A TRIDENT SCHOLAR PROJECT REPORT

NO. 517

Competitively Learned Attention Mechanism Prototypes for Network Intrusion Detection

by

Midshipman 1/C Kade M. Heckel, USN

UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
5-16-22		

4. TITLE AND SUBTITLE
Competitively Learned Attention Mechanism Prototypes for Network Intrusion Detection

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Kade M. Heckel

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Naval Academy
Annapolis, MD 21402

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Trident Scholar Report no. 517 (2022)

12. DISTRIBUTION / AVAILABILITY STATEMENT

This document has been approved for public release; its distribution is UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Maintaining secure computer networks and information systems is critical to the administrative functions and operations carried out by the Department of Defense (DOD). Ensuring the security and functionality of these networks is vital to US National Security, however, previous work on applying neural networks to intrusion detection has focused on recurrent and convolutional neural networks but has yet to explore attention-mechanism-based architectures. Inspired by the Transformer in Vaswani et al. (2017), These attention-based models rely predominantly on layers that produce rich contextualized representations through learning pairwise interactions within sequences, enabling significant advances in computer vision and natural language processing over the last five years.
This research investigated the performance of attention-based neural network architectures compared to traditional models on the University of New Brunswick's CSE-CIC-IDS2018 dataset. Reporting precision, recall, and the Area Under the Receiver Operating Characteristic Curve (AUROC), results show that models leveraging attention mechanisms performed demonstrably better than a tuned feed-forward network on the infiltration attack class. Additionally, this work explores a novel efficient attention mechanism for deep neural networks by learning a compressed representation of the data through competitively-learned memory prototypes, showing competitive performance against an alternative efficient attention architecture that utilizes gradient descent.

15. SUBJECT TERMS
Machine Learning, Network Security, Network Intrusion Detection, Neural Networks, Cybersecurity, Attention Mechanisms

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		39	**19b. TELEPHONE NUMBER** *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

U.S.N.A. --- Trident Scholar project report; no. 517 (2022)

COMPETITIVELY LEARNED ATTENTION MECHANISM PROTOTYPES FOR NETWORK INTRUSION DETECTION

by

Midshipman 1/C Kade M. Heckel
United States Naval Academy
Annapolis, Maryland

____
(signature)

Certification of Adviser Approval

Professor Frederick L. Crabbe
Computer Science Department

____
(signature)

____
(date)

Acceptance for the Trident Scholar Committee

Professor Maria J. Schroeder
Associate Director of Midshipman Research

____
(signature)

____
(date)

USNA-1531-2

Abstract

Maintaining secure computer networks and information systems are vital to the administrative functions and operations carried out by the Department of Defense (DOD). However, the diverse ecosystem of weapons platforms from various defense contractors and large enterprise networks results in many data formats, complicating the analysis process for Cyber Protection Teams conducting Defensive Cyber Operations (DCO). Additionally, identifying an adversary's actions among the noise of everyday network behavior poses a significant challenge due to the subtle methods employed to disguise their actions.

With advances in computer architecture in the last decade enabling an explosion in deep learning, neural networks with millions or billions of parameters have emerged as powerful tools in machine learning. However, previous work on applying neural networks to intrusion detection has focused on recurrent and convolutional neural networks but has yet to explore attention-mechanism-based architectures inspired by the Transformer in Vaswani et al. (2017). These attention-based models contain layers that produce rich contextualized representations through learning pairwise interactions within data sequences, enabling tremendous advances in computer vision and natural language processing over the last five years.

This research investigated the performance of attention-based neural network architectures compared to traditional models on the University of New Brunswick's CSE-CIC-IDS2018 dataset. Evaluating models on precision, recall, and the Area Under the Receiver Operating Characteristic Curve (AUROC), results show that models leveraging attention mechanisms performed demonstrably better than a tuned feed-forward network on the infiltration attack class. Additionally, this work explores a novel attention mechanism for improving the efficiency of neural network attention mechanisms by learning a compressed representation of the data through competitively-learned memory prototypes, showing competitive performance against an alternative efficient attention architecture that utilizes gradient descent.

Keywords: Machine Learning, Network Security, Network Intrusion Detection, Neural Networks, Cybersecurity, Attention Mechanisms

Acknowledgements

I would like to thank Dr. Crabbe for his continued support and advisement of this project and for allowing me to expand my understanding and explore the field of machine learning. I additionally want to thank LtCol Eric Larsen, USMC, USNA class of 2003 and Trident Scholar, and Dr. Joan Romano Shifflett, the director of the USNA Writing Center, for their incredible mentorship and support, both academically, professionally, and personally during my time as a midshipman. Finally, I want to thank my family, especially my mother and father, for their unlimited love and support and for continuously inspiring me to strive for excellence.

Table of Contents

1. Fleet Value

Currently, the US Army and US Cyber Command operate Big Data Platform (BDP) systems that orchestrate the collection and aggregation of host and network log data into a centralized repository. Hosted on government cloud resources, BDPs permit analysts and defensive cyber operations teams to query records for evidence of network exploitation and perform other auditing actions. As a newer command, Marine Corps Forces Cyberspace Command is constructing a BDP system for their networks and investigating methods for analyzing the vast amounts of anticipated data, making research into efficient intrusion detection models such as this study valuable. Due to the volume of data and frequent encryption of packet payloads, it is common for only the metadata from communication sessions to be retained to conserve storage. Containing features such as the number of packets sent or the duration of a connection, network metadata poses a challenge for analysts to interpret due to the difficulty of identifying anomalous values amongst millions of data points. As such, the reduction of the vast amount of information for a human analyst to review is also an active interest of the DOD's Joint Artificial Intelligence Center (JAIC) through the development of machine learning models, a need which this research addresses.

2. Background on Network Intrusion Detection

Traffic between computers on a network possesses a semi-structured nature; each layer of the Transmission Control Protocol/Internet Protocol adds information to the packets of data through encapsulation. Due to the wide variety of applications and protocols possible at each layer of the TCP/IP stack, creating a consistent set of features at the packet level is challenging. Furthermore, network intrusion detection poses a significant challenge for machine learning due to the difficulty and expense of acquiring labeled training data, significant class imbalances, and the difficulty of anomaly detection when adversaries are actively trying to deceive the model and evade detection. The issue of large-scale acquisition of real-world network data stems from privacy and security concerns. Companies are unwilling to share raw data from their networks that could reveal personal information, sensitive corporate data, or vulnerabilities in their networks. An additional complicating factor of obtaining labeling traffic is the time-consuming need for analysts to pour over large amounts of real network traffic to identify a small number of attacks that could be perpetrated by either malicious adversaries. Specifically, the significant imbalance between normal traffic and the rare occurrence of malicious behavior poses a challenge for training machine learning algorithms that require a large amount of data to identify trends and patterns and is further exacerbated by the fact that malicious attacks seek to appear as similar as possible to normal network behavior.

Due to the previously outlined challenges with obtaining and utilizing real network traffic as a dataset, previous work sought to create a simulated dataset that could provide a realistic scenario for machine learning approaches to tackle. As a sizeable simulated dataset, the CSE-CIC-IDS2018 intrusion detection dataset produced by the University of New Brunswick has served as the primary benchmark for evaluating intrusion detection systems in recent years. This dataset simulates an enterprise-scale network and contains a wide variety of computer network attacks using well-known tools and consists of network traffic data between Ubuntu Linux and multiple versions of the Windows operating system. The dataset includes Packet Capture (PCAP) files of the simulated traffic and Comma Separated Value (CSV) files with 80 preselected features such as the port number and number of packets sent summarizing communication sessions between machines on the network.[1]

[1] University of New Brunswick, 2018

```
> Frame 68: 118 bytes on wire (944 bits), 118 bytes captured (944 bits)
> Ethernet II, Src: 02:ca:69:c4:6d:06 (02:ca:69:c4:6d:06), Dst: 02:93:46:70:cd:82 (02:93:46:70:cd:82)
> Internet Protocol Version 4, Src: 131.202.242.193, Dst: 172.31.69.15
> Transmission Control Protocol, Src Port: 10807, Dst Port: 22, Seq: 705, Ack: 3281, Len: 64
v SSH Protocol
    Packet Length (encrypted): f990815e
    Encrypted Packet: 48c72fd4e002a11dcd56ad3273054dbaf0598a3568d2c84470201dab776c4c2d9b3e6e4b…
    [Direction: client-to-server]
```

Figure 1: Example ssh packet from PCAP file

	Dst Port	Protocol	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean
0	22	6	20553406	10	7	1063	1297	744	0	106.3
1	34989	6	790	2	0	848	0	848	0	424.0
2	500	17	99745913	5	0	2500	0	500	500	500.0
3	500	17	99745913	5	0	2500	0	500	500	500.0
4	500	17	89481361	6	0	3000	0	500	500	500.0

Figure 2: Sample columns from the first five instances in the CSV dataset. The dataset contains 80 total feature columns in the training and validation data. Each row is labeled as benign or one of several different attack types.

The CSV files contain labeled rows that are either benign or one of the various malicious attacks, such as denial of service (DOS), infiltration, or web attacks like SQL injection. Figure 2 shows the first few features of row 0, representing a connection between two computers on the network. For example, the destination port for row 0 is port 22, indicating a connection to the Secure Shell (SSH) service on the destination machine. During the communication described by row 0, the machine connecting to the server sent ten packets and received seven reply packets during the connection, as shown by Tot_Fwd_Pkts and Tot_Bwd_Pkts, respectively. Additional statistical metadata, such as the total or mean length of the packets sent in each direction are also calculated and included as features for models to learn on.

3. Prior Work on Deep Learning for Network Intrusion Detection

Previous work using the IDS-2018 dataset tested a variety of architectures for performing anomaly detection. Basnet et al. evaluated multiple machine learning frameworks such as TensorFlow or Pytorch on their ability to train a multilayer perceptron to act as an intrusion detection system.[2] Another comprehensive review of various network intrusion detection datasets and experimentation with various deep neural network designs can be found in Ferrag et al. This paper compared Recurrent Neural Networks, Convolutional Neural Networks, Deep Autoencoders, and several other architectures on the basis of accuracy and training time, with deep autoencoders achieving the best results.[3] While this paper provides diverse coverage of different model types and their application to the same IDS-2018 dataset that was used in this research, it is difficult to interpret the results due to the sole reporting of unbalanced accuracy values for each attack type. The use of accuracy is problematic for the task of network intrusion detection, which exhibits severe dataset imbalance. The large disparity on orders of several magnitudes in some cases complicates the interpretation and comparison between approaches based on accuracy; for example, if a model achieves 98% accuracy on a dataset where the rare target class only comprises a fraction of a percent, it is unclear if any of the rare class were classified correctly, if at all. Additionally, the specific architectures of the explored models, such as the number and composition of their layers, are not specified, inhibiting attempts at reproducing their results.

---

[2] Basnet et al., 2019
[3] Ferrag et al., 2019

Recurrent architectures, such as the recurrent Long-Short Term Memory (LSTM) model investigated in Ferrag et. al.[4], struggle with adequately representing increasingly long sequences of information. Another limitation of recurrent architectures lies in their lack of parallelism, reducing training speed, and limiting scalability to larger datasets. Another model tested in their work was a Convolutional Neural Network (CNN), which learns a set of filters that detect patterns in the provided input. Kim et al. is another paper that uses CNNs to classify specifically the Denial of Service attack category, reporting accuracy results for different network depths and various kernel dimensions where the kernel is a sliding window that defines the neighborhood of features under consideration. The work of Kim et al. provides greater detail on an investigated convolutional architecture, including the construction of pseudo-images by reshaping the input into arrays of dimensions 13×6 from 78 of the available features in the IDS-2018 dataset.[5] It is worth noting that by selecting relatively arbitrary dimensions to reshape the input data, the model performance becomes sensitive to the column ordering of the original dataset and raises the question of whether different two-dimensional configurations or one-dimensional convolutional designs may have performed better. Ideally, a model's performance would be agnostic to the ordering of the features rather than imposing a spatial relationship where none existed before.

Besides the various neural network architectures tested, the technique of Synthetic Minority Oversampling Technique (SMOTE) was investigated in combination with non-neural network machine learning approaches for dealing with the significant class imbalance in Karatas et. al.[6] This same oversampling approach was applied to the web attack category when training models, as the original class distribution for that class is approximately 2,000:1.

---

[4] Ferrag et al., 2019
[5] Kim et al., 2020
[6] Karatas et al., 2020

4. Prior Work on Neural Network Attention Mechanisms

It is noteworthy that the Transformer network architecture has not been applied to the task of network intrusion detection despite the tremendous expressive power that has led it to dominate the fields of natural language processing and computer vision as of late. Introduced in the paper "Attention is All You Need" published by Vaswani et al., the Transformer was designed for sequence transduction which involves tasks such as machine translation. The previous state-of-the-art in sequence modeling revolved around recurrent or convolutional neural networks that used either additive or dot-product attention mechanisms to communicate information between an encoder and decoder unit; the Transformer explored whether an architecture could be constructed purely from attention mechanisms, finding that by dropping the recurrent or convolutional layers these new sequence models were easier to parallelize and therefore faster to train while also achieving better performance on language modeling tasks.

Attention mechanisms can be intuitively explained with an example from natural language processing, the application for which they were initially developed. An English sentence can be broken apart into words which are then converted into a numerical representation referred to as an embedding vector. One way to make predictions about these words would be to use a dense layer, which learns from its inputs by stochastically optimizing a set of weights to transform the input into a latent space; in terms of processing a sentence, each word would always translate to the same representation or embedding in the learned latent space, which is a vector space where each dimension represents some inferred relationship and proximity between points serves as a measure of similarity. To improve on the utility of dense layers, dot-product attention mechanisms learn a latent representation labeled as the Value vector V and learn an additional set of Query and Key vectors Q and K to capture the pairwise and not necessarily reciprocal relationships between words in a sentence. These Query and Key vectors allow the model to learn relationships between the inputs in a separate representation space to create numerical scores for producing a weighted linear combination of the Value vectors that captures important information about the context of each input.

The Transformer architecture, which popularized attention mechanisms, leverages scaled dot-product attention, where the batch of inputs X is of length n where each element X_i is a vector with a dimension of D. The input is projected into separate Query, Key, and Value vectors $Q,\ K,$ and V through a linear transformation with learned weight matrices W^Q, W^K, and W^V of dimension $D \times d$, where d is the dimension of the vectors used in the attention calculations. W^Q, W^K, and W^V are determined by randomly initializing them and then optimizing them through stochastic gradient descent and backpropagation, where the error of the network is used to modify the parameters of the network. These projections are of the form $Q = XW^Q,\ K = XW^K,\ and\ V = XW^V$.[7] These matrices are visually depicted in figure 3.

[7] Vaswani et al., 2017

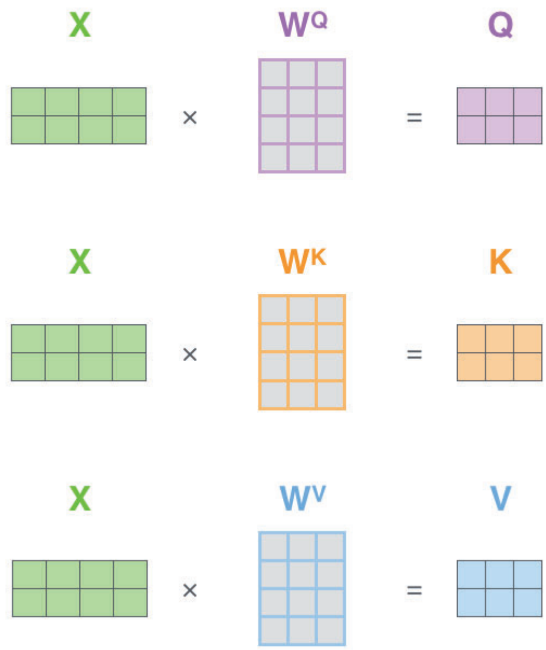

Figure 3: Visualization of multiplying input vectors against each of the three weight matrices to extract the Query, Key, and Value vectors.[8]

The previous diagram depicts the input matrix X containing two 4-dimensional vectors being multiplied against the weight matrices. In this case, the weight matrices are three columns wide, resulting in an embedding dimension of 3 for the Q, K, and V vectors. The Q, K, and V vectors used in dot-product attention each serve a distinct purpose: the Query and Key vectors allow the network to learn the asymmetric relationships between inputs and their surrounding context to produce a score that quantifies the strength of the relation between two inputs. These scores between a single input and other inputs are used to build a linear combination that encodes the context of the sequence into the newly learned embedding. The purpose for this design is to learn relationships within the input without the drawbacks of recurrent or convolutional networks, such as the loss of information over long sequences in the former or the brittle pattern

[8] Alammar, 2020

recognition induced by the learned filters in the latter. The output representation vectors Z from the self-attention layer are calculated using the equation $Z = softmax\left(\frac{Q \bullet K^{T}}{\sqrt{d_k}}\right) \bullet V$, illustrated below in figure 4:

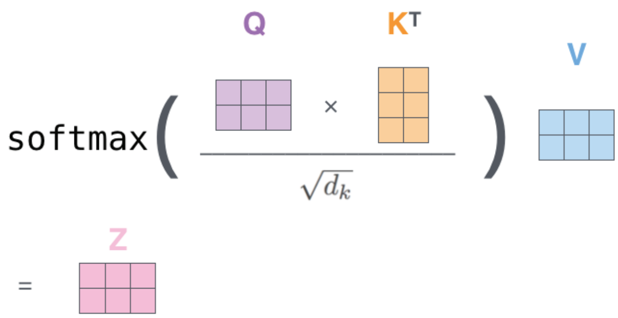

Figure 4: Matrix implementation of dot-product attention.[9]

The softmax function is as depicted in figure 5:

$$f_i(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Figure 5: The Softmax equation[10]

A step-by-step visualization of these calculations can be found on the following page in figure 6, with an example calculating an attention layer's output when comparing the words "Naval" and "Academy." In network intrusion detection, the process is identical except that the green embedding vectors are the outputs of a sparse autoencoder network which are representations of connections between computers described by rows of the IDS-2018 dataset.

[9] Alammar, 2020
[10] Kumar, 2020

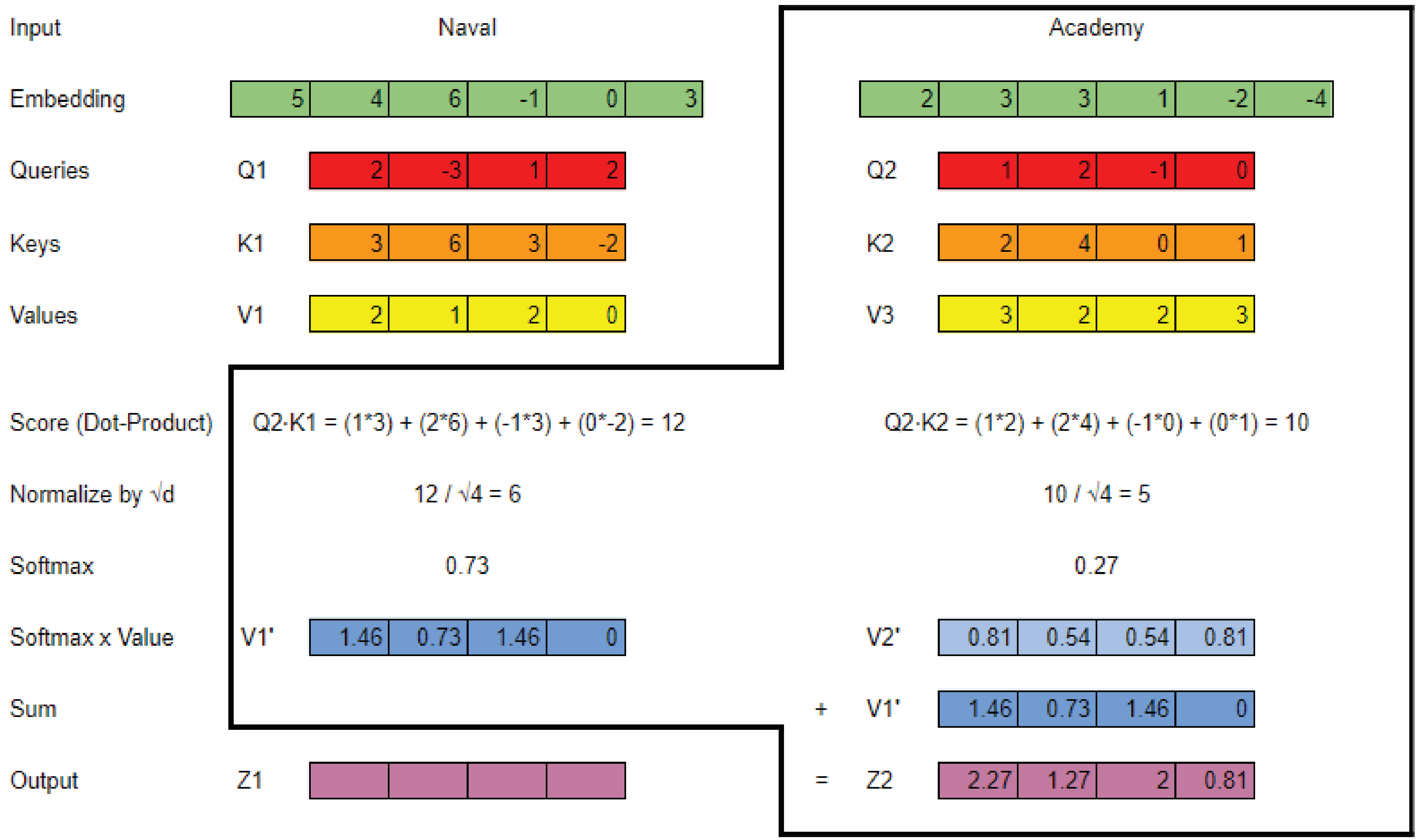

Figure 6: Attention Calculation Example

The dot product of the Query and Key vectors are normalized against the square root of the embedding dimension d to avoid saturating the softmax function and producing extremely small gradients. Without normalizing the product of Q and V, the largest resulting value would dominate the output of the softmax function with a value close to 1. At the same time, the other inputs in the sequence would receive low scores and therefore fail to capture a contextualized representation. The attention score resulting from the softmax function is then used to scale the Value vectors within the segment of input being processed. These scaled Value vectors are then summed together, yielding the contextualized embedding for a given feature by creating linear combinations from the surrounding Value vectors. The transformer block layers from Vaswani et al. use a multi-headed version of this attention mechanism where Q, K, and V are divided into smaller dimensional spaces before the V vectors are concatenated and projected back into the original embedding dimension, with the attention outputs being fed into a feed-forward dense layer with a residual connection before layer normalization is applied. The multi-head mechanism promotes each head to specialize and learn a unique attention pattern in a smaller subspace, boosting performance over the more difficult task of trying to learn a single overarching context pattern.[11]

The Transformer and pure attention architectures possess an advantage over recurrent architectures such as the LSTM through parallelization; however, an issue arises from its quadratic attention complexity when the sequence size exceeds a few hundred. As a result, research in the last four years has sought to increase the efficiency of attention mechanisms to increase the performance of Transformer style architectures. Some efforts explore optimizations or sparsification of the attention mechanism to only look at relevant sections of the input when

[11] Alammar, 2020

calculating attention scores. In contrast, other designs similar to the proposed method have investigated using a memory mechanism to extend the attention mechanism's effective context.

The $O\left(n^2\right)$ complexity of the attention mechanisms in the standard Transformer presents issues with scaling the architecture to more extensive input sequences such as entire books or images at the pixel level. Significant research has gone into reducing the computational complexity of the attention mechanism through approximation or other techniques to increase model accuracy; by having a larger context to attend over, the more information the model can leverage to make a classification. Work by Kitaev et al. leveraged locality-sensitive hashing to bucket tokens from the input sequence together before attending over the inputs, saving time and power by comparing similar vectors that contribute the most to the output once the softmax function is applied.[12] Another approach investigated in the Longformer, another efficient transformer architecture, sought to construct a sparse attention mask that avoided looking at every token while still approximating full self-attention and maintaining accuracy. This approach introduces issues when accelerators lack support for efficient sparse matrix multiplication and require the stacking of multiple layers to achieve better performance.[13]

One method for reducing the complexity of the attention mechanism is to create an information bottleneck that learns high-level information about the dataset. The Set Transformer achieves this by learning a set of parameters called "inducing points" that are trainable vectors. The Set Transformer contains a module known as the Induced Self Attention Block, which uses the input to query to the inducing points, which are initially the key and value data, before using the resulting output vectors to query against the input again. This structure is reminiscent of an autoencoder, where each input attends to a vector that converges towards a cluster center in the

[12] Kitaev et al. 2021
[13] Tay et al., 2020

input distribution throughout training. A diagram of the Set Transformer is presented in figure 7 below:

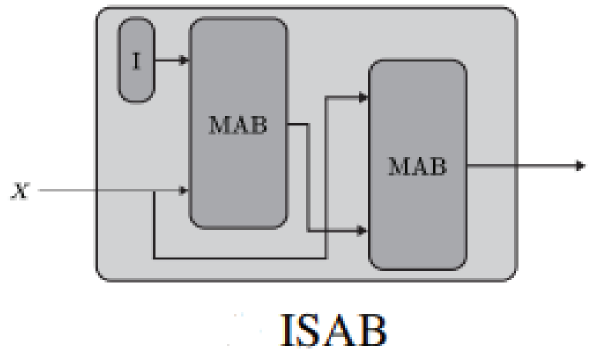

Figure 7: The Induced Self Attention Block. The learned inducing points are indicated by the input I in the top left corner of the shaded block.[14]

The Transformer-XL architecture builds on the classic transformer differently than the sparse attention mechanisms previously discussed. The vanilla Transformer architecture breaks the input sequence into segments of a few hundred instances which are then processed individually. This approach limits the context for learning patterns in the input sequence to only the immediate area and prevents the discovery of long-range dependencies such as those found in computer network traffic. Transformer-XL addresses this issue by utilizing recurrence, where the activations from the previous segment of input are cached for use in computing the current input segment's representation. Increasing the depth of the Transformer-XL network results in a linear increase in the reach-back of the model to previous segments of input for context.[15] Taking this concept a step further, the Compressive Transformer both caches activations from the previous sequences and condenses them into compressed memory vectors for long-term memory. Various

[14] Lee et al., 2019
[15] Yang et al., 2019

methods for compressing the prior representations into a memory vector exist, with options including mean-pooling where the previous values are averaged into a single vector or through a learnable function that seeks to reconstruct the earlier representations from the memory vector.[16]

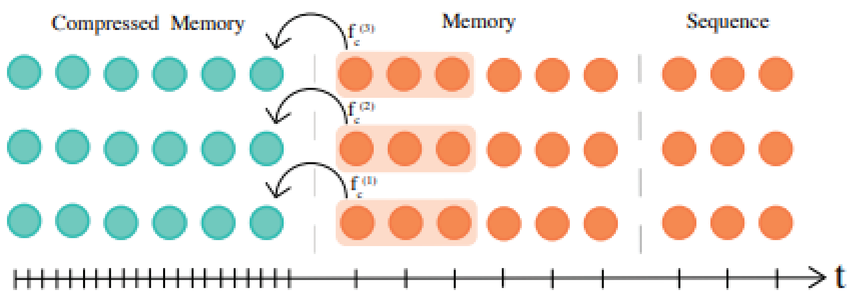

Figure 8: The Compressive Transformer's short and long-term memory. At each iteration, the oldest sequence in the short-term memory is compressed to a single long-term memory vector.[17]

Recently published as another approach to using memory compression techniques in pursuing sub-quadratic global attention, the Infinity-Transformer uses a continuous attention mechanism to extend the network's context. Rather than caching the discrete states of the previous segment of input or learning to compress a set of previous segments into a single vector in a memory set, the infinite memory transformer approximates its inputs using basis functions ranging in value from [0,1] to construct a continuous signal with a channel for each dimension of the embedding space. At each evaluation of the attention mechanism, the new inputs are concatenated to samples from the memory signal and contracted to span from [0,1] on the time axis. This forms a new discrete signal converted back into a continuous signal using multivariate ridge regression.

[16] Rae et al., 2021
[17] Rae et al., 2021

This approach does allow for attending over the entire input batch regardless of length in sub-quadratic computational complexity, unlike the compressive transformer, which is still limited to a multiple of the long-term memory's size. However, performance is still limited due to the lossy compression of the long-term memory signal when concatenating new inputs; if the frequency of the memory signal exceeds the Nyquist frequency supported by the model's fixed number of basis functions, performance may degrade. This concern of aliasing the long-term memory signal was addressed through "Sticky Memories," where critical sections of the signal are sampled at higher rates to reduce information loss in the contraction and concatenation process.[18] An overview of the Infinity-Former continuous attention mechanism is detailed in figure 9.

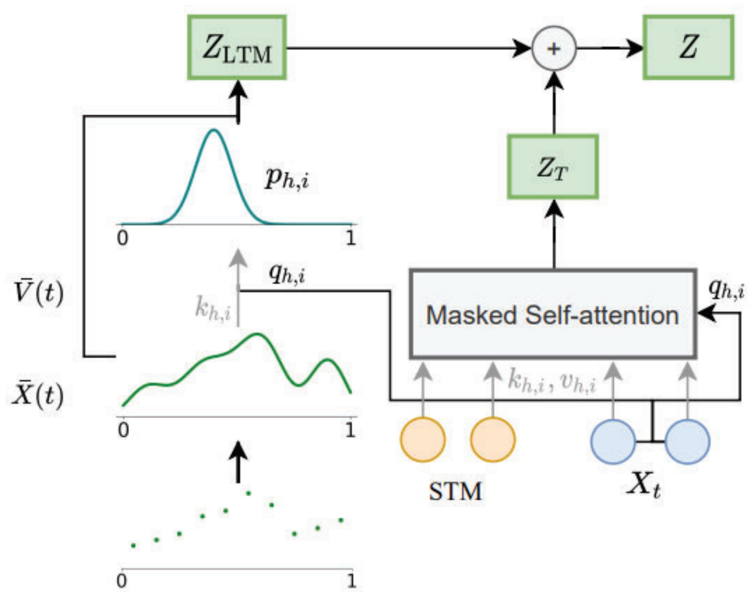

Figure 9: Continuous attention mechanism of Infinity-Former[19]

[18] Martins et al., 2021
[19] Ibid.

Figure 10 demonstrates how the current input segment X is used to query both the discrete short-term memory as well as the continuous long-term memory signal, the resulting values of which are mixed to create the resulting combined output vector denoted as *Z* which contains both local and global contextual information.[20]

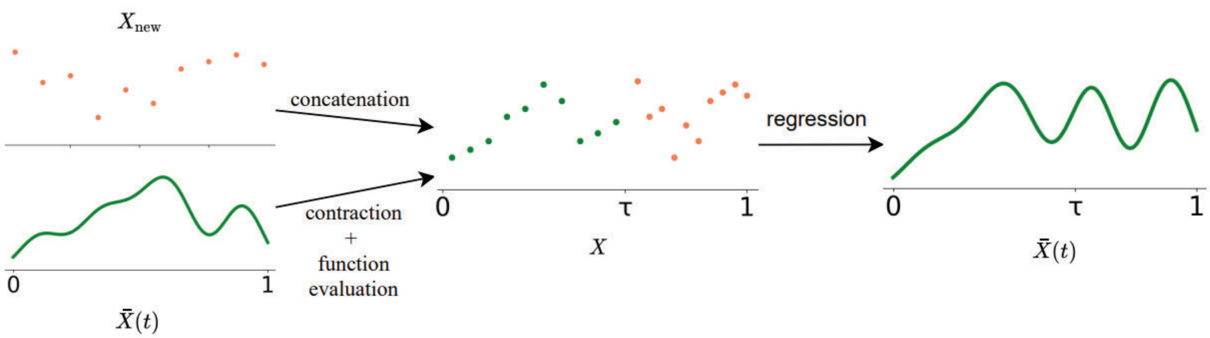

Figure 10: Attention signal contraction and concatenation to append recent inputs to memory[21]

The long-term memory contraction and concatenation process depicted above shows how information from the current input sequence is added to the long term memory, allowing for the modeling of a theoretically infinite input stream with a computational complexity tied to the number of basis functions which sample the continuous memory signal.

The most closely related and relevant work to the proposed method is the Self-Attention and Intersample Attention Transformer, which applied attention mechanisms to tabular data formatted similarly to the IDS-2018 dataset. The work by Somepalli et al.[22] advanced several interesting approaches, including contrastive pretraining on tabular datasets as well as a novel intersample/row attention mechanism of the same structure as outlined in this research. An interesting conclusion of their research is that while the intersample attention mechanism did

[20] Ibid.
[21] Ibid.
[22] Somepalli et al., 2021

boost the performance of the model, they concluded that varying batch size had little impact on performance; this finding contradicts the hypothesis of the proposed architecture that seeks to apply efficient attention mechanisms between the inputs and learned prototypes of the larger dataset to increase the effective context. A diagram of the alternating pattern of self-attention blocks learns relationships between features in a column, and intersample blocks that calculate relationships between different rows in the dataset is shown in figure 11.

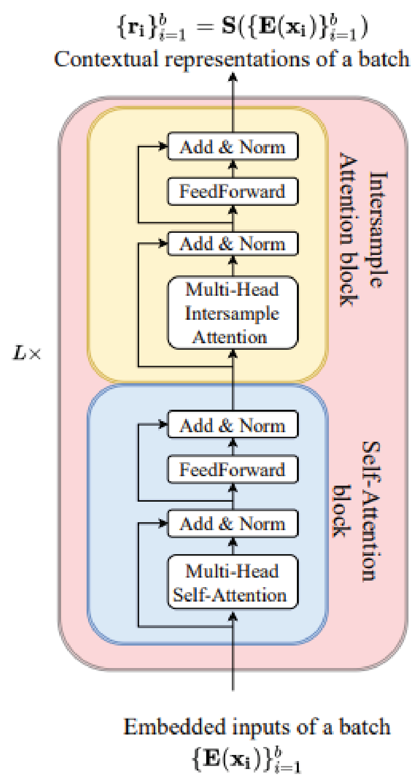

Figure 11: Intersample Attention Block from SAINT[23]

[23] Ibid.

5. Competitive Memory Transformer

This research proposes the Competitive Memory Transformer, taking a different approach to learning memory vectors for use in extending an attention model's effective context window. Instead of learning how to compress previous input segment representations into vectors using a reconstructive loss as explored by the Compressive Transformer, or iteratively contracting and sampling a continuous function in the case of the Infinity-Former, the proposed method uses a set of competitively learned vectors to model the input space. Compared to the most similar work, the Set Transformer with its Induced Self Attention Blocks, the Competitive Memory Transformer utilizes a Growing Neural Gas (GNG) network to model the topology of the input distribution over time by adjusting specialized prototype vectors rather than using gradient descent to move inducing points toward cluster centers. The prototype vectors produced by the GNG directly replace the inducing points in the ISAB architecture, with the added capability of being able to be learned in an unsupervised fashion.[24] This adaptive behavior is exemplified in figure 12:

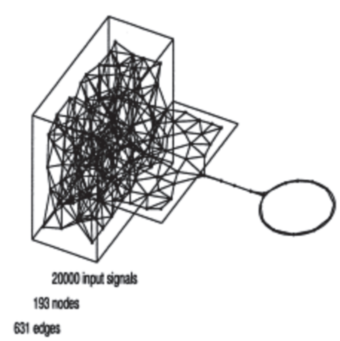

Figure 12: The GNG algorithm models the input distribution's topology in a reduced number of points, performing compression.[25]

[24] Marsland et al., 2002
[25] Ibid.

Since the Neural Gas Attention mechanism directly substitutes the inducing points of the Induced Self Attention Block examined in the Set Transformer, it similarly enjoys the same guarantees for computational complexity during prediction. By performing the attention process between a batch of n network connections and a small fixed number of k prototype vectors learned by the neural gas algorithm, the Neural Gas Attention mechanism achieves a desirable $O(kn)$ complexity which is linear with respect to the batch size, an improvement over the $O\left(n^2\right)$ complexity of traditional scaled dot-product attention mechanisms.

In this work, two different variations of the Competitive Memory Transformer were tested: a graph neural network (GNN) version and a Non-GNN version. The Non-GNN version features a custom neural gas layer that learns competitively as the rest of the network is trained using gradient descent, with the position of each prototype vector updated based on its responses to the different inputs in a batch. In contrast, the GNN version learned a static set of prototype vectors to model the input distribution of the training set before the resulting graph and edges were used as fixed parameters in the model. In both variants of the Competitive Memory Transformer architecture, the learned prototypes were used in place of the analogous inducing points. Code listings for each of these models can be found in appendix A.

In figure 13, it can be seen that the only difference between the Neural Gas Attention and the Induced Self Attention mechanism is that the Inducing points which are learned via stochastic gradient descent are now replaced with competitively learned prototype vectors from a neural gas layer. One such advantage of this neural gas layer is that it can adapt to new inputs as it makes predictions without requiring labeled data. Since it can learn unsupervised, there is no need to calculate errors and execute the backpropagation algorithm on the neural network's parameters. This property could pose especially useful for the online training of transformer-style models that are significantly deeper in layers than the models tested in this work.

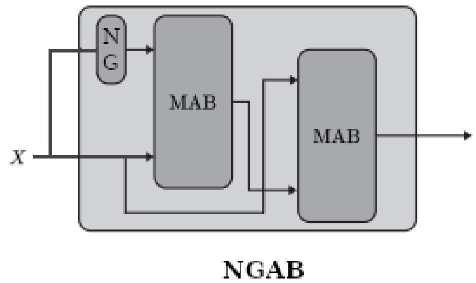

Figure 13: Neural Gas Attention Block

Figure 14 depicts the graph neural network variant of the Neural Gas Attention block, where a set of prototypes are pre-trained in an unsupervised fashion before being supplied to the model as a fixed input. The representations of these fixed prototypes are then transformed at each successive layer using a graph convolution layer; in the experiments conducted for this work, the graph attention network layer from Veličković et al. was employed, although other types of graph layers are equally valid choices.[26] The models constructed using the graph neural network version of neural gas attention did not utilize a sparse autoencoder to embed the connections but still achieved similar performance as demonstrated in the results section.

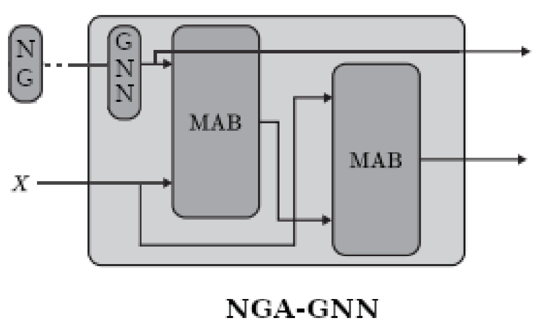

Figure 14: Neural Gas Attention - Graph Neural Network variant

[26] Veličković et al., 2017

6. Experimental Methodology

Six days of traffic from the IDS-2018 Dataset were selected to provide training and testing sets for three separate categories of attacks to classify. Each day was converted from multiclass labeling into a binary task of distinguishing hostile from benign. One day from each category would be partitioned into training and validation sets for the different model architectures to learn from, with the other day acting as a blind testing set to evaluate the model's ability to generalize and identify the same types of attacks on different days. This approach for creating train/validation/test splits differs from the other approaches to evaluating deep learning models on the IDS-2018 dataset, where the training, validation, and testing data all come from the same CSV file representing one day of communications. For the web category, the technique of SMOTE was applied to the training set only to assist the training of the models, while the testing set remained at the original and highly imbalanced class ratios.

A sparse autoencoder was found to improve the performance of the tested architectures. The sparse autoencoder design used in the experiments consisted of a 32 neuron layer followed by a 384 neuron layer with L1-regularized activations with a final 64 neuron dense layer to condense the representations into a manageable latent space. These encoding layers were first trained using the mean-squared error between the original input and the network's reconstruction as the loss function to learn new representations for the 68 dimension vectors produced from each row of the dataset. The autoencoder was then used as the first four layers of each model (with the exception of the NGA-GNN variant), where supervised training then fine-tuned the weights for the different models, such as the feed-forward model or the attention architectures. The reasoning behind this approach was that the input rows from the dataset scaled to between 0 and 1 based on their minimum and maximum values yielded representations that were still difficult to use for training models. A sparse autoencoder was used to address this issue by creating a larger representation where only specific neurons would be activated, thanks to the regularization penalty that constrains the wide layer from having too many non-zero values. This structure enabled better dense representations in the 64 neuron layer to be learned, improving the downstream model performance.

This work explored four architectures: a multi-layer feed-forward neural network, an Induced Self Attention model with four encoder blocks, a network with four Neural Gas Attention encoder blocks, and a model with 2 Neural Gas-Graph Neural Network blocks. Each model was trained with the same set of hyperparameters for consistency, especially for comparing the induced self-attention architecture to the neural gas architectures. A diagram depicting the general structure of each tested model is provided in Figure 15.

The models were constructed using the TensorFlow machine learning framework developed by Google, with the Pandas, Scikit-learn, Spektral, and Neupy libraries providing additional functionality. Scikit-learn was utilized to split CSV files into training, validation, and testing datasets, which were loaded into memory using Pandas. After loading the data, TensorFlow enabled rapid prototyping and testing of various models using a graphics processor unit to accelerate the training process in conjunction with built-in implementations of different optimization and loss functions. Encoder blocks containing layers for calculating attention or carrying out the neural gas algorithm were implemented and stacked into models for testing. Detailed diagrams of two model architectures, including the dimensions of the vectors they operate on, are depicted in figures 16 and 17.

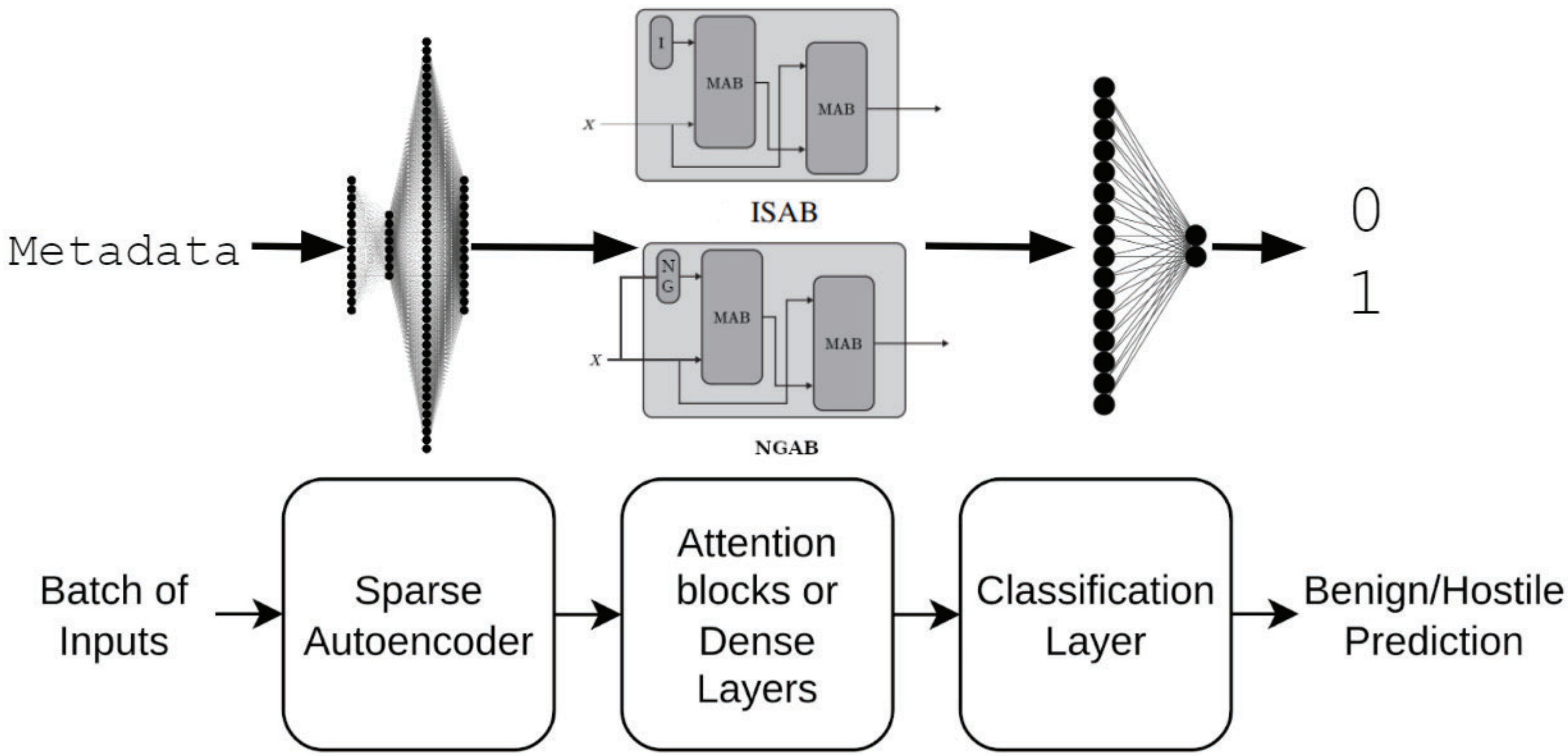

Figure 15: Overall Model Architecture. Batches from the IDS-2018 Dataset consisting of connection metadata are fed through a sparse autoencoder before a series of attention blocks or dense layers before a final classification and readout layer that provides the prediction.

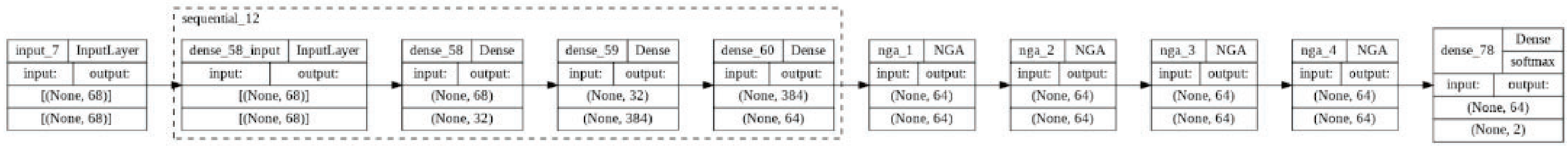

Figure 16: Neural Gas Attention Model Architecture. The dashed box indicates the sparse autoencoder used to generate embeddings of the input data.

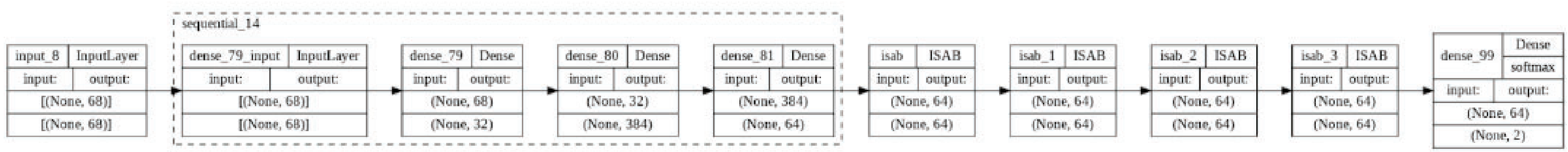

Figure 17: Induced Self Attention Architecture. Note the identical overarching structure as the NGA model to facilitate comparison

7. Results

Tasked with distinguishing malicious attacks from normal traffic, the models investigated are considered binary classifiers, meaning that they provide a yes or no response to the question "Is the metadata describing this particular communication session indicative of malicious behavior?" In order to provide a clearly interpretable set of metrics for the experiments, which are independent of the different quantities of attacks versus normal behavior across days in the dataset, standard measures, including the precision and recall performance, are reported. The table below illustrates the four categories to which the predictions from a binary classifier can be attributed to.

Confusion Matrix and Error Categories

	Non-malicious connection	**Malicious connection**
Predicted as non-malicious	**True Negative (Correct)**	**False Negative (Error)**
Predicted as malicious	**False Positive (Error)**	**True Positive (Correct)**

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad \text{(Eq. 1)}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} = \frac{True\ Positives}{Total\ Positives} = True\ Positive\ Rate \quad \text{(Eq. 2)}$$

Recall can also be termed the True Positive Rate (TPR), which has an associated metric known as the False Positive Rate (FPR):

$$False\ Positive\ Rate = \frac{False\ Positives}{False\ Positives + True\ Negatives} = \frac{False\ Positives}{Total\ Negatives} \quad \text{(Eq. 3)}$$

The Area Under the Receiver Operating Curve (AUROC) was computed to provide a single performance comparison metric between the various classifiers. The AUROC value can be interpreted as a probability of whether an attack will be properly labeled as such, meaning that models with AUROCs close to 0.5 are essentially the same as randomly guessing and have little value as a predictive tool. The curve itself represents a model's performance for varying decision thresholds, where a decision threshold is defined as the predicted probability or confidence of the model in labeling an example. A decision threshold of 1 requires the model to be 100% confident the input is an attack, resulting in neither true nor false positives as every example would be labeled as benign. Oppositely, a decision threshold of zero is depicted on the right-hand side of the ROC curve, where both the recall or true-positive rate and the false-positive rate are equal to one since the model labels every input as an attack. Figure 18 below depicts various ROC curves.

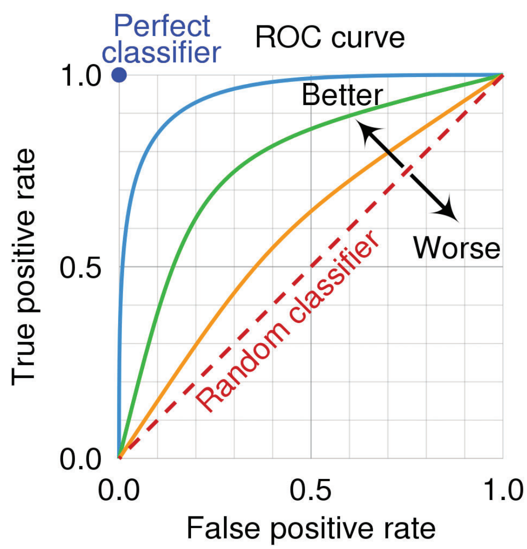

Figure 18: A visualization of the ROC Curve[27]

The hyperparameters for training the different models were as follows:

Table of Model Training Hyperparameters

Batch Size	8192
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	5 or 10

[27] CMG Lee, 2018

A batch size of 8192 was selected as the baseline for all models after experiments testing a single model's performance as the batch size was varied. The number of epochs, or the number of passes over the training dataset, was kept at a small value as the models needed few passes to fit the data before the performance began to degrade on the validation set. The Adam optimizer was used due to its robust performance in training neural networks and is standard practice across machine learning research. The following results demonstrate that different models excel for the various attack categories.

Table 1: Network Infiltration

Model	**ISAB**	**NGA**	**NGA(GNN)**	**Dense**
Precision	**0.3946**	0.2864	0.3662	0.2695
Recall	**0.2602**	0.1453	0.0700	0.5797
AUROC	**0.7266**	0.7080	0.7201	0.5117

Table 2: Web Attacks

Model	**ISAB**	**NGA**	**NGA(GNN)**	**Dense**
Precision	0.0081	0.0080	0	**0.9806**
Recall	0.3922	0.4011	1	**0.2686**
AUROC	0.9990	0.9991	0	**0.9996**

Table 3: Denial of Service Attacks

Model	**ISAB**	**NGA**	**NGA(GNN)**	**Dense**
Precision	0.9991	0.9998	**0.9998**	0.9996
Recall	0.9972	0.7439	**0.9979**	0.9996
AUROC	0.9981	0.9777	**0.9999**	0.9983

Based on the tables some inferences can be drawn about the difficulty of identifying various attack types. Table 1 provides the most exciting results, where architectures featuring attention layers markedly outperformed the feed-forward dense network. The dense network was the best architecture for web-based attacks, achieving the highest AUROC and precision, meaning that very few of the flagged attacks were false positives. One explanation for the disparity between attention models and the dense feed-forward network on the web attack category could be the employment of synthetic oversampling to the web attack training set. While balancing the training dataset provides more examples for the models to learn from, it distorts the relative composition between benign and malicious traffic of each training batch, thereby leading the attention-based models to produce a higher number of false positives predictions on the unbalanced test set compared to a feed-forward network that only considers one row at a time. In summary, it can be seen that the Denial of Service category is easy to classify regardless of architecture, that attention models outperform the dense model in the case of network infiltrations, and that severe class imbalance and oversampling challenges attention mechanisms that seek to extract information from context when making predictions.

8. Future Direction

Future work investigating the application of attention-based neural network architectures should seek to construct a labeled version of the IDS-2018 dataset that involves the packet contents alongside the metadata already extracted in the CSV files. Creating a dataset containing packet payloads as a feature would enable the application of attention at the byte level to extract information from the rich structure of network traffic packets. The models evaluated in this research were tasked with trying to identify attacks such as SQL injection only using features such as connection duration or the number of packets sent in either direction; unless the database table leaked in the case of a SQL injection attack was alarmingly large, the characteristics of a user entering a password and logging in normally versus a malicious input would be exceedingly challenging to distinguish. However, suppose the dataset included packet contents; models such as recurrent neural networks, convolutional networks, and the attention-based models explored in this work could readily be applied to the richer data and capture patterns specific to the traffic rather than the small amount of information captured in metadata. Additionally, host-based computer logs could be labeled and included to facilitate the training of graph neural networks that could perform both node and edge classification to predict compromised hosts and hostile connections within the network.

Future work should also seek to use clear metrics such as precision, recall, and AUROC to facilitate direct comparisons between models and make the code available online to enable the reproduction of results. Currently, there are no publicly available code resources for papers dealing with the IDS-2018 dataset, making it difficult to evaluate other methods since it is often impossible to recreate the same data preprocessing pipelines outlined in other works.

These challenges represent a rare opportunity for the Department of Defense to take a leadership role in this research space. The DOD could launch a challenge for intrusion detection structured similarly to the Defense Innovation Unit's xView Challenges for computer vision and remote sensing to revitalize interest and focus on network intrusion detection research. By creating a comprehensive, realistic, and large-scale public dataset and offering prizes to top solutions, the competition would spur academic and industry teams to apply more modern techniques to the problem set and create a set of baseline approaches for other researchers to build upon in future work.

Finally, further research on the use of competitively learned attention mechanism prototypes is warranted to explore their properties and capabilities fully. Future work on this new architecture should evaluate performance on mainstream tasks and datasets such as CIFAR-100, which was also examined in the work of Lee et al. with the Set Transformer.[28]

9. Conclusion

Based on the promising results in the network infiltration category, further investigation of attention-based architectures for intrusion detection is warranted. The major constraint of the attention-based models is their data-intensive nature, requiring robust and realistic datasets to effectively model adversary behavior. The competitively-learned prototypes in the neural gas attention architectures performed comparably to the baseline model that utilized Induced Self Attention encoder blocks, showing that alternative learning methods could potentially be used to overcome scenarios where the loss function landscape is noisy and challenging for gradient-based optimization. However, to properly evaluate the merits of competitively learned attention mechanism prototypes, it will be necessary to benchmark them on more conventional tasks such as language modeling or image classification, where results from similar works are easier to reproduce thanks to published code. Future work with attention mechanisms on intrusion detection should seek to extract and label the packet payloads from IDS2018 to allow experimentation with content-based detection, which would be an asset for researching models for government cybersecurity where deep packet inspection can be implemented.

[28] Lee et al., 2021

References

Alammar, J., 2020. *The Illustrated Transformer*. [online] Jalammar.github.io. Available at: <https://jalammar.github.io/illustrated-transformer/> [Accessed 3 December 2021].

Basnet, R., Shash, R., Johnson, C. and Walgren, L., 2019. *Towards Detecting and Classifying Network Intrusion Traffic Using Deep Learning Frameworks.* [online] Available at: <https://www.researchgate.net/publication/337936440_Towards_Detecting_and_Classifying_Network_Intrusion_Traffic_Using_Deep_Learning_Frameworks> [Accessed 3 December 2021].

Ferrag, M., Maglaras, L., Moschoyiannis, S., Janicke, H., 2019. [online] Available at: <https://dora.dmu.ac.uk/bitstream/handle/2086/18792/Deep_Learning_for_Cyber_Security_Intrusion_Detection__Approaches__Datasets__and_Comparative_Study%20(1).pdf?sequence=1> [Accessed 22 April 2022]

Karatas, G., Demir, O. and Sahingoz, O., 2020. Increasing the Performance of Machine Learning-Based IDSs on an Imbalanced and Up-to-Date Dataset. *IEEE Access*, 8, pp.32150-32162.

Kim, J., Kim, J., Kim, H., Shim, M., Choi, E., 2020. *CNN-Based Network Intrusion Detection against Denial-of-Service Attacks.* [online] MDPI. Available at <https://www.mdpi.com/731956> [Accessed 12 May 2022]

Kitaev, N., Kaiser, Ł. and Levskaya, A., 2021. *Reformer: The Efficient Transformer*. [online] arXiv.org. Available at: <https://arxiv.org/abs/2001.04451> [Accessed 3 December 2021].

Kumar, A., 2020. *What's Softmax Function & Why do we need it?*. [online] Data Analytics. Available at: <https://vitalflux.com/what-softmax-function-why-needed-machine-learning/> [Accessed 3 December 2021].

Lee, CMG, MartinThoma, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S. and Teh, Y., 2019. *Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks*. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1810.00825.pdf> [Accessed 3 December 2021].

Lin, P., Ye, K. and Xu, C., 2019. *Dynamic Network Anomaly Detection System by Using Deep Learning Techniques*. [online] ResearchGate. Available at:

<https://www.researchgate.net/publication/333831984_Dynamic_Network_Anomaly_Detection_System_by_Using_Deep_Learning_Techniques> [Accessed 3 December 2021].

Marsland, S., Shapiro, J. and Nehmzow, U., 2002. *A self-organising network that grows when required*. [online] ScienceDirect. Available at: <https://www.sciencedirect.com/science/article/pii/S0893608002000783> [Accessed 3 December 2021].

Martins, P., Marinho, Z. and Martins, A., 2021. *Infinite Memory Transformer*. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/2109.00301.pdf> [Accessed 3 December 2021].

Rae, J., Potapenko, A., Jayakumar, S., Hillier, C. and Lillicrap, T., 2021. *Compressive Transformers for Long-Range Sequence Modelling*. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1911.05507.pdf> [Accessed 3 December 2021].

Somepalli, G., Goldblum, M., Scwarzschild, A., Bruss, C., Goldstein, T., 2021. *SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training.* [online] Arxiv.org. Available at: <https://arxiv.org/pdf/2106.01342.pdf> [Accessed 22 April 2022].

Tay, Y., Dehghani, M., Bahri, D. and Metzler, D., 2020. Efficient Transformers: A Survey. [online] arXiv.org. Available at: <https://arxiv.org/abs/2009.06732> [Accessed 3 December 2021].

Unb.ca. 2021. *IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*. [online] Available at: <https://www.unb.ca/cic/datasets/ids-2018.html> [Accessed 3 December 2021].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. and Kaiser, L., 2017. *Attention Is All You Need*. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1706.03762.pdf> [Accessed 3 December 2021].

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y., 2017. *Graph Attention Networks*. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1710.10903v3.pdf> [Accessed 12 May 2022].

Yang, Z. and Le, Q., 2019. *Transformer-XL: Unleashing the Potential of Attention Models*. [online] Google AI Blog. Available at: <https://ai.googleblog.com/2019/01/transformer-xl-unleashing-potential-of.html> [Accessed 3 December 2021].

Appendix A: Code Samples

```
class MAB(tf.keras.layers.Layer):
    def __init__(self, embed_dim: int, num_heads: int, ax=(0)):
        super(MAB, self).__init__()
        self.mab = tf.keras.layers.MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim, attention_axes=ax)
        self.norm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6, dtype='float32')#'float16')
        self.norm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6, dtype='float32')#16')
        self.ff1 = tf.keras.layers.Dense(embed_dim)
        self.d1 = tf.keras.layers.PReLU()
        self.ff2 = tf.keras.layers.Dense(embed_dim)

    def call(self, x, y):

        z = self.mab(x, y)
        z = self.norm1(x + z)
        r = self.ff2(self.d1(self.ff1(z)))
        return self.norm2(z + r)
```

Listing 1: Multi-head Attention Block Internals

```
class ISAB(tf.keras.layers.Layer):
    def __init__(self, embed_dim: int, num_heads: int, num_inducing_points: int):
        super(ISAB, self).__init__()
        self.mab1 = MAB(embed_dim, num_heads)
        self.mab2 = MAB(embed_dim, num_heads)
        self.inducing_points = tf.random.normal(shape=(num_inducing_points, embed_dim))


    def call(self, x):
        p = self.inducing_points
        h = self.mab1(p, x)
        return self.mab2(x, h)
```

Listing 2: Induced Self Attention Block Internals

```
class NGA(tf.keras.layers.Layer):
    def __init__(self, embed_dim: int, num_heads: int, num_protos: int):
        super(NGA, self).__init__()
        self.mab1 = MAB(embed_dim, num_heads)
        self.mab2 = MAB(embed_dim, num_heads)
        self.ng = NG(num_protos)


    def call(self, x):
        x, p = self.ng(x)
        h = self.mab1(p, x)
        return self.mab2(x, h)
```

Listing 3: Neural Gas Attention Internals

```python
class NG(tf.keras.layers.Layer):
  def __init__(self, units=16, e=1, l=.5):
    super(NG, self).__init__()
    self.W = None
    self.units = units
    self.l = l # should be a reasonable number, likely > 1...
    self.e = e

  def build(self, input_shape):
    w_init = tf.keras.initializers.RandomUniform(-1,1)
    self.W = tf.Variable(w_init(shape=(self.units, input_shape[-1]), dtype="float32"), trainable=False)

  def adapt(self, inputs):
    x = tf.expand_dims(inputs, 1)
    w = tf.expand_dims(self.W, 0)
    d = tf.math.squared_difference(x, w)
    d = tf.math.reduce_sum(d, axis=-1)
    v, k = tf.math.top_k(-d, k=self.units)
    k = tf.cast(k, dtype=tf.float32)
    alpha = self.e * tf.math.exp(-k/self.l)
    delta = x - w
    alpha = tf.expand_dims(alpha, -1)
    update = tf.math.reduce_mean(alpha * delta, axis=0)
    self.W.assign(self.W + update)

  def call(self, x, training):
    if training:
      self.adapt(x)
    return x, self.W
```

Listing 4: Competitive Neural Gas Layer

```python
class AutoEncoder(tf.keras.models.Model):
  def __init__(self):
    super(AutoEncoder, self).__init__()
    self.encoder = tf.keras.Sequential([
      tf.keras.layers.Dense(32, activation="relu"),
      tf.keras.layers.Dense(384, activation="relu", activity_regularizer=tf.keras.regularizers.L1(10e-6)),
      tf.keras.layers.Dense(64, activation="relu"),

      ])

    self.decoder = tf.keras.Sequential([
      #tf.keras.layers.Dense(32, activation="relu"),
      tf.keras.layers.Dense(68, activation="relu")])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded
```

Listing 5: Sparse AutoEncoder Architecture

```
def isab_model(E):
  inputs = tf.keras.layers.Input(shape=(68))

  x = E(inputs)
  x = ISAB(64,8,256)(x)
  x = ISAB(64,8,256)(x)
  x = ISAB(64,8,256)(x)
  x = ISAB(64,8,256)(x)

  x = tf.keras.layers.Dense(2, activation="softmax")(x)
  model = tf.keras.Model(inputs, x)

  return model
```

Listing 6: ISAB Model Architecture

```
def nga_model(E):
  inputs = tf.keras.layers.Input(shape=(68))

  x = E(inputs)
  x = NGA(64,8, 256)(x)
  x = NGA(64,8, 256)(x)
  x = NGA(64,8, 256)(x)
  x = NGA(64,8, 256)(x)

  x = tf.keras.layers.Dense(2, activation="softmax")(x)
  model = tf.keras.Model(inputs, x)

  return model
```

Listing 7: Neural Gas Model Architecture

```python
def dense_model(E):
  inputs = tf.keras.layers.Input(shape=(68))

  x = E(inputs)
  x = tf.keras.layers.PReLU()(x)
  x = tf.keras.layers.Dense(64)(x)
  x = tf.keras.layers.PReLU()(x)
  x = tf.keras.layers.Dense(32)(x)
  x = tf.keras.layers.PReLU()(x)
  x = tf.keras.layers.Dense(32)(x)
  x = tf.keras.layers.PReLU()(x)
  x = tf.keras.layers.Dense(32)(x)
  x = tf.keras.layers.PReLU()(x)
  x = tf.keras.layers.Dense(16)(x)
  x = tf.keras.layers.PReLU()(x)

  x = tf.keras.layers.Dense(2, activation="softmax")(x)
  model = tf.keras.Model(inputs, x)

  return model
```

Listing 8: Dense Feed-Forward Model Architecture

```python
def GNG(X):
  n_hidden = 68
  Gas = GrowingNeuralGas(n_start_nodes=256, max_nodes=256, n_inputs=68,
                         verbose=True, n_iter_before_neuron_added=500)
  Gas.train(X, epochs=1)
  neural_map = neupy_to_spektral(Gas, sparse=False)
  neural_map.a = spektral.utils.sparse.sp_matrix_to_sp_tensor(neural_map.a)
  gc1 = spektral.layers.GATConv(n_hidden, 4, concat_heads=False)
  gc2 = spektral.layers.GATConv(n_hidden, 4, concat_heads=False)

  inputs = tf.keras.layers.Input(shape=(68), dtype=tf.float32)
  mem = gc1([neural_map.x, neural_map.a])
  out = NGAB(68, 8)(mem, inputs)
  mem = gc2([mem, neural_map.a])
  out = NGAB(68, 8)(mem, out)

  out = tf.keras.layers.Flatten()(out)
  res = tf.keras.layers.Dense(2, activation='softmax')(out)

  model = tf.keras.Model(inputs, res)
  model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
        optimizer=tf.optimizers.Adam(learning_rate=.0001),
       metrics=['Precision', 'Recall'])

  return model
```

Listing 9: Neural Gas-Graph Neural Network Model Architecture