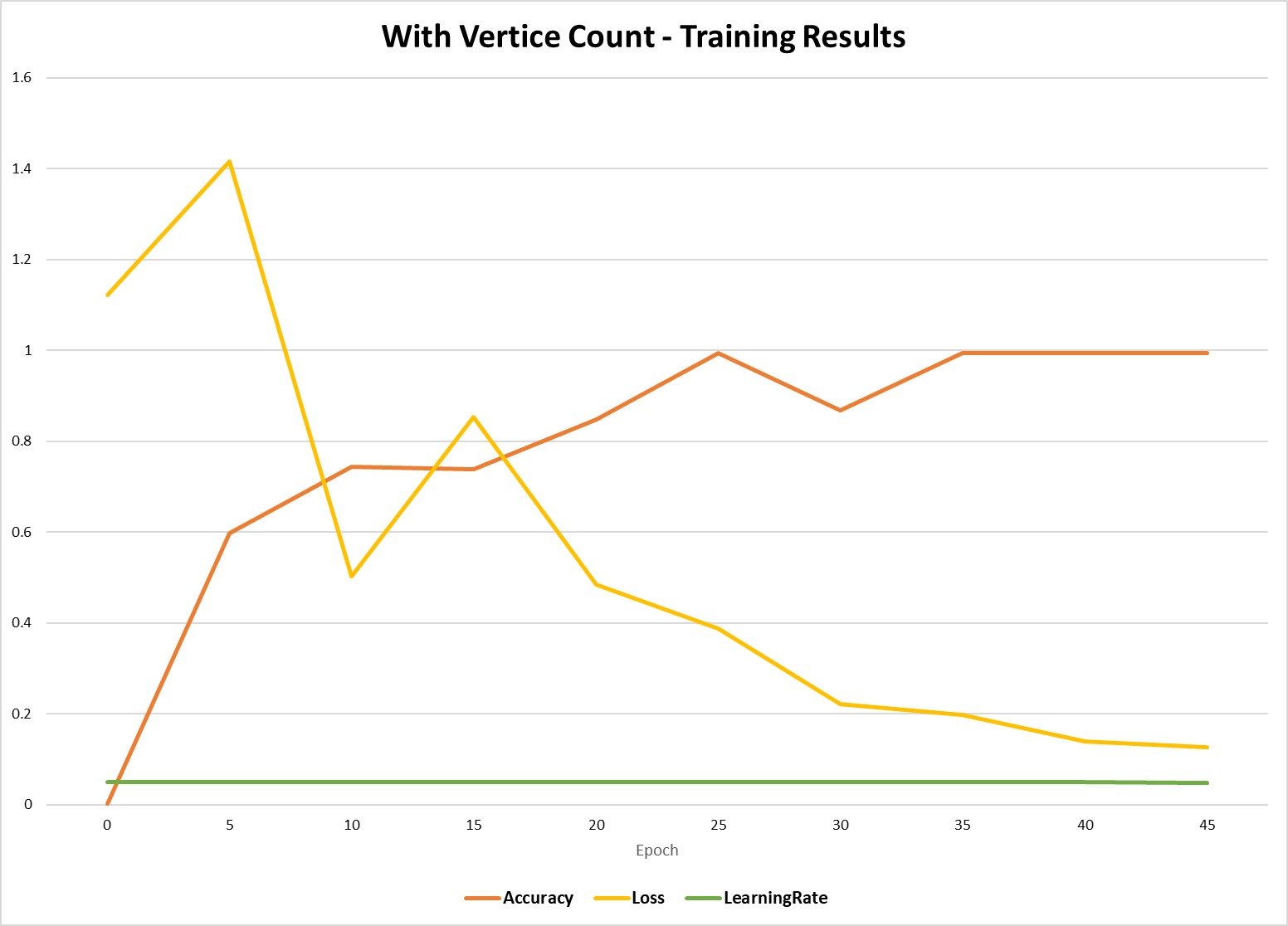
Trying to identify an attack tactic based solely on the number of connections, byte count, and connection duration between two vertices (IP addresses in a network) is not sufficient to incorporate other edges involved in the attack. It was proposed that incorporating a metric that captured the scale of the graph was needed. This would allow the ANN to factor in the number of other vertices involved in the overall attack. The following technique is the result of this proposal and is detailed below.

### With Vertice Count: Non-normalized Graph Vertex Count with normalized edge data (count, bytes, duration):

This technique augmented Technique 1 by adding the number of vertices contained within the subgraph. The new count was not normalized and ranged from 1 to 257. The training results are presented in Figure x.

**Example data used:**

| **Subgraph Vertice Count** | **Connection Count** | **Bytes** | **Duration** |
| --- | --- | --- | --- |
| 257.0000 | 0.0003 | 0.0000 | 0.0033 |
| 257.0000 | 0.0776 | 0.0004 | 0.0000 |
| 257.0000 | 0.2471 | 0.1019 | 1.0000 |
| 1.0000 | 0.0747 | 0.2722 | 0.0000 |
| 70.0000 | 0.0024 | 0.0000 | 0.0000 |



**Figure x - With Vertice Count - Training Results**

**Training results:**

| **Epoch** | **Accuracy** | **Loss** | **Learning Rate** |
| --- | --- | --- | --- |
| 0 | 0.002 | 1.122 | 0.05 |
| 5 | 0.597 | 1.416 | 0.05 |
| 10 | 0.744 | 0.503 | 0.05 |
| 15 | 0.739 | 0.854 | 0.05 |
| 20 | 0.847 | 0.484 | 0.05 |

#### Non-normalized Graph Vertex Count results and Lessons Learned:

The results of this approach were somewhat chaotic with accuracy steadily increasing and peaking around the 25th epoch, followed by some instability which was regained around the 35th epoch. Loss seemed somewhat erratic as well, with peaks and valleys before setting down to a gradual decrease around the 20th epoch. This may have been due to adding a non-normalized data element, which may have had an initial overwhelming effect on early epochs.

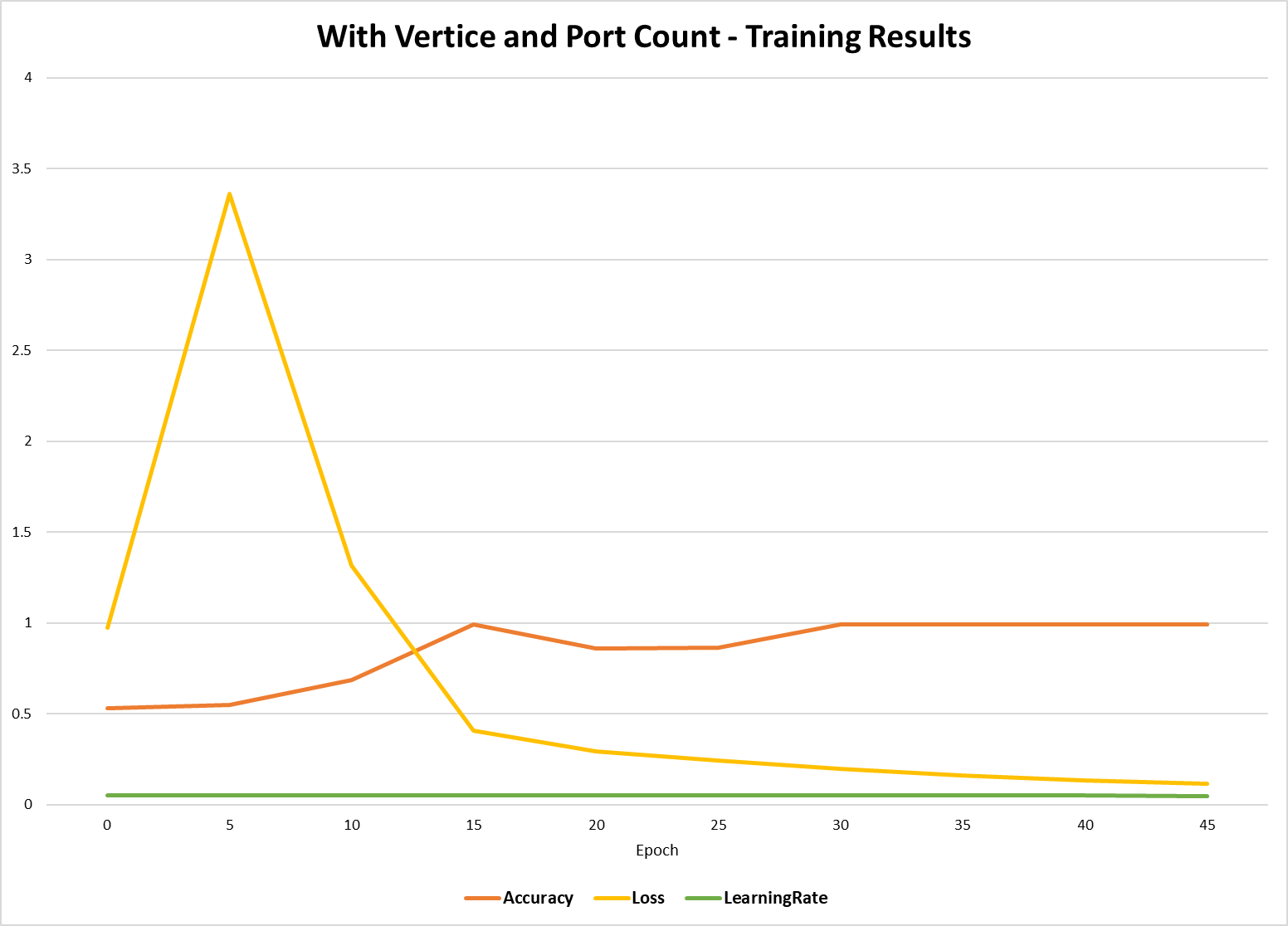
This approach was promising as it ultimately achieved the accuracy and loss values desired. With this encouragement, it was decided to add port data back into the attributes, as ports had initially been eliminated as representing ports appeared to be overly complicated and as a wide range of ports were possible (ranging from 0 to 65,536), it appeared to be of little or no value. With vertex counts showing promise, it was proposed that a count of unique ports used throughout the subgraph could add valuable information regarding the attack tactic. This approach is presented in the next technique.

### With Vertice and Port Count: Non-normalized Graph Vertex Count, with Unique Port Count, with normalized edge data:

This approach augmented Technique 2 by adding the number of unique ports utilized in the subgraph. The port count was not normalized and ranged from 1 to 1814. The training results are presented in Figure x2.

**Example data used:**

| **Graph Vertices** | **Graph Ports** | **Connection Count** | **Bytes** | **Duration** |
| --- | --- | --- | --- | --- |
| 257.000 | 1003.000 | 0.000 | 0.000 | 0.003 |
| 257.000 | 1002.000 | 0.078 | 0.000 | 0.000 |
| 257.000 | 1011.000 | 0.247 | 0.102 | 1.000 |
| 257.000 | 1009.000 | 0.463 | 0.280 | 0.000 |
| 257.000 | 1005.000 | 0.311 | 0.081 | 0.000 |



**Figure x - With Vertice and Port Count - Training Results**

**Training results:**

| Epoch | Accuracy | Loss | Learning Rate |
| --- | --- | --- | --- |
| 0 | 0.529 | 0.974 | 0.05 |
| 5 | 0.548 | 3.361 | 0.05 |
| 10 | 0.685 | 1.318 | 0.05 |
| 15 | 0.994 | 0.408 | 0.05 |
| 20 | 0.859 | 0.294 | 0.05 |

#### Non-Normalized Graph Vertex Count with Unique Port Count results and Lessons Learned:

The results of this approach achieved a high level of accuracy at approximately epoch 15 but had an extreme loss value during the initial epochs until a steady reduction was achieved. Although accuracy was obtained at epoch 15, it appeared unstable and fluctuated until approximately epoch 30. It was proposed that the non-normalized values may have been the cause of the instability. Technique 4 is the implementation of this proposal and involves normalization of non-normalized attributes.

### Normalized Vertice and Port Count: Normalized Graph Vertex Count, with Unique Port Count, with normalized edge data:

This approach enhanced the previous technique by normalizing the vertex and port count. The training results are presented in Figure x.

**Example Data Used:**

| **Graph Vertices** | **Graph Ports** | **Connection Count** | **Bytes** | **Duration** |
| --- | --- | --- | --- | --- |
| 0.9961 | 3.8988 | 0.0003 | 0.0000 | 0.0033 |
| 0.9961 | 3.8949 | 0.0776 | 0.0004 | 0.0000 |
| 0.9961 | 3.9300 | 0.2471 | 0.1019 | 1.0000 |
| 0.9961 | 3.9222 | 0.4628 | 0.2804 | 0.0000 |
| 0.9961 | 3.9066 | 0.3113 | 0.0808 | 0.0000 |

# 

**Figure x - Normalized Vertice and Port Count - Training Results**

**Training Results:**

| Epoch | Accuracy | Loss | Learning Rate |
| --- | --- | --- | --- |
| 0 | 0.032 | 1.099 | 0.05 |
| 5 | 0.586 | 0.487 | 0.05 |
| 10 | 0.992 | 0.096 | 0.05 |
| 15 | 0.994 | 0.058 | 0.05 |
| 20 | 0.994 | 0.06 | 0.05 |

#### Normalized Graph Vertex Count with Unique Port Count results and Lessons Learned:

The results of this technique were the most promising of the results as accuracy increased in a nearly inverse proportion to the loss, reaching desired results at approximately the 15th epoch. At this point the ANN appears to be functioning as desired, and has a smooth increase in accuracy accompanied with a smooth decrease in the loss rate.

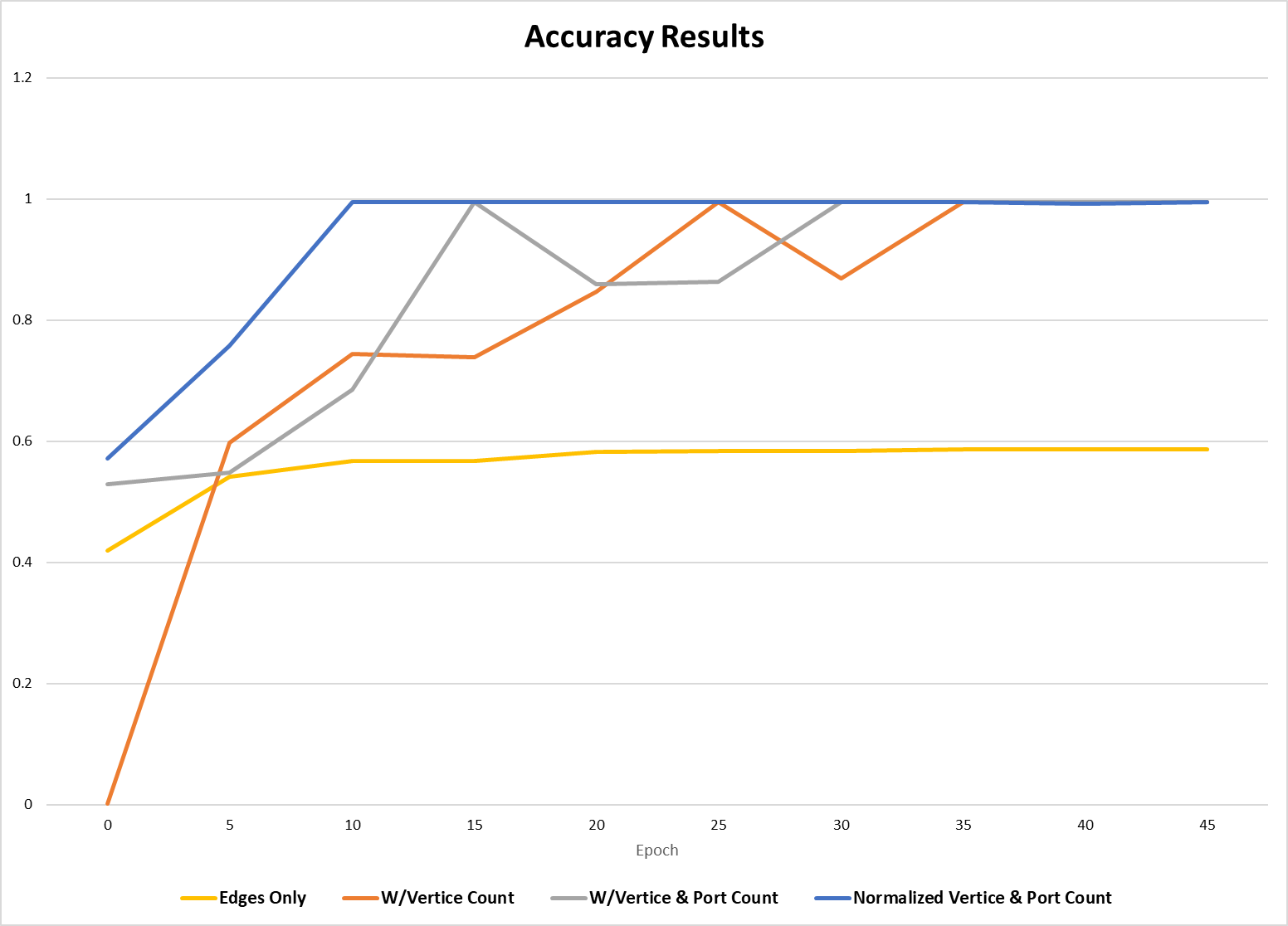
# Results

Utilizing basic ANN algorithms, it was shown that a combination of generalized subgraph information as well as detailed network traffic information was a determining factor in the performance of an ANN when applied to UWF-ZeekData22 data. In the case of intrusion detection, keeping both detailed information from the point to point connections (edges showing connections, duration, and bytes transferred) as well as more global information about the environment in which those connections took place (graph vertices and number of ports utilized) allowed the ANN algorithms to obtain a higher level of accuracy while reducing loss levels during training. From the figure presented in technique 4, you can see that once accuracy was obtained, it was stable and didn’t fluctuate as learning progressed. Technique 4 was the best of both aspects of the data and combined detailed information with global environmental data about the subgraph the edges were included in. This produced accuracy quickly and predictably.

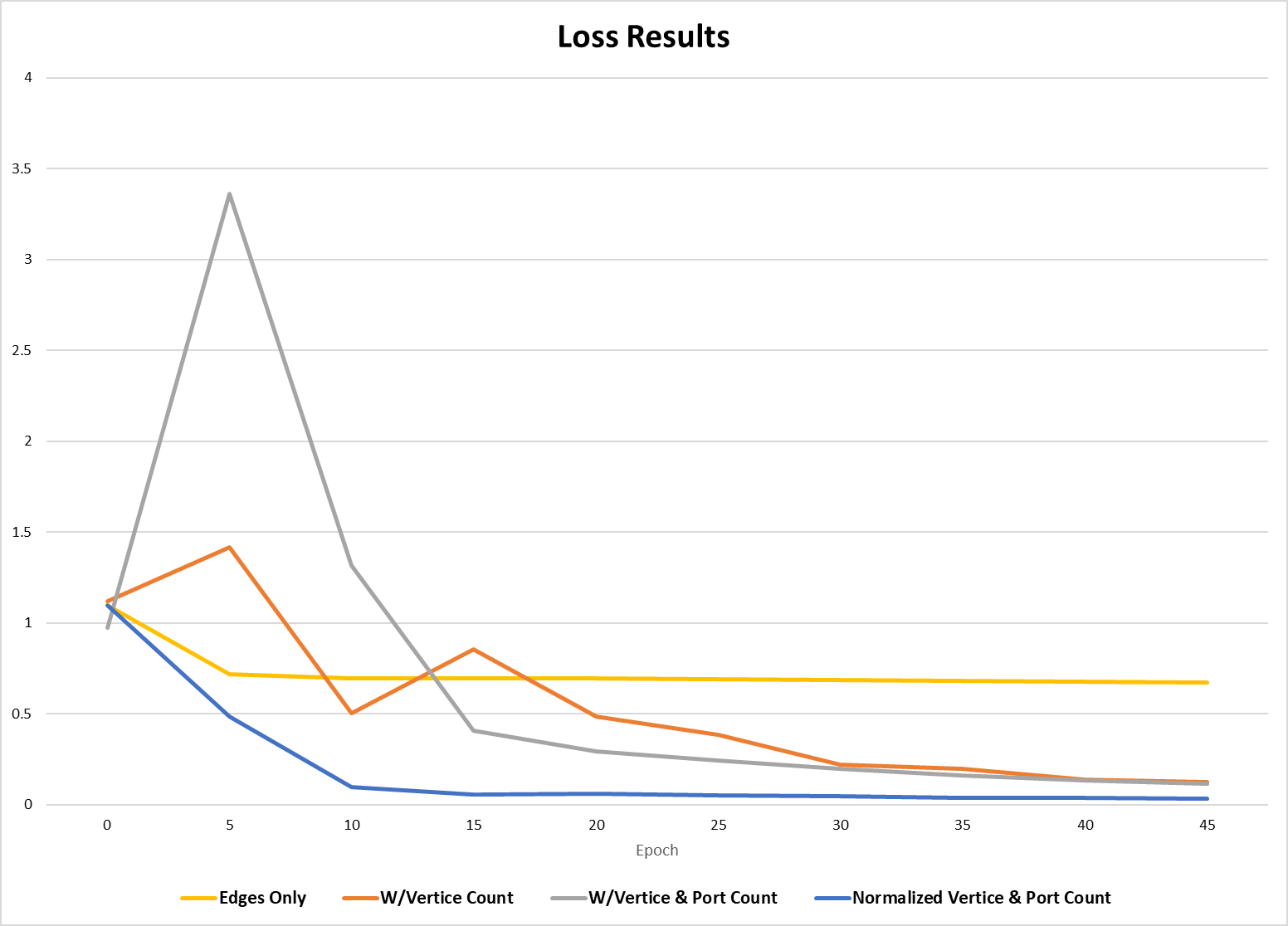
Below are the combined results of each of the techniques, in terms of accuracy and loss. Learning rate was not graphed as the rated applied was consistent across all 4 techniques and was therefore not an interesting or contributing factor to the results.

The comparison of accuracy, across all approaches taken, shows that using edges data only had the worst outcome at less than 59% accuracy, while adding just the vertice count to the edge data eventually provided a high level of accuracy but it took the most epochs to obtain this and at one point, near 30 epochs, the accuracy rate actually fell. Adding the subgraph’s unique port count to the vertice count, while promising at first, obtaining high accuracy near 15 epochs, wasn’t sustained and fell back until near 30 epochs. Only the normalization of the vertice and port counts provided a level of accuracy early in the process, at around 10 epochs, that was sustained. Thus this final approach was deemed successful.

Analysis of the loss ratio is also important in understanding how successful the data is in teaching the ANN. Using edge data only, the loss rate never dropped below 0.6, meaning that even with expected outcomes, the ANN could not reliably generate the correct answer. Adding vertice counts for the subgraph was better but actually had worse initial results that fluctuated until finally settling down around epoch 30, but never obtained a low level of loss. The next approach of adding the unique port across the subgraph, was very poor and ended near a high of 3.4, early in the process, but finally started to improve until near epoch 15 and slowed and never improved beyond the previous approaches loss level. When normalization was introduced to the subgraphs vertice and port counts, the loss rate quickly dropped to near zero by epoch 10 and obtained a steady rate thereafter. This graph, again shows that the final approach was the most successful of the four approaches used in training the ANN.



**Figure x - Accuracy comparison between the progressive approaches taken.**



**Figure x - Loss comparison between the progressive approaches taken.**

# Future Suggestions

Future work might include producing a real-time environment where edges have an “age” and would be eliminated over time. This would allow data to continually flow into the graph as network traffic occurs. At specific time increments, a new edge would form from new data while the older, but not timed-out edges would be used to detect what was going on in near real-time. Extractions would be made from the graph at a set time interval and that data would progress into the neural network to determine if an attack was occuring.

# References

1. Bagui, S., Mink, D., Bagui, S., Ghosh, T., McElroy, T., Paredes, E., ... & Plenkers, R. (2022). Detecting Reconnaissance and Discovery Tactics from the MITRE ATT&CK Framework in Zeek Conn Logs Using Spark’s Machine Learning in the Big Data Framework. *Sensors*, *22*(20), 7999.

2. Paper from last semester

3. Liu, Y. (2020). *Python Machine Learning By Example: Build intelligent systems using Python, TensorFlow 2, PyTorch, and scikit-learn, 3rd Edition* (3rd ed.). Packt Publishing.

4. Kinsley, Harrison, and Daniel Kukieła. *Neural Networks from Scratch in Python Building Neural Networks in Raw Python*. Printed by the Author, 2020.

5. Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7).

6. Hinton, G. (n.d.). *Neural Networks for Machine Learning*. Retrieved April 11, 2023, from  
 http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\_slides\_lec6.pdf

7. Ward, R., Wu, X., & Bottou, L. (2020). Adagrad stepsizes: Sharp convergence over nonconvex landscapes. *The Journal of Machine Learning Research*, *21*(1), 9047-9076.

8. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

9. Landro, N., Gallo, I., & La Grassa, R. (2020). Mixing Adam and SGD: a combined optimization method. *arXiv preprint arXiv:2011.08042*.

10. Dunne, R. A., & Campbell, N. A. (1997, June). On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne* (Vol. 181, p. 185). Citeseer.

11. Zou, F., Shen, L., Jie, Z., Zhang, W., & Liu, W. (2019). A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 11127-11135).

12. Leevy, J.L., Hancock, J., Zuech, R. *et al.* Detecting cybersecurity attacks across different network features and learners. *J Big Data* 8, 38 (2021). https://doi.org/10.1186/s40537-021-00426-w

13. Chapman, A., Allen, M. D., & Blaustein, B. T. (2012, June). It's About the Data: Provenance as a Tool for Assessing Data Fitness. In *TaPP*.

14. Lv, M., Dong, C., Chen, T., Zhu, T., Song, Q., & Fan, Y. (2021). A Heterogeneous Graph Learning Model for Cyber-Attack Detection. *arXiv preprint arXiv:2112.08986*.

*15. Removed.*

16. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, *32*(1), 4-24.

17. Bagui, S. S., Mink, D., Bagui, S. C., Ghosh, T., Plenkers, R., McElroy, T., ... & Shabanali, S. (2023). Introducing UWF-ZeekData22: A Comprehensive Network Traffic Dataset Based on the MITRE ATT&CK Framework. Data, 8(1), 18.