ZTA DEBRIEF (2/7)

Sam

This past week I have been working on finding a suitable DCNN model for network threat detection. I found a model called Deep Packet Inspection (DPI) that is tailored to analyzing live network traffic. I am working on an implementation of nDPI, an open-source DPI toolkit.

nDPI Notes

• Can work with live network data from our network tap

• Unsupervised machine learning models using nDPI can identify new, unknown threat patterns in network traffic.

• Reinforcement learning with nDPI can optimize threat detection and response strategies in distributed networks.

Ethan

I am doing research and working on implementing an RNN/LSTM.

Working on:

- Imports
    - Ignore warnings
    - Import dataset
    - Preprocessing
    - Trainer function
        - Define hyperparameters
            - Define/train model
            - Predict/test model
            - Generate/print metrics
    - Visualization?

RNN Notes:

Architecture of Recurrent Neural Networks

1. **Basic Structure:**
    - RNNs have an internal state or hidden state that is updated at each time step.

- The hidden state at time t is a function of the input at time t and the hidden state at time $-1t-1$.

2. **Formulas:**
    - The hidden state update equation can be represented as: $h=(hhh-1+h+h)ht=f(Whhht-1+Wxhxt+bh)$, where hht is the hidden state at time t, xt is the input at time t, $hhWhh$ and $hWxh$ are weight matrices, hbh is the bias term, and f is an activation function.
    -
3. **Vanishing and Exploding Gradients:**
    - RNNs are prone to vanishing or exploding gradient problems, making it challenging to capture long-term dependencies.
    - Techniques like gradient clipping, weight initialization, and more advanced architectures like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been introduced to address these issues.
4. **Long Short-Term Memory (LSTM) Networks:**
    - LSTMs are a type of RNN designed to overcome the vanishing gradient problem.
    - They have a more complex structure with memory cells, input, forget, and output gates, allowing them to capture long-range dependencies in data.

Implementing Recurrent Neural Networks

1. **Choose a Framework:**
    - Popular deep learning frameworks like TensorFlow and PyTorch provide tools for implementing RNNs.
2. **Data Preprocessing:**
    - Sequence data often needs special preprocessing, including normalization and padding.
3. **Model Design:**
    - Define the architecture, specifying the type of RNN (vanilla, LSTM, or GRU), the number of layers, and the size of the hidden state.
4. **Training:**
    - Train the model using appropriate optimization algorithms and loss functions.
    - Monitor performance on validation data to avoid overfitting.
5. **Hyperparameter Tuning:**
    - Experiment with hyperparameters like learning rate, batch size, and architecture to optimize performance.

Implementing a Recurrent Neural Network within a Zero Trust Architecture involves considering security measures and principles to protect the neural network and associated data. Zero Trust Architecture is a security concept that assumes no implicit trust in any entity, inside or outside the organization's network.

Data Encryption:

Implement end-to-end encryption for data at rest and in transit to ensure that sensitive information, including training and model data, is securely handled.

Secure Model Deployment:

Ensure that the deployment environment for the RNN model follows Zero Trust principles. Authenticate and authorize every interaction with the model, and encrypt model weights and parameters.

Access Controls:

Apply strict access controls to limit who can access the RNN model and associated resources. Use identity verification mechanisms and adhere to the principle of least privilege.

Multi-Factor Authentication (MFA):

Enforce multi-factor authentication for accessing and managing the RNN model. This adds an extra layer of security by requiring multiple forms of identification.

Continuous Monitoring:

Implement continuous monitoring of the RNN model and its environment. This includes monitoring for unusual patterns, unauthorized access attempts, and any other potential security threats.

Network Segmentation:

Employ network segmentation to isolate the RNN model from other systems and applications. This prevents lateral movement of attackers in case one part of the system is compromised.

Behavior Analytics:

Incorporate behavior analytics to detect anomalous activities related to the RNN model. Analyze patterns of usage and trigger alerts for any deviations from the expected behavior.

Secure Training Data Handling:

Apply secure data handling practices during the training phase. Ensure that training data is anonymized if necessary and access to it is restricted to authorized personnel.

Secure APIs:

If the RNN model is exposed through APIs, secure the APIs using authentication and authorization mechanisms. Regularly audit and update API security measures.

Secure Data Exfiltration Prevention:

Implement measures to prevent unauthorized data exfiltration. Monitor and control the flow of data both into and out of the RNN model to prevent leakage of sensitive information.

Regular Security Audits and Assessments:

Conduct regular security audits and assessments of the RNN model and its infrastructure. Identify and address vulnerabilities promptly to maintain a robust security posture.

LSTM Research

LSTM

Long Short-Term Memory (LSTM) networks, a specific type of recurrent neural network (RNN) designed to overcome the vanishing gradient problem and capture long-term dependencies in sequential data:

1. **Basic LSTM Structure:**
    - LSTM networks have a more complex structure compared to traditional RNNs. They include memory cells, input gates, forget gates, and output gates, allowing them to selectively retain and update information over time.
2. **Memory Cells:**
    - The key component of LSTM is the memory cell, which can store information for long durations. The cell state (*Ct*) carries information from previous time steps.
3. **Input Gate:**
    - The input gate (*it*) regulates how much new information should be stored in the memory cell.
4. **Forget Gate:**
    - The forget gate (*ft*) decides what information from the previous cell state should be discarded.
5. **Output Gate:**
    - The output gate (*ot*) determines the next hidden state (*ht*) based on the current input and the memory cell content.
6. **LSTM Unrolling:**
    - Similar to traditional RNNs, LSTMs can be unrolled in time to visualize the flow of information through different time steps.
7. **Gating Mechanisms:**
    - The gating mechanisms in LSTMs enable them to control the flow of information, selectively updating and retaining information based on the context.
8. **Applications:**
    - LSTMs are widely used in natural language processing tasks such as language modeling, machine translation, and sentiment analysis. They are also applied in time series analysis, speech recognition, and many other sequential data tasks.
9. **Training Process:**
    - LSTMs are trained using backpropagation through time (BPTT), similar to traditional RNNs. The gradients are calculated, and the weights are updated to minimize the error.

10. **Stacked LSTMs:**
    - Multiple LSTM layers can be stacked on top of each other to form a deep LSTM network, allowing for hierarchical representation of sequential data.
11. **Variants:**
    - Various LSTM variants have been proposed, such as peephole connections and coupled forget and input gates, to enhance the capabilities of the original LSTM architecture.

Giovanni

Collaborative Machine Learning (CML) Research:

- CML is a type of machine learning that allows multiple entities to collaborate on training and improving machine learning models.

- Also referred to as federated machine learning.

- Federated machine learning is CML without centralized training data.

- The big difference is that central machine learning "moves data to the computation" and federated machine learning "moves the computation to the data."

- Currently using a research blog from Google Research that focuses on Federated Learning.

- A possible framework that we can use to implement CML is Flower, listed as "a friendly federated learning framework."

- Flower offers a great tutorial with visuals that help aid the understanding of both machine learning and federated machine learning.

- Federated machine learning allows the use of machine learning where it wasn't possible before.

- Other possible CML frameworks are NVFlare, FATE, and TensorFlow. I am still looking into these frameworks more in-depth.

Nick

This past week I have been working on finding a suitable DCNN model for network threat detection. I found a model called Deep Packet Inspection (DPI) that is tailored to analyzing live network traffic. I am working on an implementation of nDPI, an open-source DPI toolkit.

nDPI Notes

• Can work with live network data from our network tap

• Unsupervised machine learning models using nDPI can identify new, unknown threat patterns in network traffic.

• Reinforcement learning with nDPI can optimize threat detection and response strategies in distributed networks.

Justin

XGBoost (Extreme Gradient Boosting) is a machine learning algorithm that excels in classification and can easily and quickly train both binary and multiclass classification algorithms. It differs from other machine learning algorithms through its use of Decision Trees, Gradient Boosting, and Boosting rounds instead of standard machine learning processes such as epochs. The tree splits the data through binary decisions into more manageable pieces for the algorithm and the gradient boosting constructs new trees to get better results from the last session. The number of boosting rounds is the number of times this process is completed.

In our algorithm, we first take the large data set we've been working with and slim it down to just the information that we will be getting from the Zeek-Kafka live stream so that we can train our model with the same information that we will be getting in practice. We then split the data into x and y, where x is the raw data, and y is the test results on whether or not the data was an attack or not, and what type of attack it was. The attack type data is then fed into an ordinal encoder which converts the different categories of attack into numerical representations that we can train the model on. We then do something similar with the non-numeric x data by classifying it as the type "category". The next thing we do is split the data into train and test groups, with 70 percent of the data being used to train the model and then 30 percent of the data being used to test the model after training. The data is then put into a classification matrix and trained. The parameters we use when training are a multi:softprob as the objective, gpu_hist as the tree method, 15 different classifications, and 25 boosting rounds. After that, we can predict the test cases and measure the accuracy and precision of the model.