**Approach:**

(10 points) What did you do exactly?

The structure of the problem was characterized by a highly imbalanced dataset, with normal network activity dominating and anomalies representing a minority class. Given the significance of detecting these rare anomalies, the structure of the autoencoder model was designed to reflect the underlying data dynamics. We start the problem with 80% training dataset and 20% testing dataset. The encoder is trained on a subset of the training dataset, capturing the essential characteristics of the network traffic.

Here are the steps we structure the model:

1. Created an autoencoder with an encoder and a decoder in PyTorch.
2. Trained the autoencoder using the training dataset.
3. Extracted the encoder from the trained autoencoder.
4. Built a classifier using the extracted encoder for feature representation.
5. Trained the classifier using the features from the training set.
6. Evaluated the model on a separate test set, passing inputs through the autoencoder and then through the classifier, calculating accuracy for anomaly detection.

How did you solve the problem?

In the autoencoder model, the parts with learned parameters were primarily associated with the encoder and decoder components. These components consisted of linear layers (fully connected) and activation functions, such as ReLU, designed to transform and compress the input data. Specifically, the encoder learned to map the input features to a lower-dimensional representation, capturing essential information about normal network behavior. Conversely, the decoder learned to reconstruct the input data from this compressed representation.

The post-processing classifier, which was applied for decision-making or anomaly identification, involved a separate linear layer followed by a sigmoid activation function. This part of the model facilitated the conversion of the encoded features into decision probabilities, determining whether a given input represented normal or anomalous network activity. While the weights of the linear layer in the post-processing step were learned during training, this part of the model served more as a decision-making component than a feature extraction one.

In the quest to optimize the autoencoder model, several hyperparameters were explored and fine-tuned to enhance overall performance. The initial architecture, characterized by three linear layers in both the encoder and decoder with hidden dimensions (16, 8, 4), underwent significant adjustments. The optimized architecture featured larger hidden dimensions in the encoder (64, 32, 16) and a mirrored structure in the decoder (16, 32, 64). This architectural refinement aimed to empower the model with the capacity to capture more intricate patterns within the data. Moreover, the learning rate, a critical factor influencing training dynamics, was systematically tested at values of 0.001, 0.005, and 0.01. Through careful evaluation, a learning rate of 0.005 emerged as the most effective in striking a balance between convergence speed and stability.

Batch size, another crucial hyperparameter, was subjected to experimentation with various sizes. The optimal batch size was identified as 512, showcasing its pivotal role in facilitating efficient model training and convergence. Furthermore, the choice of optimizer played a significant role in the training process. The Adam optimizer was selected for its effectiveness in optimizing neural networks, contributing to the overall success of the model.

Why did you think it would be successful?

In our initial attempt to model the data directly, we encountered a significant challenge due to the highly imbalanced nature of the dataset. Despite achieving an impressive overall accuracy of 98.15%, the F1 score, a critical metric for evaluating model performance, remained notably low at 13%. This low F1 score suggested that the model struggled to discern and learn the essential patterns or features crucial for accurate predictions.

Recognizing the limitations of our first approach, we pivoted to employ the sliding window method. This strategic adjustment proved instrumental in addressing the imbalance issue and yielded tangible improvements, particularly in the F1 score. The sliding window method allowed for a more nuanced analysis of the data, contributing to a refined understanding of the underlying patterns. Consequently, our model exhibited enhanced performance, showcasing the importance of tailored preprocessing techniques in handling imbalanced datasets and improving overall predictive accuracy.

Is anything new in your approach?

In response to addressing the highly imbalanced nature of the dataset, we have innovatively incorporated a sliding window methodology as part of our preprocessing strategy. This approach entails carefully segmenting the dataset into distinct windows, allowing for a more balanced representation of both normal and anomaly instances. By implementing this sliding window technique, we aim to mitigate the inherent imbalance and create a more equitable distribution of data points, fostering a robust learning environment for our model. This comprehensive evaluation involves comparing the performance of our model across different preprocessing strategies to discern the most effective approach for anomaly detection.

**Experiments and Results:**

(10 points) How did you measure success?

The dataset we had is highly imbalanced. It's important to note that while the accuracy metric is commonly used, its reliability can be compromised in the presence of imbalanced datasets. For instance, if anomalies are significantly outnumbered by normal cases, a high overall accuracy may be misleading. In the context of anomaly detection, the actual detection of anomalies becomes crucial.

In this scenario, precision, recall, and F1-score metrics can provide a more nuanced evaluation of the model's performance. Precision evaluates the accuracy of positive predictions, recall assesses the model's ability to capture all actual positives, and the F1-score balances precision and recall. These metrics are particularly useful for addressing class imbalance and providing a more comprehensive understanding of the model's strengths and weaknesses.

What experiments were used?

Throughout the development of the autoencoder model for anomaly detection, a series of experiments were systematically conducted to address various challenges and enhance the model's overall performance. Given the highly imbalanced nature of the initial dataset, the effectiveness of different techniques, such as the sliding window method and subsampling, was thoroughly explored to balance the dataset and improve the model's ability to detect anomalies. Additionally, hyperparameter tuning played a crucial role in optimizing the model. The architecture of the autoencoder, including the number of layers and neurons per layer, as well as hyperparameters like the learning rate and batch size, were systematically adjusted and tested to identify the most effective configuration. The training process involved monitoring the loss function over 100 epochs, ensuring convergence and assessing the model's learning progress.

In conjunction with the autoencoder, a classifier was trained on the encoded features to perform binary classification. The training of this classifier was evaluated over multiple epochs to gauge its effectiveness in distinguishing between normal and anomalous instances. Quantitative evaluation experiments included the calculation of various metrics such as accuracy, precision, recall, and F1-score on a separate test set. These metrics provided valuable insights into the model's ability to correctly classify instances and handle class imbalances effectively. Finally, the performance of the autoencoder model was compared with other models, such as a Random Forest classifier, to assess its relative effectiveness in anomaly detection. Through these comprehensive experiments, the iterative refinement of the model aimed to overcome challenges, optimize its architecture, and achieve accurate anomaly detection.

What were the results, both quantitative and qualitative?

The quantitative result is reflected in the training loss values across the 100 epochs. The results are showed below

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall | F1-Score |
| Autoencoder | 92.54% | 68.80% | 34.16% | 45.65% |

While the quantitative results indicate a decrease in the reconstruction loss, a qualitative analysis involves inspecting the reconstructed outputs visually or through other means. The loss function used in the provided code is the Mean Squared Error (MSE) loss. The MSE loss is computed between the reconstructed outputs (outputs) generated by the autoencoder and the original inputs (inputs). It measures the average squared difference between corresponding elements of these two tensors. The criterion variable, which is used to calculate the loss, is initialized as nn.MSELoss() earlier in the code. The model then minimizes this loss during training to improve the reconstruction performance and learn a meaningful representation of the input data.

The progression from an initial loss of 0.4728 to a final loss of 0.1211 highlights the effectiveness of the Autoencoder in minimizing the dissimilarity between the original and reconstructed data. This trend suggests that the model has successfully captured meaningful representations of the input features, resulting in a more refined and optimized reconstruction of the data. Such a quantitative improvement supports the overall success of the Autoencoder in its learning objectives.

**A graph of a training loss

Description automatically generated**

Did you succeed? Did you fail? Why?

The success of the Autoencoder model can be assessed based on the performance metrics summarized in the results. While the Autoencoder achieved a high accuracy of 92.54%, indicating its ability to make correct predictions on the overall dataset, it is crucial to delve deeper into other metrics for a more comprehensive evaluation.

Precision, recall, and F1-Score provide insights into the model's performance in detecting anomalies specifically. The Autoencoder demonstrates a precision of 68.80%, indicating that when it predicts an anomaly, it is correct approximately 68.80% of the time. However, the recall is 34.16%, implying that the model may not be capturing all actual anomalies, missing a significant portion.

The F1-Score, which balances precision and recall, is 45.65%, suggesting a trade-off between correctly identifying anomalies and minimizing false positives. In summary, while the Autoencoder shows a high overall accuracy, its success in anomaly detection is tempered by a lower recall and F1-Score, indicating room for improvement in capturing true anomalies.

Potential Areas of Improvement

The success of the autoencoder model hinges on its performance in identifying anomalies during the subsequent evaluation phase. The effectiveness of anomaly detection will be assessed in conjunction with the classifier's performance.

The dataset exhibits a significant class imbalance, with the majority being normal network instances, while accurate anomaly detection remains crucial. Despite our attempt to address this through the window sliding method for balancing the dataset, an alternative strategy involves selectively omitting some normal data. This approach aims to enhance accuracy in identifying anomaly data points by mitigating the impact of the dominant normal class.

**Additional:**

15 additional points will be distributed based on:

(5 points) Appropriate use of figures / tables / visualizations. Are the ideas presented with

appropriate illustrations? Are the results presented clearly; are the important differences

illustrated?

(5 points) Overall clarity. Is the manuscript self-contained? Can a peer who has also taken Deep

Learning understands all of the points addressed above? Is sufficient detail provided?

(5 points) Finally, points will be distributed based on your understanding of how your project

relates to Deep Learning. Here are some questions to think about:

* What was the structure of your problem? How did the structure of your model reflect the structure of your problem?
* What parts of your model had learned parameters (e.g., convolution layers) and what parts did not (e.g., post-processing classifier probabilities into decisions)?
* What representations of input and output did the neural network expect? How was the data pre/post-processed?
* What was the loss function?
* Did the model overfit? How well did the approach generalize?
* What hyperparameters did the model have? How were they chosen? How did they affect performance? What optimizer was used?
* What Deep Learning framework did you use?
* What existing code or models did you start with and how did these starting points help?