Project 2: Model Engineering

Use of Feed Forward Neural Network (FFNN), Recurrent Neural Network (RNN) and Graph Neural Network (GNN)

1 Objective

The objectives of this project is to learn how to perform modelling techniques. To this end, students need to model the same problem and data with different architectures to understand how to apply the different preprocessing techniques based on the architectures used for a classification task.

Specifically, students will:

- Learn how to preprocess categorical data using different methodologies.
- Learn how to preprocess data based on the Neural Network architectures.
- Experiment with different architectures (FFNN, RNN and GNN) using different hyperparameters, optimizers and architectures.

2 Submission Rules

- Groups consist of 3 students.
- Each group must submit a zip file containing the following:
  - A report (maximum 10 pages) describing the approach, experiments and results, including tables and plots.
  - The Juypyter notebook(s) and the code (e.g., libraries with classes and functions written by you) to solve the tasks.
    - **Best practice**: Add comments and headers (Markdown) sections to understand what you are doing. They will i) help you tomorrow to understand what you did today and ii) help us to interpret your solution correctly.
    - **The notebook needs to be executed**: code and results must be visible so that we can interpret what you have done and what the results look like.
    - **Must run**: the code must work if we need to run it again.
    - **Submission format**: Include the notebook file (`.ipynb`) and an HTML export for easier review.
- Each group must upload the zip file to the teaching portal via Moodle before the deadline.

3 Dataset: Malware Analysis Datasets: API call sequences

This dataset is part of research into the detection and classification of malware using Deep Learning. It contains 42,797 malware API call sequences and 1,079 goodware API call sequences. Each API call sequence consists of **up to** 100 non-repeated consecutive API calls

associated with the parent process, extracted from the 'calls' elements of the Cuckoo Sandbox reports. Some of the API sequence can be comade upewer API calls.

For more information on this dataset, see:

- Kaggle Dataset
- and on the paper Oliveira, Angelo; Sassi, Renato José (2019), "Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks.", TechRxiv. Preprint. at `https://doi.org/10.36227/techrxiv.10043099.v1`

The dataset is in the form of a `Json` file, where each document describing an API call sequence consists of the following fiels

- **md5hash**: MD5 hash of the example encoded with 32 byte string.
- : **api_call_sequence**: the sequence of API calls, each described by its name. (**note: this sequence can have a variable length**)
- **is_malware**: the class label coded with an integer: 0 (Goodware) or 1 (Malware).

4 Tasks

Students must create a Machine Learning pipeline based on a Feed Forward Neural Network (FFNN), a Recurrent Neural Network (RNN) and a Graph Neural Network (GNN) and perform all required steps. Each task describes the steps to be followed.

4.1 Task 1: BoW Approaches

Each sequence of processes is a string, all processes are words in a sentence –> Fit a `CountVectorizer` object from sklearn!

- How many columns do you have after using the CountVectorizer? what does this number mean?
- What does each row represent? Can you still track the order of the processes (how they were called)?
- Do you have any out of vocabulary from the test set? If yes, how many? How does CountVectorizer deal with them?
- Try to fit a classifier (your choice, shallow deep or neural network). Report **how you chose the hyperparameters** of your classifier and what the final performance was on the test set.

4.2 Task 2: Feed Forward Neural Network

Now get some statistics on the number of processes called per sample.

- Do you have the same number of calls for each sample? Is the training distribution the same as the test distribution?

- Suppose you really want to use a simple Feed Forward Neural network to solve the problem. Can a Feedforward Neural Network handle a variable number of elements? And why?

use padding in shorter sequences

- What technique do you use to get everything to a fixed size during training? What happens if you have more processes to process at test time?

- Each process is actually a string: use sequential identifiers first and then use learnable embeddings. Use a FeedForward network in both cases. Report how you chose the hyperparameters of your final model and justify your choice. Can you achieve the same results for the two alternatives (sequential identifiers and learnable embeddings)?

  Explain (report on the results, whether one training was longer/more unstable than the other, etc.)

4.3 Task 3: Recursive Neural Network (RNN)

Next, use a Recursive Neural Network (RNN) to model your problem.

- Do you still need to pad your data? If yes, how?

Yes, to create the tensors

- Do you need to truncate the test sequences? Justify your answer with an understanding of why this is or is not the case.

No

- Is there any memory advantage to using an RNN over an FFNN when processing your dataset? And why?

- Start with a simple one-directional RNN. Is your network as fast as the FFNN? If not, where do you think the time-overhead comes from?

- Train and test three variants of networks:
  - Simple one-directional RNN
  - Bi-Directional RNN
  - LSTM (choose whether you want it one-directional or bi-directional)

  Can you see differences during their training? Can you see the same performance as the FF? Report the training details, your choices of hyperparameters, the test results.

4.4 Task 4: Graph Neural Network (GNN)

Finally, use a Graph Neural Network (GNN) to model your problem. Consider each sequence of API calls as a graph.

No, because the Data class of torch.geometrics automatically manage the different lengths creating a vector concatenating all graphs and maintaining the information of the starting and ending of each graph in a different vector

- Do you still need to pad your data? If so, how?

- Do you need to truncate the testing data? Justify your answer with your understanding of why this is or is not the case.

- What is the advantage of modelling your problem with a GNN compared to FFNN and RNN? Are there any disadvantages?

- Start by creating a first simple GCN and train/test it first on the CPU and then on the GPU. How long does it take to train and test in each configuration? How is it different from previous architectures? Can you guess why?
- Finally, train and tune variations of the GNN considering different message and aggregation functions and architectures:
  - Simple GCN
  - GraphSAGE
  - and GAT

  Can you see any differences in your training? Can you obtain the same performance as with the previous architectures?