Complex Networks

Ljiljana Trajković
ljilja@cs.sfu.ca

Communication Networks Laboratory
http://www.sfu.ca/~ljilja/cnl
School of Engineering Science
Simon Fraser University, Vancouver,
British Columbia, Canada

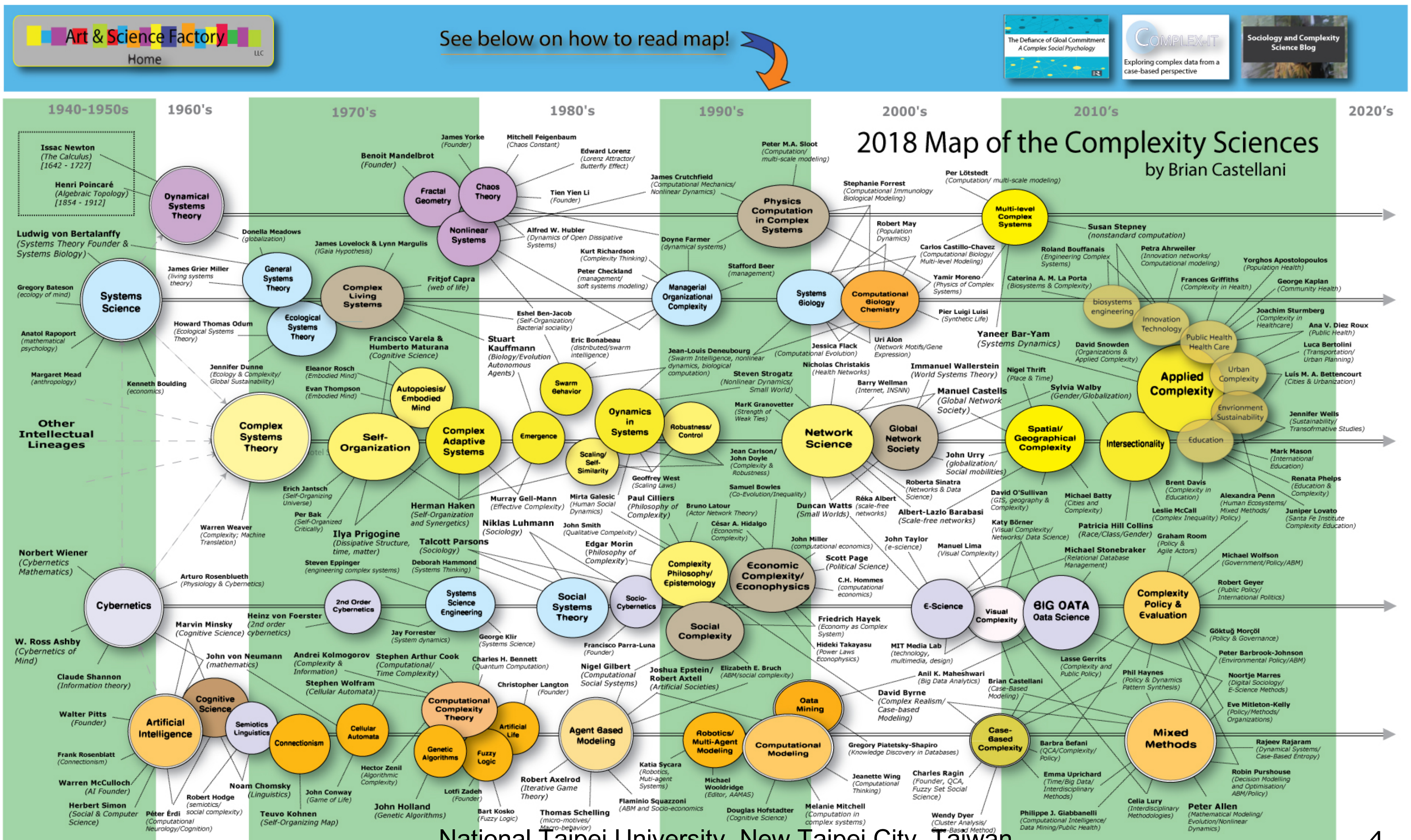
Roadmap

- Introduction
- Data processing
- Machine learning models
- Experimental procedure
- Performance evaluation
- Conclusion and References

Roadmap

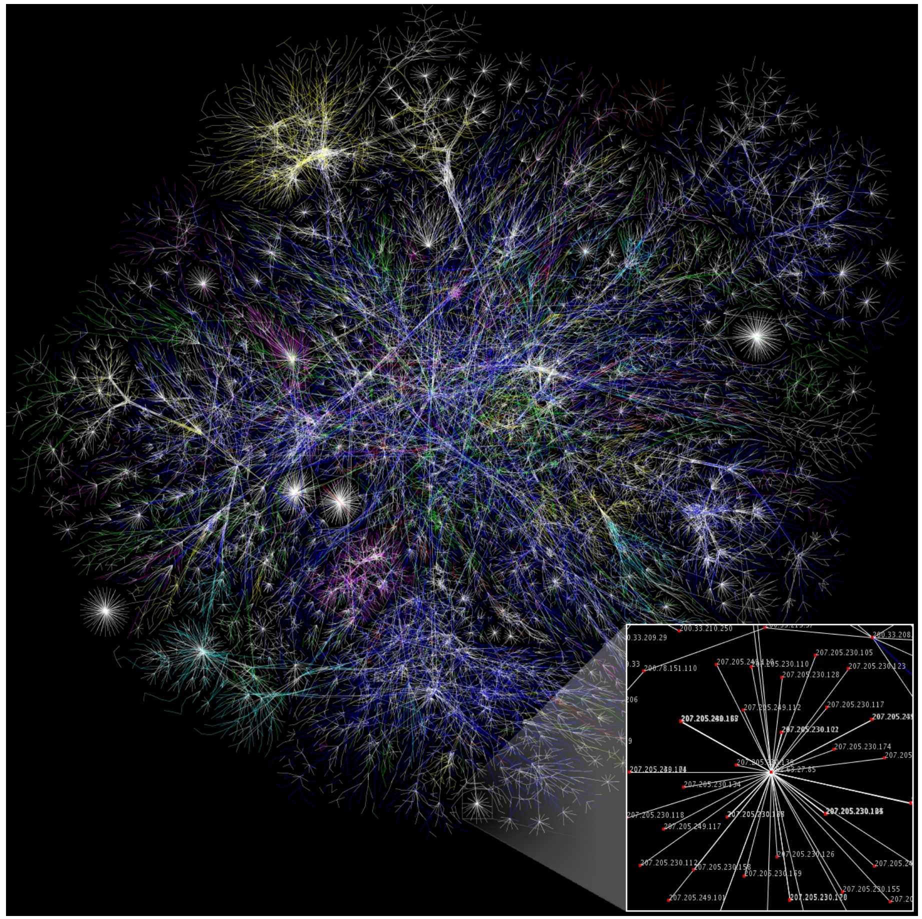
- Introduction:
  - Complex metworks
  - Machine learning
- Data processing
- Machine learning models
- Experimental procedure
- Performance evaluation
- Conclusion and References

Complex Systems



Complex Networks

The Internet



https://en.wikipedia.org/wiki/Complex_network#/media/File:Internet_map_1024.jpg
By The Opte Project - Originally from the English Wikipedia
https://commons.wikimedia.org/w/index.php?curid=1538544

The Internet

- Partial map of the Internet based on the January 15, 2005 data found on opte.org.
- Each line is drawn between two nodes, representing two IP addresses.
- The length of the lines are indicative of the delay between those two nodes.
- This graph represents less than 30% of the Class C networks reachable by the data collection program in early 2005.
- Lines are color-coded according to their corresponding RFC 1918 allocation as follows: Dark blue: net, ca, us Green: com, org Red: mil, gov, edu Yellow: jp, cn, tw, au, de Magenta: uk, it, pl, fr Gold: br, kr, nl White: unknown

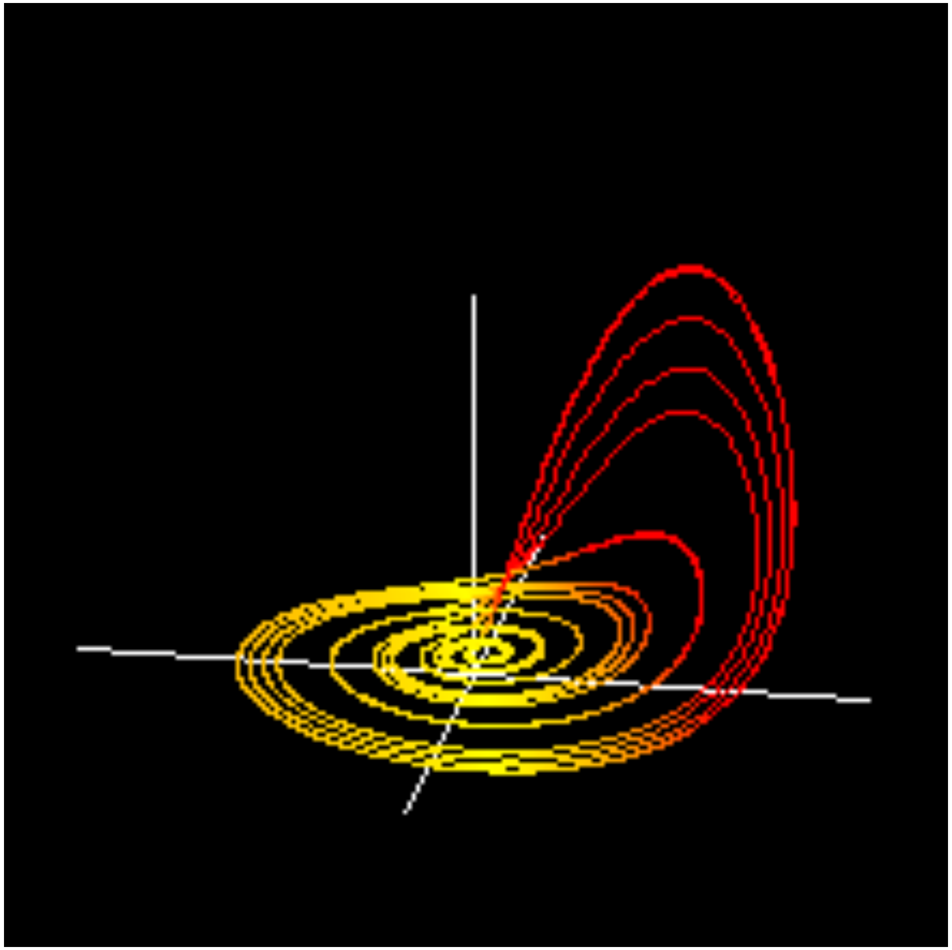
Scale-Free Networks

Scale-Free Network

- An example of complex scale-free network.
- Graph represents the metadata of thousands of archive documents, documenting the social network of hundreds of League of Nations personals.

- M. Grandjean, "La connaissance est un réseau," *Les Cahiers du Numérique,* vol. 10, no. 3, pp. 37-54.

Dynamical Systems



Dynamical Systems

- The Rössler attractor is a chaotic attractor solution to the system:

  $\dot{x} = -y - z$

  $\dot{y} = x + ay$

  $\dot{z} = b + z\,(x - c)$
- Proposed by Rössler in 1976
- Often called *Rössler system*
- Here, $(x,y,z) \in \mathbb{R}^3$ are dynamical variables defining the phase space and $(a,b,c) \in \mathbb{R}^3$ are parameters

Roadmap

- Introduction:
  - Complex networks
  - Machine learning
- Data processing
- Machine learning models
- Experimental procedure
- Performance evaluation
- Conclusion and References

Machine Learning

- Using machine learning techniques to detect network intrusions is an important topic in cybersecurity.
- Machine learning algorithms have been used to successfully classify network anomalies and intrusions.
- Supervised machine learning algorithms:
  - Support vector machine: SVM
  - Long short-term memory: LSTM
  - Gated recurrent unit: GRU
  - Broad learning system: BLS

Roadmap

- Introduction
- Data processing:
  - BGP datasets
  - NSL-KDD dataset
- Machine learning models
- Experimental procedure
- Performance evaluation
- Conclusion and References

BGP and NSL-KDD Datasets

- Used to evaluate anomaly detection and intrusion techniques
- BGP:
  - Routing records from Réseaux IP Européens (RIPE)
  - BCNET regular traffic
- NSL-KDD:
  - an improvement of the KDD’99 dataset
  - used in various intrusion detection systems (IDSs)

BGP Datasets

- Anomalous data: days of the attack
- Regular data: two days prior and two days after the attack
- 37 numerical features from BGP update messages
- Best performance: 60% for training and 40% for testing

	Regular (min)	**Anomaly (min)**	**Regular (training)**	**Anomaly (training)**	**Regular (test)**	**Anomaly (test)**
Code Red I	6,599	600	3,678	362	2,921	239
Nimda	3,678	3,521	3,677	2,123	1	1,399
Slammer	6,330	869	3,209	531	3121	339

NSL-KDD Dataset

- KDDTrain+ and KDDTest+: training and test datasets
- KDDTes$^{-21:}$ a subset of the KDDTest+ dataset that does not include records correctly classified by 21 models

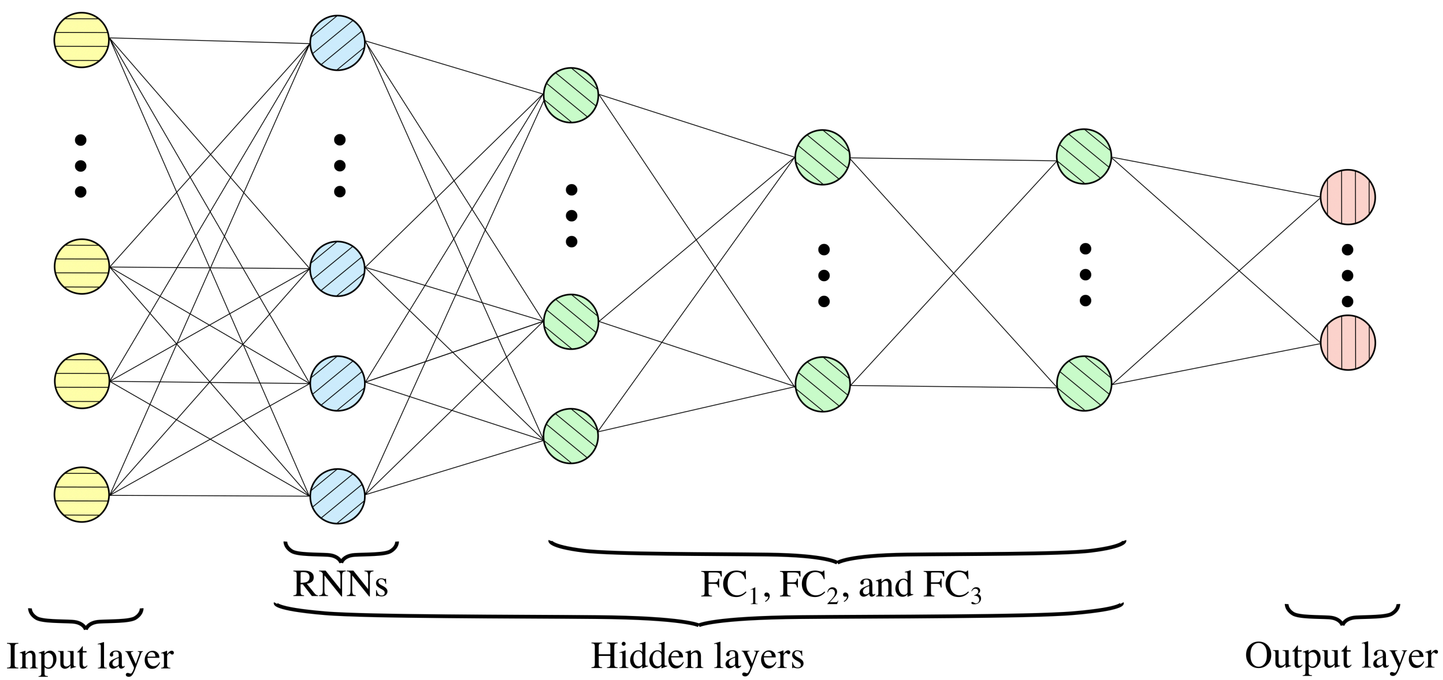
	Regular	DoS	U2R	R2L	Probe	Total
KDDTrain^{+}	67,343	45,927	52	995	11,656	125,973
KDDTest^{+}	9,711	7,458	200	2,754	2,421	22,544
KDDTest^{-21}	2,152	4,342	200	2,754	2,402	11,850

Roadmap

- Introduction
- Data processing
- Machine learning models:
  - Deep learning: multi-layer recurrent neural networks
  - Broad learning system
- Experimental procedure
- Performance evaluation
- Conclusion and References

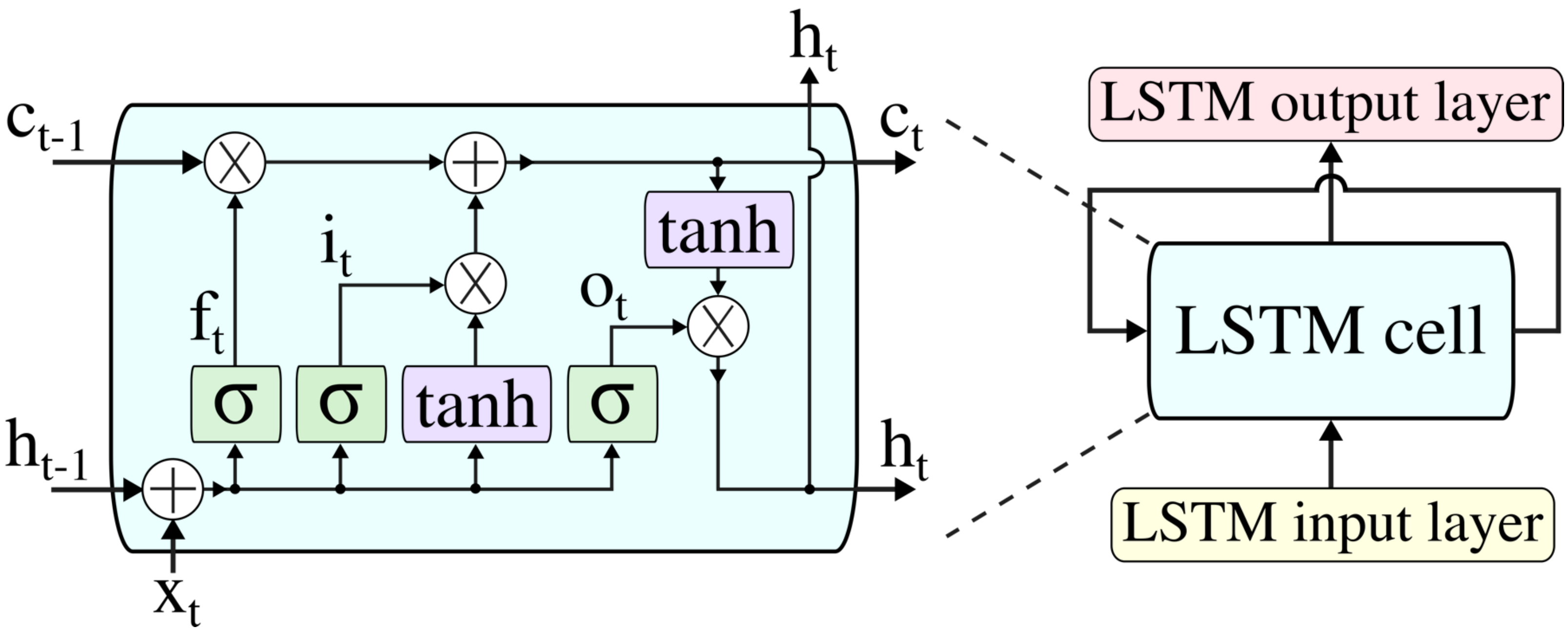
Deep Learning Neural Network

- 37 (BGP)/109 (NSL-KDD) RNNs, 80 FC_1, 32 FC_2, and 16 FC_3 fully connected (FC) hidden nodes:



Long Short-Term Memory: LSTM

- Repeating module for the Long Short-Term Memory (LSTM) neural network:



Long Short-Term Memory: LSTM

- The outputs of the forget gate f_t, the input gate i_t, and the output gate o_t at time t are:

$$f_t = \sigma\left(W_{if}x_t + b_{if} + U_{hf}h_{t-1} + b_{hf}\right)$$
$$i_t = \sigma(W_{ii}x_t + b_{ii} + U_{hi}h_{t-1} + b_{hi})$$
$$o_t = \sigma(W_{io}x_t + b_{io} + U_{ho}h_{t-1} + b_{ho}),$$

where:

$\sigma(\cdot)$: logistic sigmoid function

x_t: current input vector

h_{t-1}: previous output vector

W_{if}, U_{hf}, W_{ii}, U_{hi}, W_{io} and U_{ho}: weight matrices

b_{if}, b_{hf}, b_{ii}, b_{hi}, b_{io}, and b_{ho}: bias vectors

Long Short-Term Memory: LSTM

- Output i_t of the input gate decides if the information will be stored in the cell state. The sigmoid function is used to update the information.
- Cell state c_t:
  $c_t = f_t * c_{t-1} + i_t * tanh(W_{ic}x_t + b_{ic} + U_{hc}h_{t-1} + b_{hc})$,

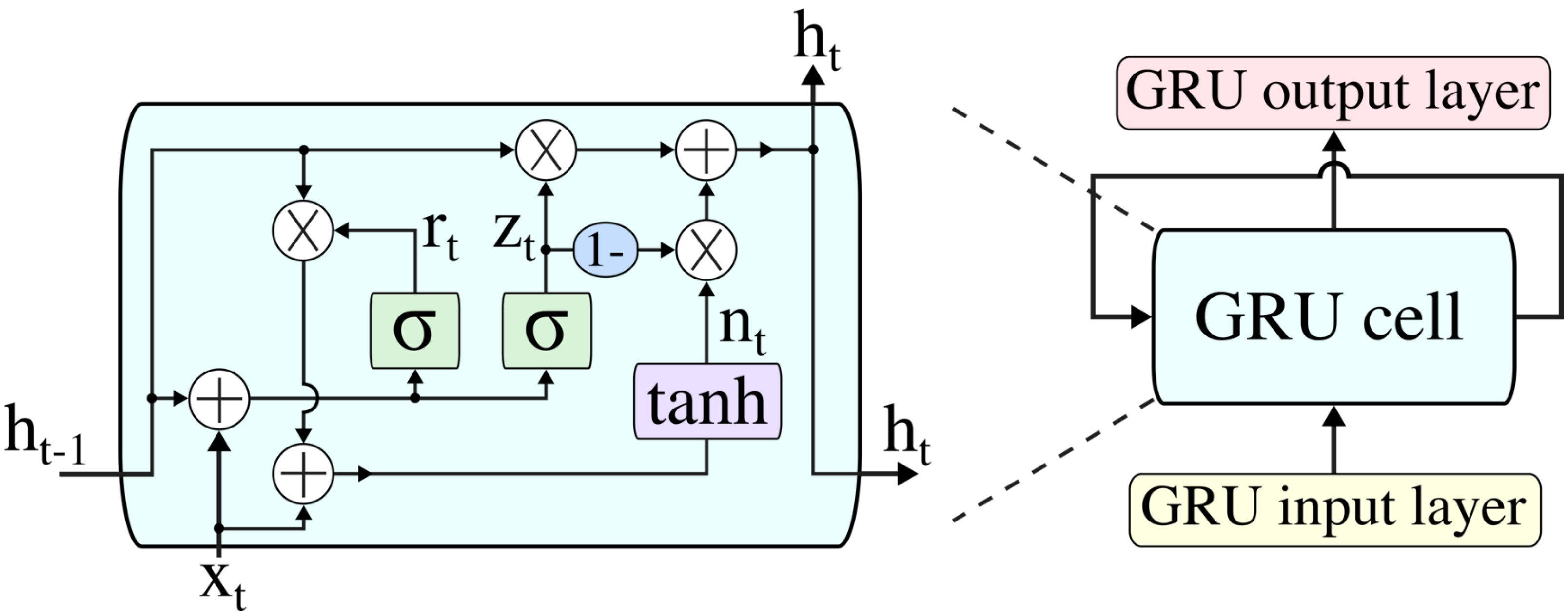
where:

  - $*$ denotes element-wise multiplications
  - $tanh$ function: used to create a vector for the next cell state
- Output of the LSTM cell:

$$h_t = o_t * tanh(c_t)$$

Gated Recurrent Unit: GRU

- Repeating module for the Gated Recurrent Unit (GRU) neural network:



Gated Recurrent Unit: GRU

- The outputs of the reset gate r_t and the update gate z_t at time t are:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + U_{hr}h_{t-1} + b_{hr})$$
$$z_t = \sigma(W_{iz}x_t + b_{iz} + U_{hz}h_{t-1} + b_{hz}),$$

where:

- σ: sigmoid function
- x_t: input, h_{t-1} is the previous output of the GRU cell
- W_{ir}, U_{hr}, W_{iz}, and U_{hz}: weight matrices
- b_{ir}, b_{hr}, b_{iz} +, and b_{hz} : bias vectors

Gated Recurrent Unit: GRU

- Output of the GRU cell:

$$h_t = (1 - z_t) * n_t + z_t * h_{t-1},$$

where n_t:

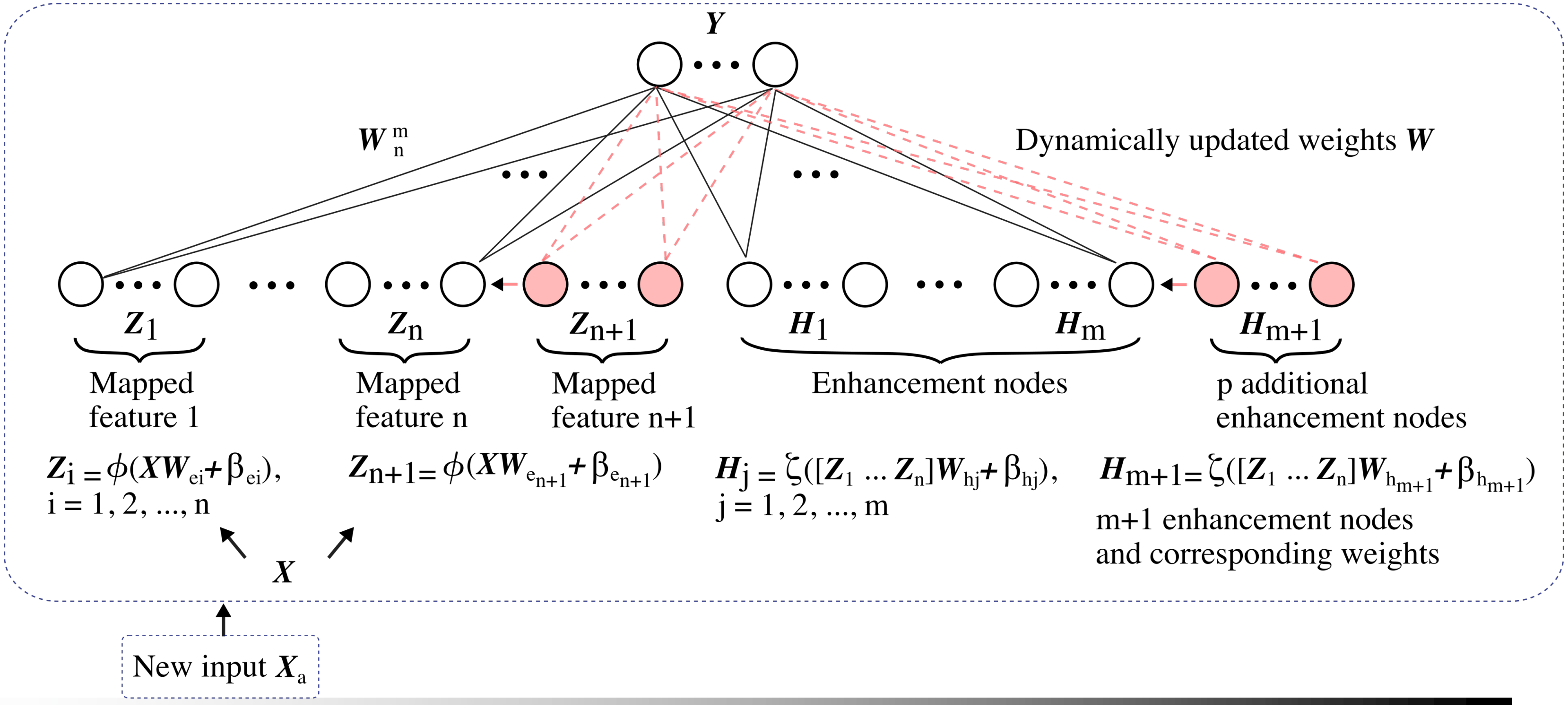
- $n_t = tanh(W_{in}x_t + b_{in} + r_t * (U_{hn}h_{t-1} + b_{hn}))$
- W_{in} and U_{hn}: weight matrices
- b_{in} and b_{hn}: bias vectors

Roadmap

- Introduction
- Data processing
- Machine learning models:
  - Deep learning: multi-layer recurrent neural networks
  - Broad learning system
- Experimental procedure
- Performance evaluation
- Conclusion and References

Broad Learning System: BLS

- Module of the Broad Learning System (BLS) algorithm with increments of mapped features, enhancement nodes, and new input data:



Original BLS

- Matrix $\boldsymbol{A}_x$ is constructed from groups of mapped features $\boldsymbol{Z}^n$ and groups of enhancement nodes $\boldsymbol{H}^m$ as:

$$\begin{aligned} A_x &= [\boldsymbol{Z}^n \mid \boldsymbol{H}^m] \\ &= \left[\phi(\boldsymbol{X}\boldsymbol{W}_{ei} + \beta_{ei}) \mid \xi(\boldsymbol{Z}_x^n \boldsymbol{W}_{hj} + \beta_{hj})\right], \end{aligned}$$

where: $i = 1, 2, \dots, n \; and \; j = 1, 2, \dots, m$

  - ϕ and ξ: projection mappings
  - $\boldsymbol{W}_{ei}$, $\boldsymbol{W}_{hj}$: weights
  - β_{ei}, β_{hj}: bias parameters

Modified to include additional mapped features $\boldsymbol{Z}_{n+1}$, enhancement nodes $\boldsymbol{H}_{m+1}$, and/or input nodes $\boldsymbol{X}_a$

RBF-BLS Extension

- The RBF function is implemented using Gaussian kernel:

$$\xi(x) = exp\left(-\frac{||x - c||^2}{\gamma^2}\right)$$

- Weight vectors of the output $\boldsymbol{HW}$ are deduced from:

$$\begin{aligned} \boldsymbol{W} &= (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T\boldsymbol{Y} \\ &= \boldsymbol{H}^{+}\boldsymbol{Y}, \end{aligned}$$

where:

- $\boldsymbol{W} = [\omega_1, \omega_2, \dots, \omega_k]$: output weights
- $\boldsymbol{H} = [\xi_1, \xi_2, \dots, \xi_k]$: hidden nodes
- $\boldsymbol{H}^{+}$: pseudoinverse of $\boldsymbol{H}$

Cascades of Mapped Features

- Cascade of mapped features (CFBLS):
  the new group of mapped features is created by using the previous group $(k - 1)$.
- Groups of mapped features are formulated as:

$$\begin{aligned} \boldsymbol{Z}_k &= \phi(\boldsymbol{Z}_{k-1}\boldsymbol{W}_{ek} + \beta_{ek}) \\ &\triangleq \phi^k\left(\boldsymbol{X}\,; \{\boldsymbol{W}_{ei}, \beta_{ei}\}_{i=1}^{k}\right), for\ k = 1, \ldots, n \end{aligned}$$

Cascades of Enhancement Nodes

- The first enhancement node in cascade of enhancement nodes (CEBLS) is generated form mapped features.
- The subsequent enhancement nodes are generated from previous enhancement nodes creating a cascade:

$$H_u \triangleq \xi^u(\boldsymbol{Z}^n\,;\,\{\boldsymbol{W}_{hi}, \beta_{hi}\}_{i=1}^{u}), for\ u = 1, \dots, m,$$

where:

  - $\boldsymbol{W}_{hi}$ and β_{hi}: randomly generated

Roadmap

- Introduction
- Data processing:
- Machine learning models:
- Experimental procedure
- Performance evaluation
- Conclusion and References

Experimental Procedure

- Step 1: Normalize training and test datasets.
- Step 2: Train the RNN models and BLS using 10-fold validation. Tune parameters of the RNN and BLS models.
- Step 3: Test the RNN and BLS models.
- Step 4: Evaluate models based on:
  - Accuracy
  - F-Score

RNN: recurrent neural network
BNN: board learning system

Number of BLS Training Parameters

Parameters	Code Red I	Nimda	Slammer	NSL-KDD
Mapped features	100	500	100	100
Groups of mapped features	1	1	25	5
Enhancement nodes	500	700	300	100
Incremental learning steps	10	9	2	3
Data points/step	100	200	100	3,000
Enhancement nodes/step	10	10	50	60

Roadmap

- Introduction
- Data processing:
  - BGP datasets
  - NSL-KDD dataset
- Machine learning models:
  - Deep learning: multi-layer recurrent neural networks
  - Broad learning system
- Experimental procedure
- Performance evaluation
- Conclusion and References

Training Time: RNN Models

	Datasets	$LSTM_2$	$LSTM_3$	$LSTM_4$	GRU_2	GRU_3	GRU_4
				Python (CPU)			
Time (s)	BGP (Slammer)	224.52	259.91	819.78	54.12	60.76	759.82
	NSL-KDD	4,481.73	4,614.66	11,478.62	1,108.31	1,161.80	11,581.30
				Python (GPU)			
Time (s)	BGP (Slammer)	30.74	34.94	38.82	31.03	35.46	40.22
	NSL-KDD	344.93	355.86	394.55	317.53	345.04	369.86

Training Time: BLS Models

	Datasets	BLS	RBF-BLS	CFBLS	CEBLS	CFEBLS
				Python (CPU)		
Time (s)	BGP (Slammer)	21.53	18.68	18.89	32.36	32.13
	NSL-KDD	99.47	98.27	98.13	108.23	108.14
				MATLAB (CPU)		
Time (s)	BGP (Slammer)	1.36	1.20	1.03	5.49	5.98
	NSL-KDD	6.91	6.24	6.55	8.88	8.95

LSTM Models: BGP Datasets (Python)

Model	Training Dataset	Accuracy (%) Test	Accuracy (%) RIPE (regular)	Accuracy (%) BCNET (regular)	F-Score (%) Test
$LSTM_2$	Code Red I	94.08	83.75	60.49	68.89
	Nimda	78.36	47.15	48.61	87.87
	Slammer	92.98	92.99	85.97	72.42
$LSTM_3$	Code Red I	88.54	79.38	58.82	55.96
	Nimda	85.57	39.10	40.28	92.22
	Slammer	90.90	92.01	84.38	67.29
$LSTM_4$	Code Red I	86.96	75.00	57.01	51.53
	Nimda	92.00	26.94	35.21	95.83
	Slammer	92.49	92.22	86.18	70.72

GRU Models: BGP Datasets (Python)

Model	Training Dataset	Accuracy (%) Test	Accuracy (%) RIPE (regular)	Accuracy (%) BCNET (regular)	F-Score (%) Test
GRU_2	Code Red I	87.47	80.07	60.21	52.97
	Nimda	70.71	48.96	58.26	82.83
	Slammer	91.88	93.33	90.90	69.42
GRU_3	Code Red I	88.07	79.44	60.56	53.51
	Nimda	80.21	38.40	44.24	89.02
	Slammer	91.76	95.21	90.83	68.72
GRU_4	Code Red I	91.84	77.50	60.07	63.87
	Nimda	87.36	35.00	39.38	93.25
	Slammer	92.14	92.15	90.35	70.11

BLS Models: BGP Datasets (Python)

Model	Training Dataset	Accuracy (%) Test	Accuracy (%) RIPE (regular)	Accuracy (%) BCNET (regular)	F-Score (%) Test
BLS	Code Red I	94.97	69.79	65.21	66.38
	Nimda	76.57	70.69	54.93	86.73
	Slammer	87.65	75.62	68.40	57.68
RBF-BLS	Code Red I	95.92	90.69	73.96	70.07
	Nimda	57.92	70.63	57.22	73.36
	Slammer	91.21	90.55	70.76	64.57

BLS Models: BGP Datasets (Python)

Model	Training Dataset	Accuracy (%) Test	Accuracy (%) RIPE (regular)	Accuracy (%) BCNET (regular)	F-Score (%) Test
CFBLS	Code Red I	95.16	69.38	61.74	71.08
	Nimda	55.71	68.06	58.26	71.56
	Slammer	89.28	71.25	61.81	60.99
CEBLS	Code Red I	94.94	70.69	60.35	65.22
	Nimda	66.43	74.10	54.51	79.83
	Slammer	91.01	87.71	82.43	66.38
CFEBLS	Code Red I	95.66	70.07	59.51	71.75
	Nimda	64.29	70.83	57.43	78.24
	Slammer	86.36	71.11	57.71	55.30

RNN and BLS Models: NSL-KDD Dataset (Python)

	Accuracy (%)		F-Score (%)	
Model	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$
$LSTM_4$	82.78	66.74	83.34	76.21
GRU_3	82.87	65.42	83.05	74.06
CFBLS	82.20	67.47	82.23	76.29

Incremental BLS Model: BGP and NSL-KDD Datasets (MATLAB)

Test	Accuracy (%)	F-Score (%)	Time (s)
Code Red I	94.37	65.10	0.926
Nimda	91.64	95.64	2.757
Slammer	89.31	63.07	2.805
KDDTest^{+}	81.34	81.99	32.99
KDDTest^{-21}	78.70	88.06	29.71

Roadmap

- Introduction
- Data processing:
- Machine learning models:
- Experimental procedure
- Performance evaluation
- Conclusion and References

Conclusion

- We evaluated performance of:
  - LSTM and GRU deep recurrent neural networks with a variable number of hidden layers
  - BLS models that employ radial basis function (RBF), cascades of mapped features and enhancement nodes, and incremental learning
- BLS and cascade combinations of mapped features and enhancement nodes achieved comparable performance and shorter training time because of their wide and deep structure.

Conclusion

- BLS models:
  - consist of a small number of hidden layers and adjust weights using pseudoinverse instead of back-propagation
  - dynamically update weights in case of incremental learning
  - better optimized weights due to additional data points for large datasets (NSL-KDD)
- While increasing the number of mapped features and enhancement nodes as well as mapped groups led to better performance, it required additional memory and training time.

Roadmap

- Introduction
- Data processing:
- Machine learning algorithms:
- Experimental procedure
- Performance evaluation
- Conclusion and References

References: Datasets

- BCNET :
  http://www.bc.net/
- RIPE RIS raw data:
  https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris
- NSL-KDD dataset:
  https://www.unb.ca/cic/datasets/nsl.html
- M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. in Security and Defense Appl. (CISDA)*, Ottawa, ON, Canada, July 2009, pp. 1–6.

References: Intrusion Detection

- Pandas: https://pandas.pydata.org/
- PyTorch: https://pytorch.org/docs/stable/nn.html
- Broadlearning: http://www.broadlearning.ai/
- C. M. Bishop, *Pattern Recognition and Machine Learning*. Secaucus, NJ, USA: Springer-Verlag, 2006.

- V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: a survey," *ACM Comput. Surv.,* vol. 41, no. 3, pp. 15:1–15:58, July 2009.
- T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Wireless Netw. Mobile Commun. (WINCOM)*, Fez, Morocco, Oct. 2016, pp. 258–263.
- N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, Feb. 2018.

References: Deep Learning

- R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Mach. Learn.*, vol. 8, no. 3, pp. 229–256, May 1992.
- S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Oct. 1997.
- G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *Computing Research Repository (CoRR)*, abs/1207.0580, pp. 1–18, Jul. 2012.
- K. Cho, B. van Merriënboer, C. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder–decoder for statistical machine translations,” in *Proc. 2014 Conf. Empirical Methods Natural Lang. Process*., Doha, Qatar, Oct. 2014, pp. 1724–1734.
- D. P. Kingma and J. Ba, “Adam: a method for stochastic optimization,” in *Proc. 3rd Int. Conf. Learn. Representations*, San Diego, CA, USA, May 2015, pp. 1–15.
- K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, “LSTM: a search space odyssey,” *IEEE Trans. Neural Netw. Learn. Syst*., vol. 28, no. 10, pp. 2222–2232, Oct. 2017.
- C. Yin, Y. Zhu, J. Fei, and X. He, “A deep learning approach for intrusion detection using recurrent neural networks,” *IEEE Access*, vol. 5, pp. 21954–21961, Nov. 2017.

References: Broad Learning System

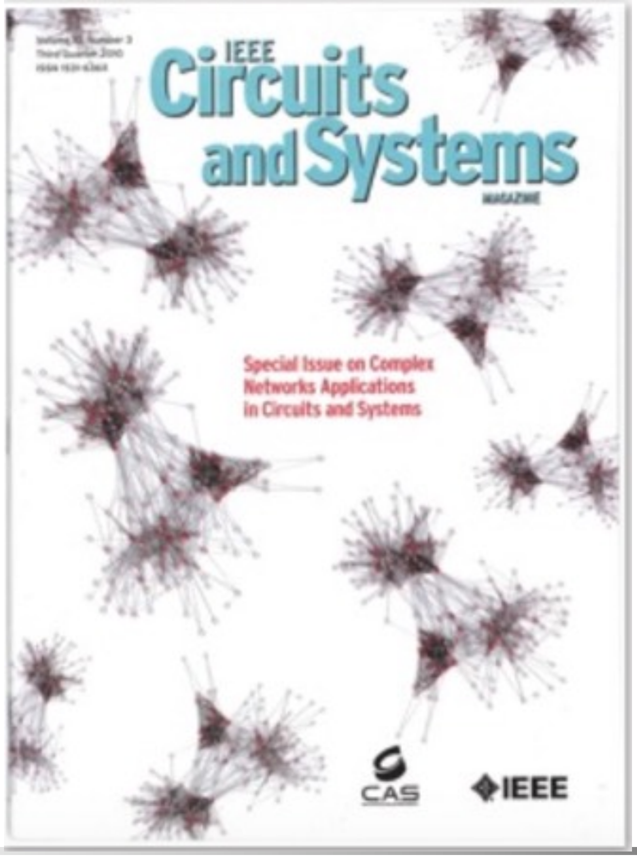
- Z. Liu and C. L. P. Chen, "Broad learning system: structural extensions on single-layer and multi-layer neural networks," in *Proc. 2017 Int. Conf. Secur., Pattern Anal., Cybern.*, Shenzhen, China, Dec. 2017, pp. 136–141.
- C. L. P. Chen and Z. Liu, "Broad learning system: an effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.
- C. L. P. Chen, Z. Liu, and S. Feng, "Universal approximation capability of broad learning system and its structural variations," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 4, pp. 1191–1204, Apr. 2019.

IWCSN: 2004 - 2019

- International Workshop on Complex Systems and Networks (IWCSN): http://iwcsn.eie.polyu.edu.hk/

Publications

- *IEEE CAS Magazine* Special Issue on Applications of Complex Networks, vol. 10, no. 3, 2010.



Publications: http://www.sfu.ca/~ljilja

Book chapters:

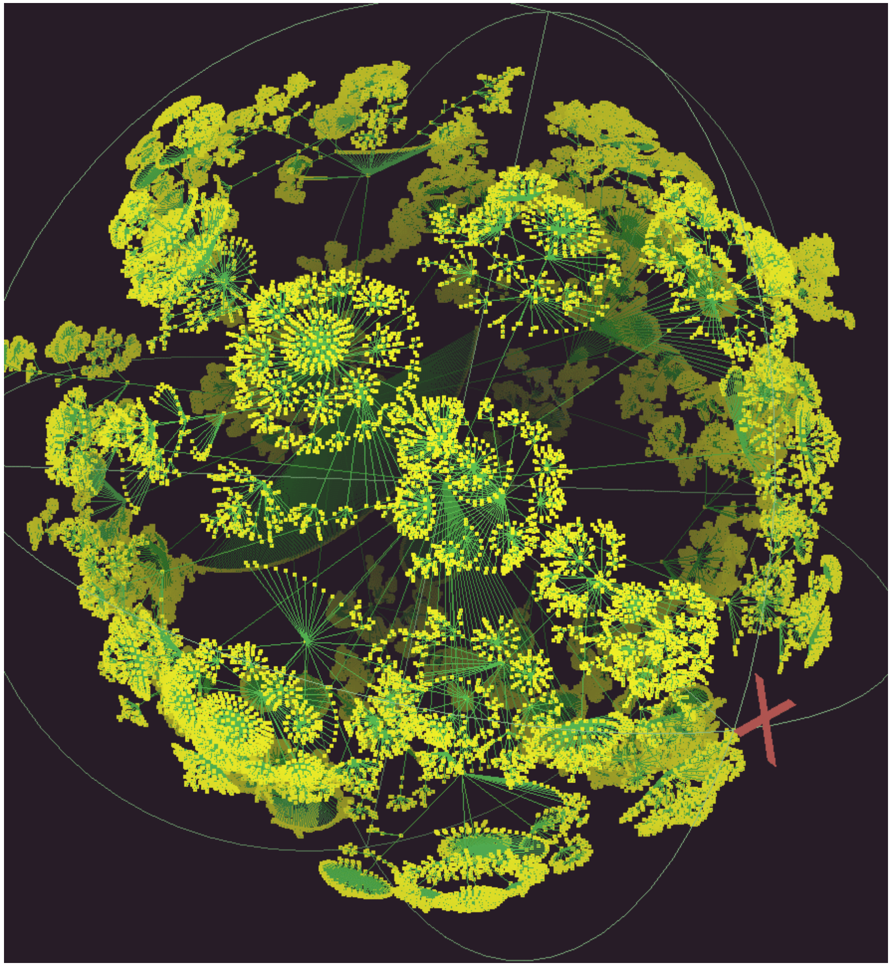
- Q. Ding, Z. Li, S. Haeri, and Lj. Trajković, “Application of machine learning techniques to detecting anomalies in communication networks: Datasets and Feature Selection Algorithms” in *Cyber Threat Intelligence,* M. Conti, A. Dehghantanha, and T. Dargahi, Eds., Berlin: Springer, pp. 47-70, 2018.
- Z. Li, Q. Ding, S. Haeri, and Lj. Trajković, “Application of machine learning techniques to detecting anomalies in communication networks: Classification Algorithms” in *Cyber Threat Intelligence,* M. Conti, A. Dehghantanha, and T. Dargahi, Eds., Berlin: Springer, pp. 71-92, 2018.

Publications: http://www.sfu.ca/~ljilja

Conference publications:

- Z. Li, A. L. Gonzalez Rios, G. Xu, and Lj. Trajkovic, "Machine learning techniques for classifying network anomalies and intrusions," in *Proc. IEEE Int. Symp. Circuits and Systems*, Sapporo, Japan, May 2019.
- A. L. Gonzalez Rios, Z. Li, G. Xu, A. Dias Alonso, and Lj. Trajković, "Detecting Network Anomalies and Intrusions in Communication Networks," in *Proc. 23rd IEEE International Conference on Intelligent Engineering Systems 2019*, Gödöllő, Hungary, Apr. 2019, pp. 29-34.
- Z. Li, P. Batta, and Lj. Trajkovic´, "Comparison of machine learning algorithms for detection of network intrusions," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Miyazaki, Japan, Oct. 2018, pp. 4248–4253.
- P. Batta, M. Singh, Z. Li, Q. Ding, and Lj. Trajković, "Evaluation of support vector machine kernels for detecting network anomalies," in *Proc. IEEE Int. Symp. Circuits and Systems*, Florence, Italy, May 2018, pp. 1-4.
- Q. Ding, Z. Li, P. Batta, and Lj. Trajković, "Detecting BGP anomalies using machine learning techniques," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics (SMC 2016),* Budapest, Hungary, Oct. 2016, pp. 3352-3355.

lhr: 535,102 nodes and 601,678 links



http://www.caida.org/home/