Binary similarity

Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection

Xu, Xiaojun, et al. "Neural network-based graph embedding for cross-platform binary code similarity detection." Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017.

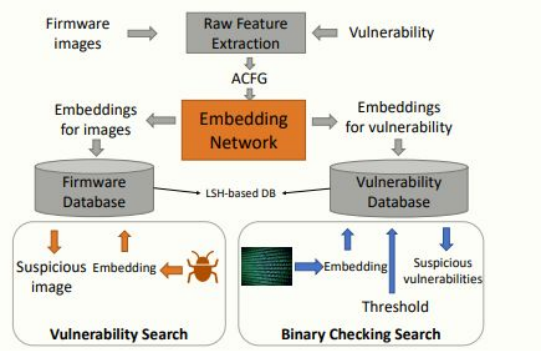


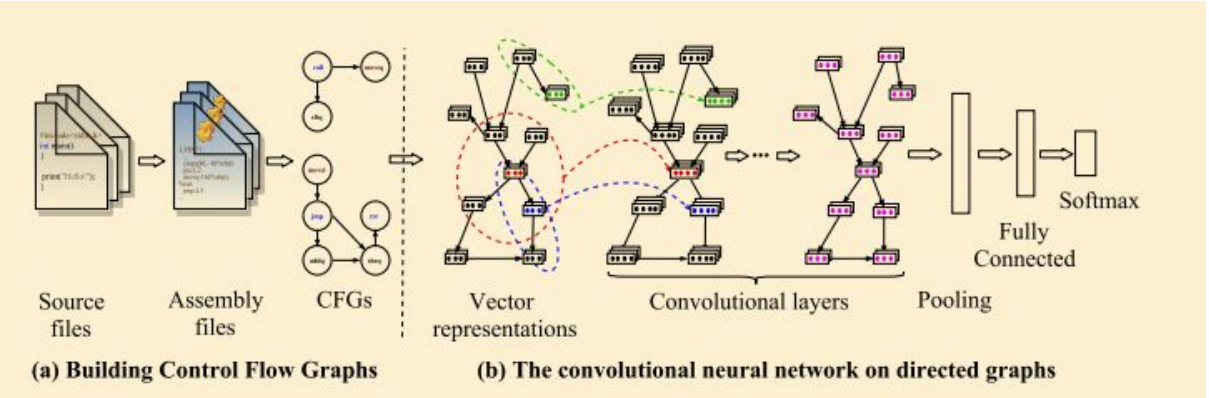
Figure 1: Cross-platform Binary Code Search Workflow

Type	Attribute name
Block-level attributes	String Constants
	Numeric Constants
	No. of Transfer Instructions
	No. of Calls
	No. of Instructions
	No. of Arithmetic Instructions
Inter-block attributes	No. of offspring
	Betweenness

ACFG + struct2vec + Siamese

Convolutional Neural Networks over Control Flow Graphs for Software Defect Prediction

Phan, Anh Viet, Minh Le Nguyen, and Lam Thu Bui. "Convolutional neural networks over control flow graphs for software defect prediction." *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2017.



cfg + DGCNN (dynamic pooling for different size cfg)

[Investigating Graph Embedding Neural Networks with Unsupervised Features Extraction for Binary Analysis]

Massarelli, Luca, et al. "Investigating Graph Embedding Neural Networks with Unsupervised Features Extraction for Binary Analysis." (2016).

acfg (extract feature + i2v) + struct2vec

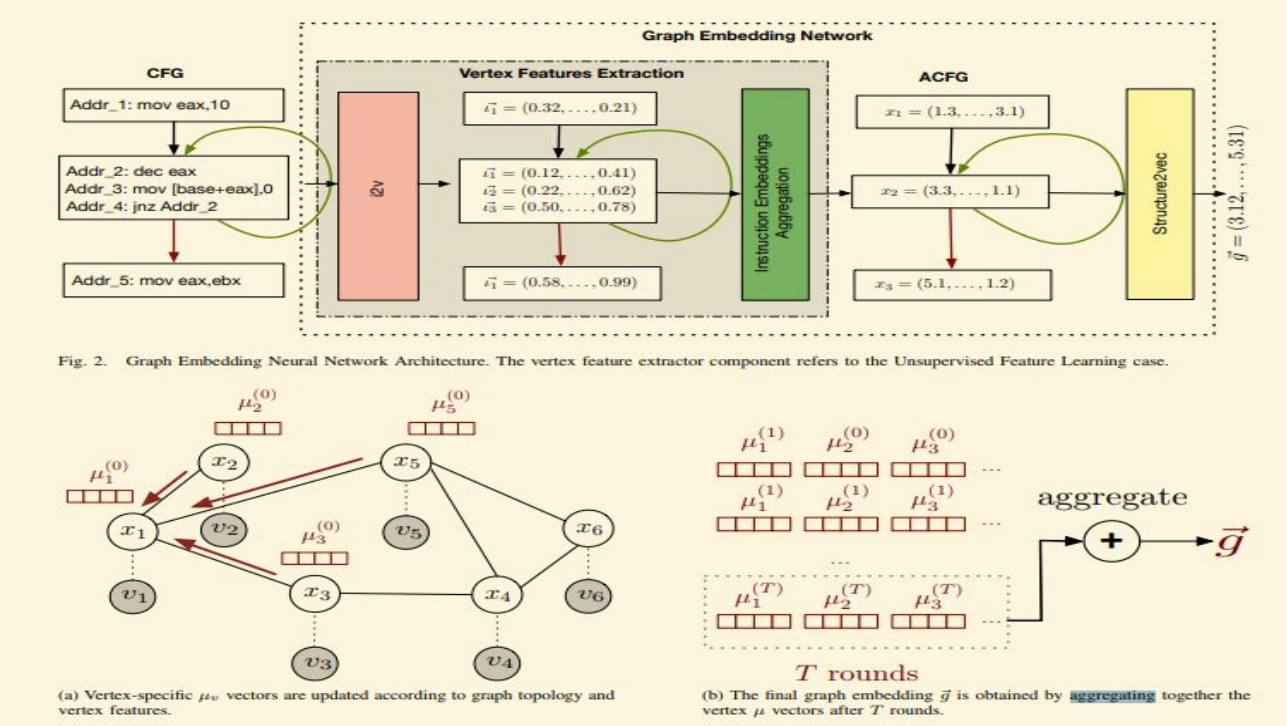


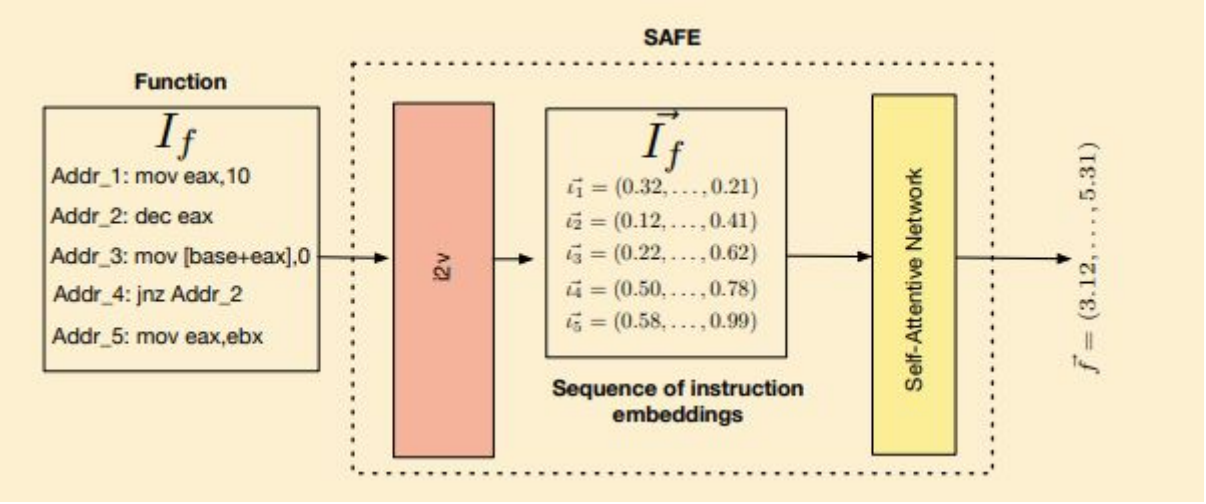
Fig. 2. Graph Embedding Neural Network Architecture. The vertex feature extractor component refers to the Unsupervised Feature Learning case.

(a) Vertex-specific μ_v vectors are updated according to graph topology and vertex features.

(b) The final graph embedding $\vec{g}$ is obtained by aggregating together the vertex μ vectors after T rounds.

[SAFE: Self-Attentive Function Embeddings for Binary Similarity]

Massarelli, Luca, et al. "SAFE: Self-Attentive Function Embeddings for Binary Similarity." *arXiv preprint arXiv:1811.05296* (2018).



basic block + i2v + attention rnn

[Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs]

Zuo, Fei, et al. "Neural machine translation inspired binary code similarity comparison beyond function pairs." *arXiv preprint arXiv:1808.04706* (2018).

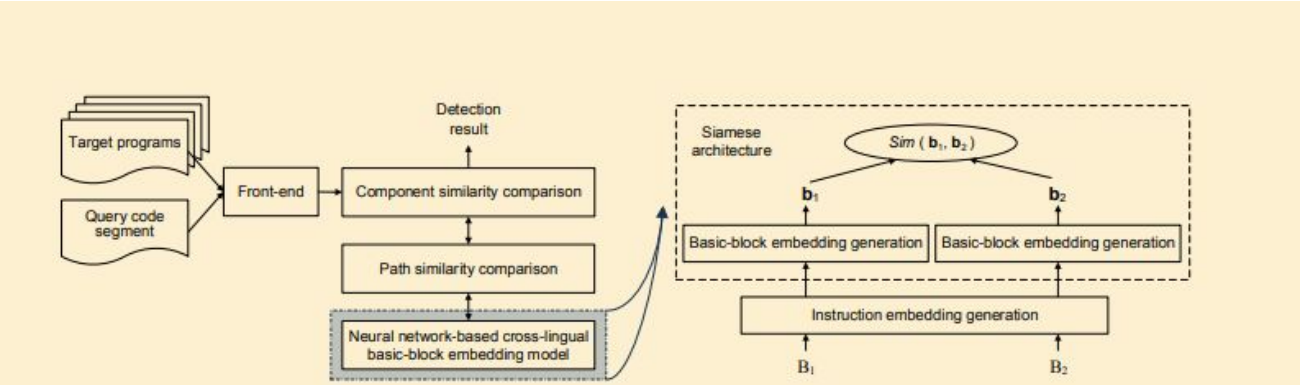
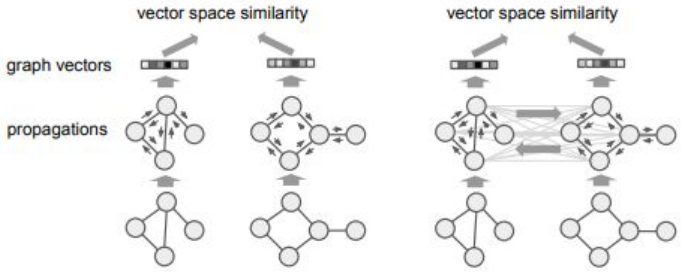


Fig. 1: System architecture.

basic block + i2v + lstm + Siamese + path similarity comparison

[Deepmind : Graph Matching Networks for Learning the Similarity of Graph Structured Objects]

Li, Yujia, et al. "Graph Matching Networks for Learning the Similarity of Graph Structured Objects." *arXiv preprint arXiv:1904.12787* (2019).



Aggregator After a certain number T rounds of propagations, an aggregator takes the set of node representations $\{\mathbf{h}_i^{(T)}\}$ as input, and computes a graph level representation $\mathbf{h}_G = f_G(\{\mathbf{h}_i^{(T)}\})$. We use the following aggregation module proposed in (Li et al., 2015),

$$\mathbf{h}_G = \mathrm{MLP}_G\left(\sum_{i\in V}\sigma(\mathrm{MLP}_{\mathrm{gate}}(\mathbf{h}_i^{(T)}))\odot\mathrm{MLP}(\mathbf{h}_i^{(T)})\right), \quad (3)$$

which transforms node representations and then uses a weighted sum with gating vectors to aggregate across nodes. The weighted sum can help filtering out irrelevant information, it is more powerful than a simple sum and also works significantly better empirically.

After the graph representations $\mathbf{h}_{G_1}$ and $\mathbf{h}_{G_2}$ are computed for the pair (G_1, G_2), we compute the similarity between them using a similarity metric in the vector space, for example the Euclidean, cosine or Hamming similarities.

Note that without the propagation layers (or with 0 propagation steps), this model becomes an instance of the Deep Set (Zaheer et al., 2017) or PointNet (Qi et al., 2017), which does computation on the individual nodes, and then pool the node representations into a representation for the whole graph. Such a model, however, ignores the structure and only treats the data as a set of independent nodes.

考慮pair的結構差異的資訊 而不是只用自己的graph 得到 embedding 然後去計算similarity

google simhash search 工具 https://github.com/googleprojectzero/functionsimsearch

Graph & Malware detection

Function call graphs versus machine learning for malware detection

Rajeswaran, Deebiga, et al. "Function call graphs versus machine learning for malware detection." *Guide to Vulnerability Analysis for Computer Networks and Systems*. Springer, Cham, 2018. 259-279.

(external function + opcode sequence) → match → HMM SVM

Mal-Netminer: Malware Classification Approach Based on Social Network Analysis of System Call Graph

Jang, Jae-wook, et al. "Mal-netminer: Malware classification approach based on social network analysis of system call graph." *Mathematical Problems in Engineering* 2015 (2015).
analysis Degree Distribution, Degree Centrality, Betweenness Centrality….useless

DLGraph: Malware Detection Using Deep Learning and Graph Embedding

Jiang, Haodi, Turki Turki, and Jason TL Wang. "DLGraph: Malware Detection Using Deep Learning and Graph Embedding." *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018.
(FCG(idx pro) + node2vec) + api call → dnn softmax

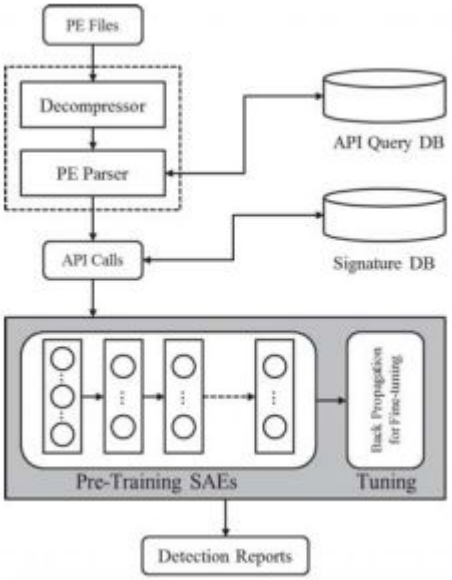
???? 看不懂
Given a computer program, we first extract the function call graph from the program. This function-call graph is then mapped to a vector in a low-dimensional feature space by using the graph embedding technique, node2vec [10]. This embedded vector has about 500 dimensions. SDA1 takes this embedded vector as input and converts it into a 50-sized vector v1. In addition, we build the API vector for the given program.The API vector has approximately 22,000 dimensions. SDA2 takes this API vector as input and also converts it into a 50-sized vector v2. We then concatenate v1 and v2 to create a 100-dimensional combined vector v. We add a softmax regression layer to our deep learning model, which classifies v to predict whether the given program is malware or not.

Analyzing, Comparing, and Detecting Emerging Malware: A Graph-based Approach

Alasmary, Hisham, et al. "Analyzing, Comparing, and Detecting Emerging Malware: A Graph-based Approach." *arXiv preprint arXiv:1902.03955* (2019).
graph properties (closeness cetrality etc) → ML classifier

DL4MD: A Deep Learning Framework for Intelligent Malware Detection

Hardy, William, et al. "DL4MD: A deep learning framework for intelligent malware detection." *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.



Malware Detection based on Graph Classification

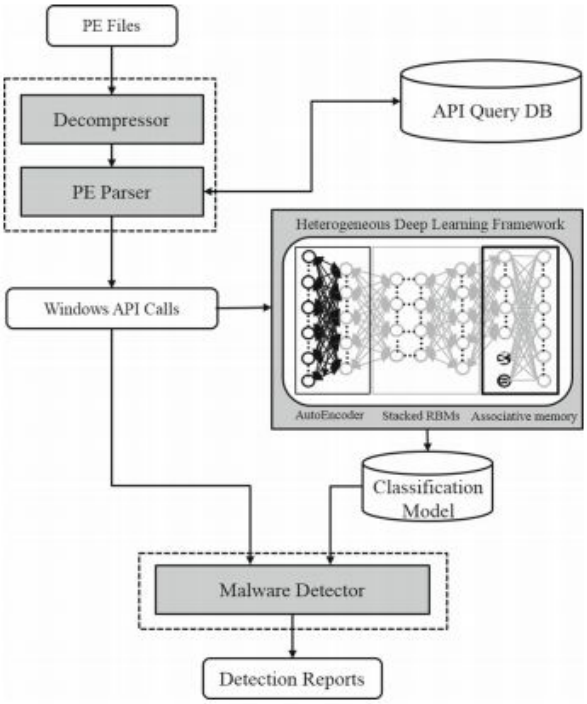
Khanh-Huu-The Dam, and Tayssir Touili. "Malware Detection based on Graph Classification." *ICISSP*. 2017.
cfg, fcg → WL kernel → SVM

Graphs for Malware Detection: The Next Frontier

Sharma, Abhishek B., and B. Aditya Prakash. "Graphs for Malware Detection: The Next Frontier."

DeepAM: a heterogeneous deep learning framework for intelligent malware detection

Ye, Yanfang, et al. "DeepAM: a heterogeneous deep learning framework for intelligent malware detection." *Knowledge and Information Systems* 54.2 (2018): 265-285.

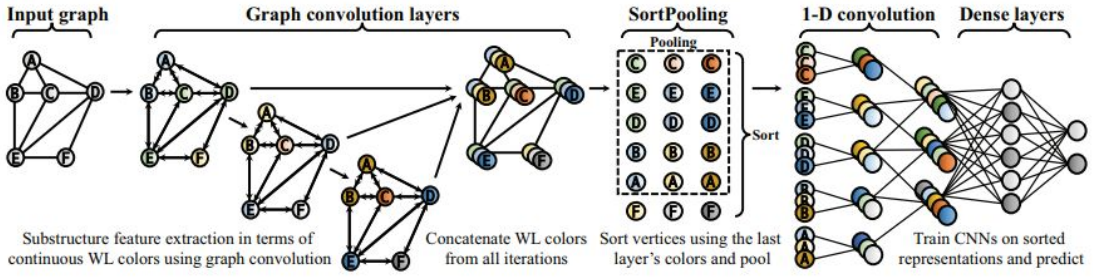


An End-to-End Deep Learning Architecture for Graph Classification

Zhang, Muhan, et al. "An end-to-end deep learning architecture for graph classification." *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

DGCNN has three sequential stages:
1) graph convolution layers extract vertices' local substructure features and define a consistent vertex ordering
2) a SortPooling layer sorts the vertex features under the previously defined order and unifies input sizes
3) traditional convolutional and dense layers read the sorted graph representations and make predictions.

To perform graph-level classification, GNNs supervise a single node, and Diffusion-CNN and Neural Graph Fingerprints use summed node features. In comparison, DGCNN sorts vertices with some order and applies traditional CNNs on the ordered representations, which keeps much more information and enables learning from global graph topology.



\

Duvenaud et al. 2015; Atwood and Towsley 2016; Niepert, Ahmed, and Kutzkov 2016; Simonovsky and Komodakis 2017; Lei et al. 2017

Proposed Method

angr angr-utils bingraphvis

用angr parse cfg

```
import angr
proj = angr.Project("a.out", load_options={'auto_load_libs':False})
cfg = proj.analyses.CFGFast(force_complete_scan=False)
from bingraphvis.angr import *
vis = Vis()
vis.set_source(AngrCFGSource())
vis.add_content(AngrCFGHead())
vis.add_content(AngrAsm(proj))
vis.preprocess(cfg.graph)
graph = vis.pipeline.process()
data = []
for n in graph.nodes:
    data.append(n.content)
```

```
pip3 install angr
git clone https://github.com/axt/bingraphvis
pip3 install -e ./bingraphvis
git clone https://github.com/axt/angr-utils
pip3 install -e ./angr-utils
```

pip3 install timeout_decorator

graph neural network

1) node embedding + aggregator (gate + weighted sum)
2) node embedding + global node
3) node embedding + global maxpolling
https://github.com/BrizziB/Graph-Classification-with-GCN
https://github.com/tkipf/gcn/issues/4
4) graph embedding (DGCNN or other)

https://github.com/nguyenlab/DGCNN