December 2016

Neuromorphic Learning Systems for Supervised and Unsupervised Applications

Qiuwen Chen
Syracuse University

ABSTRACT

The advancements in high performance computing (HPC) have enabled the large-scale implementation of neuromorphic learning models and pushed the research on computational intelligence into a new era. Those bio-inspired models are constructed on top of unified building blocks, i.e. neurons, and have revealed potentials for learning of complex information. Two major challenges remain in neuromorphic computing. Firstly, sophisticated structuring methods are needed to determine the connectivity of the neurons in order to model various problems accurately. Secondly, the models need to adapt to non-traditional architectures for improved computation speed and energy efficiency. In this thesis, we address these two problems and apply our techniques to different cognitive applications.

This thesis first presents the self-structured confabulation network for anomaly detection. Among the machine learning applications, unsupervised detection of the anomalous streams is especially challenging because it requires both detection accuracy and real-time performance. Designing a computing framework that harnesses the growing computing power of the multicore systems while maintaining high sensitivity and specificity to the anomalies is an urgent research need. We present AnRAD (Anomaly Recognition And Detection), a bio-inspired detection framework that performs probabilistic inferences. We leverage the mutual information between the features and develop a self-structuring procedure that learns a succinct confabulation network from the unlabeled data. This network is capable of fast incremental learning, which continuously refines the knowledge base from the data streams. Compared to several existing anomaly detection methods, the proposed approach provides competitive detection accuracy as well as the insight to reason the decision making. Furthermore, we exploit the massive parallel structure of the AnRAD

framework. Our implementation of the recall algorithms on the graphic processing unit (GPU) and the Xeon Phi co-processor both obtain substantial speedups over the sequential implementation on general-purpose microprocessor (GPP). The implementation enables real-time service to concurrent data streams with diversified contexts, and can be applied to large problems with multiple local patterns. Experimental results demonstrate high computing performance and memory efficiency. For vehicle abnormal behavior detection, the framework is able to monitor up to 16000 vehicles and their interactions in real-time with a single commodity co-processor, and uses less than 0.2ms for each testing subject.

While adapting our streaming anomaly detection model to mobile devices or unmanned systems, the key challenge is to deliver required performance under the stringent power constraint. To address the paradox between performance and power consumption, brain-inspired hardware, such as the IBM Neurosynaptic System, has been developed to enable low power implementation of neural models. As a follow-up to the AnRAD framework, we proposed to port the detection network to the TrueNorth architecture. Implementing inference based anomaly detection on a neurosynaptic processor is not straightforward due to hardware limitations. A design flow and the supporting component library are developed to flexibly map the learned detection networks to the neurosynaptic cores. Instead of the popular rate code, burst code is adopted in the design, which represents numerical value using the phase of a burst of spike trains. This does not only reduce the hardware complexity, but also increases the result's accuracy. A Corelet library, NeoInfer-TN, is implemented for basic operations in burst code and two-phase pipelines are constructed based on the library components. The design can be configured for different tradeoffs between detection accuracy, hardware resource consumptions, throughput and energy. We evaluate the system using network intrusion detection data streams. The results show higher detection rate than some conventional approaches and real-time performance, with only 50mW power consumption. Overall, it achieves 10^8 operations per Joule.

In addition to the modeling and implementation of unsupervised anomaly detection, we

also investigate a supervised learning model based on neural networks and deep fragment embedding and apply it to text-image retrieval. The study aims at bridging the gap between image and natural language. It continues to improve the bidirectional retrieval performance across the modalities. Unlike existing works that target at single sentence densely describing the image objects, we elevate the topic to associating deep image representations with noisy texts that are only loosely correlated. Based on text-image fragment embedding, our model employs a sequential configuration, connects two embedding stages together. The first stage learns the relevancy of the text fragments, and the second stage uses the filtered output from the first one to improve the matching results. The model also integrates multiple convolutional neural networks (CNN) to construct the image fragments, in which rich context information such as human faces can be extracted to increase the alignment accuracy. The proposed method is evaluated with both synthetic dataset and real-world dataset collected from picture news website. The results show up to 50% ranking performance improvement over the comparison models.

NEUROMORPHIC LEARNING SYSTEMS FOR SUPERVISED AND UNSUPERVISED APPLICATIONS

By

Qiuwen Chen
B.Sc. Beijing University of Posts and Telecommunications, 2009
M.Sc. Beijing University of Posts and Telecommunications, 2012

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical and Computer Engineering

Syracuse University
December 2016

ACKNOWLEDGMENTS

Many people have helped during my graduate career, without whom this dissertation could not have been finished.

First of all, I would like to express my most sincere gratitude to my advisor, Dr. Qinru Qiu, who trusted me and offered me the chance of doctorate study on March 20, 2012 when I needed such recognition most. From since, she provided patient guidance and support all the way through the completion of this degree. Without her, I would have never experienced and known so much more. I would like to thank my committee members, Dr. Roger Chen, Dr. Peng Gao, Dr. Yingbin Liang, Dr. Jian Tang and Dr. Yanzhi Wang, for their valuable feedback. I would also like to thank Dr. Qing Wu for his advices.

I would like to thank all my labmates: Dr. Yang Ge, Dr. Hao Shen, Wei Liu, Jianwei Cui, Yukan Zhang, Khadeer Ahmed, Ryan Luley, Zhe Li, Amar Shrestha, Yilan Li and Ziyi Zhao. Your collaboration, company and academic insights are precious during my research journey.

My final but everlasting appreciation goes to my family members, who have always supported me in my choices, and encouraged me when I was down. Without their trusts and constant love, I would never be able to come this far. This thesis is dedicated to my parents, Zixi Chen and Shuqing Lian, and my fiance Zhiruo Zhao.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Neuromorphic learning systems leverage bio-inspired computation to model different applications. Generally, they employ integrate-and-fire architectures that mimics the human decision-making processes. The challenges of building such systems include fitting data patterns with such knowledge models using supervised or unsupervised learning, constructing appropriate network structures, as well as implementing the system on parallel and brain-inspired hardware architectures.

In this chapter, we discuss the motivation of the study. We introduce the general neural computing model, and discuss the sample complexity and strategies to train such networks. Then, applications of different learning systems are proposed. Finally, the contributions of the thesis are reviewed.

1.1 Motivation

In recent years, studies on machine learning, especially neural networks have received wide attentions. This is due to the advancements in computation devices and the availability of the training data, i.e.“Big Data". Sophisticated models can be built to capture very complex patterns in various applications. For example, Krizhevsky et. al. [53] developed

deep convolutional neural network that trained on ImageNet [30] using Graphic Processing Units (GPU), which achieved close to human accuracy in image classification; Hochreiter et. al. [39] proposed Long-short term memory (LSTM) recurrent neural network (RNN) that can effectively process sequential information such as the voice and natural language as a human does. One groundbreaking feat of the artificial intelligence research is the victory of Google's Alpha Go [84] system over the human Go champion on March 2016, which also revealed three major aspects of the machine learning systems.

- **The importance of effective modeling**. The solution space of the Go game is in the order of 10^{170} possible paths. No ordinary learning model is capable of capturing so many patterns, because fitting the solution space will require a huge amount of learnt parameters, and thus an infeasible number of training samples. In the Alpha Go case, Monte Carlo tree search is used to mitigate the problem, in that the program plays with itself to improve the learning. But the method is not always available for other applications. Therefore, many techniques are studied to prevent overfitting. Typical examples include transfer learning, parameter sharing and regularization. In this study, more interests are drawn to use the network structure and learning rule to resolve the sample complexity.

- **The demand of computation power**. With all those delicate learning strategies, Alpha Go would not be possible if it cannot perform the computations quickly. According to the reports [31] about the Go game challenge, Alpha Go uses 1920 CPUs and 280 GPUs to keep on the pace of an interactive game. The time consumption is always a concern as the model being more and more complex, and the solution usually turns to parallelization. However, parallelization is not simply stacking the hardware. Effective utilization of the computing device depends on the partition of the problems and the workload distribution of the particular models. Therefore, another target of this study is to investigate the concurrent structure of the learning model that improve the throughput and responsiveness, so that real-time processing

can be achieved.

- **The challenge of power reduction**. The Alpha Go system consumes 10^6 Watts of power, while the human brain works with around 20 Watts. Although the power consumption may not be the first priority for a server or data center environment, it is critical when implementing the learning models on mobile devices and embedded platforms. Different from traditional signal processing, a learning system's computation does not always require high precision, and it has intensive memory accesses. As we are developing and parallelizing neuromorphic algorithms, the study also looks into implementation on top of neuromorphic hardware to achieve ultra-high energy efficiency.

This thesis addresses the above three major research problems. Firstly, the neuromorphic learning models are explored. We improve the quality of the model from the algorithm point of view, and try to solve different problems by modeling the supervised and unsupervised applications with proper network architectures. In this way, different applications can be adapted with unified computing units that ease the implementation. Secondly, we implement the networks using parallel devices. The concurrent structure of the neural models is exploited to accelerate the inference computation. Thirdly, the models are further implemented with non-traditional architectures of spiking neural network. We investigate mapping and signal processing using such hardware to enable the network operate in an energy efficient manner.

1.2 Neuromorphic Computing Model

Fig. 1.1 shows how neurons communicate with each other. Neuron is the basic building block of the brain. It contains a set of dendrites that receive impulses from other neurons, and a cell body that process the signals. When a neuron accumulates certain amount of

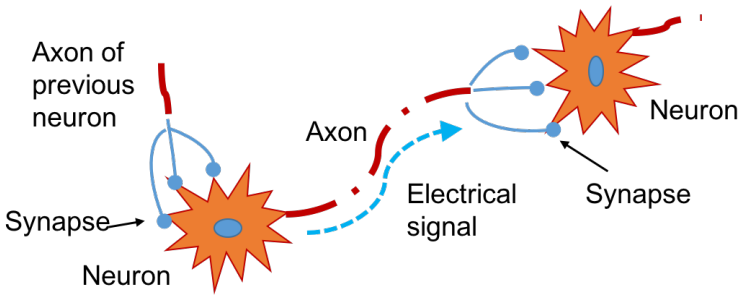


Fig. 1.1: Neuron Model

impulses, it fires and passes the electrical signal to the others through its axons. The junction between two neuron cells are called synapse, whose strength decide the influence of one neuron on the other. The emitter and the receiver of the signal are referred to as the presynaptic neuron and the postsynaptic neuron respectively. The learning is achieved by forming the neuron connections and the synaptic strength, and the complex inference task can be achieved by chain reaction in large neuron networks. Although neurons fire at a low frequency (1 - 200Hz), they are massive in the amount and operate in parallel. Therefore, the brain is capable of fast cognitive tasks.

By mimicking the biological nerve system, the neuromorphic model is simplified as in Fig. 1.2, which shows a single neuron cell and its input/output. The computation referred to as *integrate-and-fire* is described by Equation (1.1).

$$y = f(\sum_{i=0}^{N} w_i x_i + b) \tag{1.1}$$

Here, this neuron receives signals from another N presynaptic neurons. x_i is the signal received from the i^{th} neuron's axon and w_i denotes the synaptic weight, which is a multiplier determining the influence of x_i. The neuron integrates the incoming excitations and a bias term b, and applies the sum to $f(.)$, the *activation function* to add non-linearity. And y is the output of this neuron that can be passed to other neurons or used as the final output. In this function, w_i and b are learnt parameters that are tuned using training samples by different learning methods. The main idea is to have w_i and b reach values such that the

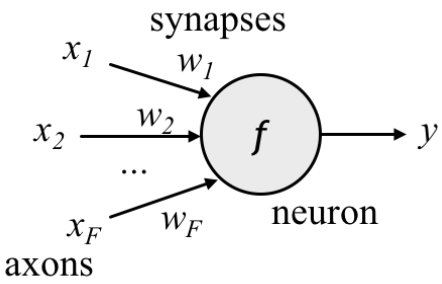


Fig. 1.2: Integrate and Fire Unit

generated y approaches the desired result.

In the case of artificial neural network (ANN), a popular learning method is stochastic gradient descent (SGD). A loss function loss(y) is defined reflecting the discrepancy between the expected output and the network-generated output. Then the error is back-propagated to update the model by finding the derivatives of the loss with respect to the parameters.

$$\Delta w_i = \sum_{\text{batch}j} \frac{\partial \text{loss}(y)}{\partial y} \frac{\partial y}{\partial w_i} \tag{1.2}$$

In Equation (1.2), Δw_i is the gradient. It is used to adjust w_i by a certain updating strategy. For a fixed learning rate α, the parameter is updated by $w_i' = w_i - \alpha \Delta w_i$, while there are other more sophisticated updating rules to improve the learning quality. Training is performed by iterating the data, drawing random sample mini-batches and updating the parameters in multiple epochs. Take logistic regression for example, the activation is calculated by sigmoid function $f(x) = 1/[1 + exp(\sum_{i=0}^{N} w_i x_i + b)]$, and the parameter is trained to directly optimize the conditional probability $p(y|x)$. In a supervised binary classification task, we can use the 0-1 prediction error to tune the parameters. Such method is categorized as a *Discriminative Model* [70]. Other than discriminative model and ANN, there are also neuromorhpic methods using *Generative Model* that optimize the joint probability $p(y, x)$. They offer different asymptotic error and sample complexity in obtaining the model parameters. We will discuss the model selections in Section 1.3 and further in

Section 2.3.2.

The neuromorphic learning models draw wide attention partly due to their capacity. It is proved that neural network is capable of approximate any function given sufficient neurons [5] and training samples. However, another important reason is that the neuromorphic computing architecture facilitates efficient implementation of the learning systems on non-traditional hardware platforms.

Traditional computing systems are built based on the Von Neumann architecture, in which the computation unit (CPUs) and the data/code storage are separated and interact with each other through a bus. While such setup is general enough to handle most of the logic and arithmetic tasks, it also suffers from inefficiency that the data have to be moved back and forth through the bus. This is referred to as the Von Neumann bottleneck [6]. This limitation is especially significant with large-scale learning systems because the models usually include some form of knowledge base and the computation is data-intensive. The inefficiency in the traditional hardware results in both slow computation and high power consumption.

Human brain performs learning tasks in such efficient way, so does the bio-inspired computation model. Firstly, the neuromorphic models feature tightly coupled memory (synapses) and computation (neuron cell body). Therefore, they can easily adapt to non-traditional architectures that eliminate the memory bottleneck. Secondly, the computation of neuromorphic models don't usually require high precision. So approximate computation such as spike codes can be adopted to reduce the power consumption. Thirdly, the inference computation of neurons are massively concurrent. It can be exploited using parallel devices to improve the performance and provide real-time services.

1.3 Sample Complexity

While the neuromorphic systems provide benefits such as modeling capacity and efficient implementation, a key challenge is how to obtain the set of parameters, i.e. w_i and b in Equation (1.1). If we have infinite amount of data, we can simply apply SGD and the model would eventually arrive at a good quality, but this is obviously an unreasonable assumption, as real world data, especially labeled data are costy to collect. Therefore, it is important to know how many data are needed to fit a good model given certain capacity.

Continue with the example of logistic regression in the previous section, we consider binary classification here for simplicity. The output of such a system can be obtained by thresholding the y so the solution can be labeled $\alpha = I(y > 0) \in \{0, 1\}$. Being a discriminative model, the parameters are fit to optimize the conditional probability $p(y|x)$ on the training data. So essentially the model defines a mapping h_{NN} that map a sample x to a label α. The sample complexity in training such a model can by analyzed by consider the VC (Vapnik-Chevonenkis) dimension [89].

Consider H the hypothesis space that contains all the parameter setup of classifier h_{NN}. Given a set of sample points X, it is defined that H *shatters* X if and only if for any label assignment on X, there always exists some hypothesis $h \in H$ that correctly classify the samples. And the VC dimension of H is the cardinality of the largest sample set X that can be shattered by H. If arbitrary large set of X can be shattered, then $\text{VC}(H) = \infty$. For example, logistic regression is a linear model that divide the samples with a hyperplane in the feature space, so the VC dimension of an N-input network is $N + 1$. For classifier such as 1-NearestNeighbor, the VC dimension is infinity, because a sample can always be correctly classified when its nearest neighbor is itself. However, larger VC dimension does not indicate better model. According to Vapnik et. al [89], with probability at least

$1-\delta, 0 \leq \delta \leq 1$, the error bound of a classifier $h \in H$ is as Equation (1.3).

$$\varepsilon(h) < \varepsilon(h_\infty) + \sqrt{\frac{1}{M}[\text{VC}(H)\ln\frac{2M}{\text{VC}(H)} + \ln\frac{4}{\delta}]} \tag{1.3}$$

Here, $\varepsilon(h)$ is the classification error of h. $\varepsilon(h_\infty)$ is the asymptotic error when the model is trained with infinite amount of data. M is the number of training samples. For logistic regression whose $VC(H) = N+1$, in order to have the error gap being some fixed constant, the sample size required would be $M = O(N)$. That said, we need as many training samples as the number of parameters. While deeper networks do not strictly follow the derivation, the training complexity generally has similar relationship to the model capacity. As we know, today's neural networks often have millions of learnt parameters, but the available training data are much more limited. When the training sample is too small with respect to the number of parameters, *overfitting* will occur in that testing error is much higher than the training error.

To address the problem, different methods are studied. An incomplete list is as the followings.

- **Transfer learning**. In the case when labeled data is hard or expensive to obtain, it is possible to leverage unlabeled data to perform pre-training, and thus reduced the sample complexity in fitting the classifier. For example, Raina et. al [80] used unlabeled image randomly downloaded from the Internet to train a feature detector, which reduces the input dimension from the raw image pixels to a representational vector. Then, labeled data is used to fine-tune the classifier, and overfitting is prevented. Nowadays, such techniques are widely used in applications based on deep network. Deep convolutional neural networks trained on large image dataset, e.g. ImageNet [30] are cascaded with application-specific layers to identify information such as human faces or scenes.

- **Weight sharing**. When a network contains millions of synapses, it is not always

necessary to have the same number of distinct weights since many of the features and neurons could have similar behaviors. Take convolutional neural network (CNN) [56] for instance, it uses a sliding window to scan the whole image while pixels in different regions are processed with the same set of weights. It is assumed that regions of an image are not that different in the sense of feature detection, but the technique significantly reduced the unique parameters to learn.

- **Regularization**. Except from reducing the number of parameters, we can also limit the value selection of them. Classical techniques include L1 and L2 regularization, which limit the parameter to taking smaller values. Basically, the idea is to apply some form of prior information on the parameters such that they cannot be optimized to take any value. Also, regularization can be achieved by intentionally add noise to the input samples to stable the learning process. A popular technique in neural network is dropout [86], which randomly disable a subset of the neurons during the training, so that the learnt parameters tend to have less correlation with each others. Then in test, all neurons are enabled and the total input values are rescaled to match those during training.

- **Network structuring**. This group of techniques modifies the network connections and topologies based on some prior knowledge. For example, in some Bayesian networks, connections are selected based on expert knowledge; with deep neural networks, the numbers of neurons and layers, as well as their behaviors are tuned with different application scenarios such as vision and voice. In this thesis, we develop self-structured algorithm [26] to optimize the network structure for quality and performance. We also design sequential embedding layers for neural networks [23] to deal with assumption in noisy data.

- **Generative model**. Finally, the learning rule matters a lot. The neural network and logistic regression are usually trained with SGD which provides great quality given

enough training data. However, in the case that less data are available, generative model such as Naive Bayes could provide better accuracy [70]. Therefore, targeting on different applications, we choose different learning models to best fit the requirements. Specifically, we use inference network called cogent confabulation [38] for unsupervised anomaly detection, and use deep neural network for image-text retrievals.

1.4 Cogent Confabulation

For generative model based inference network, we adopt *cogent confabulation* [38] as the computing model for probabilistic inference. Cogent confabulation is a connection-based cognitive model that captures correlations between features at the symbolic level. It describes the basic dimensions of the observation using a set of features (e.g. color and shape). The attributes of a given feature (e.g. red color, round shape) are referred to as the *symbols*, which are analogous to neurons in the biological nervous system. Their pairwise conditional probabilities are referred to as the *knowledge links*, which are analogous to synapse plasticity between neurons. The link updates follow Hebian learning, and can be potentially learned using STDP rule [69], which is known to be the biological process to adjust the strength of neuron connections. To better organize the knowledge, neurons that represent the same features (e.g. colors) are grouped into lexicons, and knowledge links between symbols of two lexicons are realized as probability matrices. The i, jth entry of a matrix gives the conditional probability $p(s_i|t_j)$ between the symbols s_i in the source lexicon and t_j in the target lexicon. To distinguish the term from knowledge links, we refer to the presence of such probability matrix as a *connection*. Lexicons and the connections between them form a knowledge graph; therefore, we also refer to lexicons as nodes. During learning, the knowledge links are established and strengthened as symbols being co-activated.

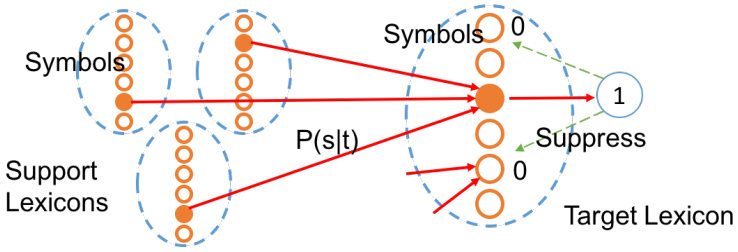


Fig. 1.3: Confabulation Network Example

Whenever an attribute is observed, the corresponding symbol (i.e. neuron) is activated, and an excitation is passed to the other symbols through the knowledge links (i.e. synapses). The excitation of a symbol t in lexicon l is calculated by summing up all incoming knowledge links as in function (1.4):

$$y(t) = \sum_{k \in F_l} \{ \sum_{s \in S_k} [I(s) \ln \frac{p(s|t)}{p_0}] + B \}, t \in S_l \tag{1.4}$$

In this function, t is one of the symbols of node l. F_l denotes the set of nodes that have connections to l, and S_k is the symbol set of lexicon k. $I(s)$ is the firing indicator, which takes value 0 or 1 given the occurrence of the source symbol s. p_0 is the minimum probability that is empirically selected to ensure that t has positive excitations. B is a constant band gap to favor the symbols that receive more activation from distinct lexicons. We use $B = 0$ in this work to let the abnormality determined only by the synaptic weights. The excitation level is essentially the log likelihood of symbol t given the rest of the observations.

The computation model of confabulation takes the same form of a general neural computing model in Equation (1.1). The fan-in connections are the collection of all symbols from multiple support lexicons. Equation (1.4) uses a linear activation to find the excitation level of symbol t. When a decision is to be made, we can use a winner-takes-all function as the activation that select the symbol in the target lexicon whose excitation level is the highest. Fig. 1.3 shows an example of the inference operation. Only the third neuron in the target lexicon outputs “1”, while its cohorts are suppressed to “0”.

The inference network is categorized a generative model, because the synaptic weights

$p(s|t)$ is learnt by modeling the joint probability of s and t, $\hat{p}(s|t) = c(s,t)/c(t)$, where $c(.)$ counts the occurrences of the event in the training set $\{s^i, t^i\}_{i=1}^{M}$. Here M is the training sample size. We denote the mapping $h \rightarrow X- > Y$ trained using confabulation model h_{CFB} and that trained using logistic regression h_{NN}. Let the total number of parameters be N and $\varepsilon(h)$ is the error of the classifier, we have Equation (1.5).

$$\varepsilon(h_{\text{NN}}^{\infty}) \leq \varepsilon(h_{\text{CFB}}^{\infty}) + \epsilon_0 \tag{1.5}$$

Here ϵ_0 is a constant error. That said, given infinite amount of training samples, logistic regression usually reaches a lower asymptotic error than confabulation model does. This is because logistic regression uses discriminative learning that directly estimates $p(y|x)$. Since the parameters can take any value, with ideal learning process, logistic regression can reach the best accuracy among the set of all linear models H that share the same connection topology. So logistic regression with infinite training data must be no worse than a linear model such as confabulation.

Since confabulation assumes parameter to be probabilities (add up to 1), and assume independency among the support lexicons, the parameters learnt are more constrained compare to those of neural networks. The constraints decides confabulation could possibly have a lower sample complexity. Actually, we will show in Section 2.3.2 that compared with logistic regression's $O(N)$ complexity, confabulation only requires $O(\ln N)$ samples to reach asymptotic error. The error bound follows Equation (1.6).

$$\varepsilon(h_{\text{CFB}}) \leq \varepsilon(h_{\text{CFB}}^{\infty}) + e^{-(\gamma-\omega)^2 N}, \gamma > 0, \omega = O(\sqrt{\frac{\ln N}{M}}) \tag{1.6}$$

By taking $M = O(\ln N)$ samples, the asymptotic error will be less than a constant. On the other hand, just because confabulation is more constrained, its parameters will be more biased when its prior assumptions are not satisfied in some datasets.

To summarize the section, neural networks are more suitable when training data are

sufficient. For applications that finds the most likely answers, such as classification and recommendation, we use artificial neural network to achieve lower error bound. However, for unsupervised learning tasks such as anomaly detection, large training sample may not always be favored [96]. In such cases, we would choose confabulation network and ensemble multiple models trained with data subsamples to obtain fast convergence.

1.5 Applications

Based on the discussion in previous sections, we will apply appropriate types of neuromorphic computing models to different applications. Specifically, self-structured confabulation network is applied to unsupervised anomaly detection on streaming data; deep neural network is applied to cross-modal information retrievals between image and text.

- **Anomaly Detection** refers to the task of identifying data patterns that do not conform to regular observations. It is different from classification in that there are no labels in the data, so we cannot train models that assign a sample to a certain class. Anomaly detection systems are usually unsupervised, and target on finding unseen patterns from the training set. Many studies have been carried out for detection on static datasets [21], in which samples are single points in the feature space. Such data can be analyzed using very complex algorithms and applied iterative processing. However, real application usually cannot afford such post-processing, and thus detection on streaming data is required. In streaming applications, data are no longer single points, but series of samples spanning in the time domain. The samples come in event-driven way and demand the system to process them in one pass. These characteristics require the detection method being capable of modeling historical patterns, capture subtle feature interactions, and process the data in realtime. Anomaly detection can be applied to various domains, including network intrusion detection and traffic monitoring. However, most of the researches are focused on traditional data

mining and statistic methods. In the machine learning community, neuromorphic models, on the other hand, are mostly neural networks for supervised tasks, while less studies are conducted for inference network. We choose confabulation network because of its fast convergence and flexible network structuring. We also exploit its parallel computing pipeline to provide high computing performance and power efficient solutions.

- **Visual-semantic Modeling**. In recent years, it has been an emerging topic to bridge the gap between image and natural language using learning methods. It an important feature for many cross-modal retrieval tasks. For example, we can search for image using natural language, automatically assign illustrations for articles, and provide relevant image given conversation contexts. From image to text, we can assign captions to images, or use image query to retrieve related articles from a text database. However, most of the existing works are focused on associations between tightly coupled image and text samples [48, 91], which are not how data appear in their natural form. In this thesis, we elevate the problem by considering loosely couple pairs. We choose deep neural network for its superior quality in supervised learning tasks, while we also optimize from the network structure perspective to handle the noisy texts and rich image contexts.

1.6 Contributions

This thesis studies neuromorphic learning models for both supervised and unsupervised applications. We improve the algorithms to offer state-of-the-art quality, and also investigate the computation performances. From the implementation aspect, neural networks have been widely studied for their parallelization [9, 46] and low-power implementations [32], but the studies on inference network are relatively less. Therefore, we focus on using non-traditional architectures to optimize the anomaly detection network in order to provide fast

processing and energy efficiency.

The organization and contributions of this thesis are concluded as the following.

- In Chapter 2, we propose the method of cogent confabulation based detection algorithm and analyze the algorithm's error bounds. We start the chapter by first conducting an empirical study on using manually configured network structure to capture abnormal vehicle behaviors on a large road network, and demonstrate the inefficiency of such setup. Then, we develop a self-structured method using data-driven approach. The network is able to capture the data patterns with succinct topological configuration. Our method is compared with different baselines on multiple datasets and show competitive accuracy.

- In Chapter 3, we implement the confabulation network on parallel computing devices. Our structuring algorithm moves the complexity to the feature design space, and offers a highly concurrent pipeline that is suitable for acceleration. We exploit multi-threaded CPU, general purpose graphical computing unit (GPGPU) and the Intel Xeon Phi co-processor. Our implementations feature fine-grained parallelism which efficiently distributes the workload, while provide scalable extension to adapt to multiple trained models and concurrent data streams.

- In Chapter 4, the confabulation network is further designed and realized on spiking neural networks. By using the IBM neurosynaptic system, we implement an intrusion detection network with extremely low power. We propose a new spike burst code that offers compact representations and convenient implementations. And the burst code pipeline is supported by our Corelet Library. Compared with implementation on traditional architectures, the spiking network achieve both high accuracy and power/energy efficiency.

- In Chapter 5, we improve the deep fragment embedding method. An bi-directional mapping mechanism is established between image and text. As mentioned in the

previous section, this model is able to handle loosely coupled text and image data. We achieve this with two optimizations. Firstly, two embedding layers are stacked, with the first one filtering out irrelevant text segments, the other one performing accurate association. Secondly, multiple image CNN's are integrated to capture both object level and facial information from the image. The final network significant improve the retrieval results on real data set crawled from the web.

- Finally in Chapter 6, the works in the thesis are reviewed and summarized. New directions for further improving the studies will be proposed.

CHAPTER 2
SELF-STRUCTURED CONFABULATION NETWORK FOR ANOMALY DETECTION

2.1 Introduction

Anomalous data stream detection is inherently hard. Firstly, labeled data are expensive to obtain, and some abnormal classes are not foreseeable by the time of modeling. Secondly, the input streams are continuous and infinite, which requires the model to learn new data by one pass and recall in a real-time manner. Thirdly, users require more than just output labels, so the algorithm should also provide insights into the decision-making process. Such restrictions rule out many traditional methods such as multi-class classifiers, off-line analysis and obscure models.

Confabulation network is a candidate solution to detect outliers and provide reasoning in some well understood problems. However, building such networks might require expert knowledge. Thus the neuron nodes and the synapses between them must be reconfigured when applied to new applications. This limitation makes the confabulation-based approaches inflexible in handling different datasets. To address the above problems, we present AnRAD (Autonomous Anomaly Reasoning and Detection), a transparent

framework that provides real-time online detection using a self-structured confabulation network.

To motivate the discussion about transparent network and self-structuring, consider an example of detecting voice recordings spoken in foreign languages. An artificial neural network can be used to identify a non-English clip. It encodes vocal features into nodes through weighted links and the weight adjustment is achieved by repeated practice of native tongues. Although such detection is fast, it does not reveal why the clip is not in English since the nodes lose the original meanings. In contrast, a confabulation network preserves the meaning of the features, and therefore, it might reveal some inconsistencies of tone combinations in the English context. Although the recall process might be slightly slower than that of a neural network, it provides valuable information about why a tested subject is labeled as an anomaly. Unlike neural networks that employs fixed numbers of nodes and links, which nodes to select and how to connect them together in the confabulation network is application specific, and hence usually requires domain knowledge from the experts.

In this chapter, we present a self-structured confabulation network which learns the structure of a probabilistic inference network from unlabeled training data. The network consists of a succinct set of nodes that represent the original features or the combinations of the features. These nodes are named lexicons because they record the symbolic representations of the possible inputs. In this chapter, we refer to node and lexicon interchangeably. The connections between nodes captures their associativity. Among the nodes, those with incoming connections are *key nodes*, which serve as the basic testing units. Given a learned network configuration, new data streams incrementally refine the weights of the connections, which are conditional probability matrices between the lexicon symbols. The non-zero entries in a matrix are referred to as the *knowledge links*. The learned knowledge bases are accessed in the recall phase to test the inconsistency, i.e. the "amount of surprises" in each key node, and the results are accumulated to make a network-wide decision. Parallel implementations are adopted to achieve computation acceleration. The proposed

framework is generalized to a wide range of applications, including road-traffic monitoring, network intruder detection and program control flow monitoring for case studies.

The key contribution in this chapter is an automatic procedure that learns the structure of a confabulation network from the incoming data.

- A case study using manually configured network is constructed to prove the concept of using cogent confabulation as anomaly detection method.
- An algorithmic analysis is presented that compares the training complexity of AnRAD with that of a neural network. It shows that the Bayesian property of AnRAD enables it to use much less training samples to achieve the asymptotic error. The analysis explains some of the design choices in network structuring and incremental learning.
- A self-structuring algorithm is proposed. The method uses a data-driven approach to decide the feature representations and their combinations in each lexicons. Then, connections between lexicons are constructed by connecting the most relevant nodes. The network node all have explicit meaning, and thus provide introspection ability for the reason of the anomalies.
- The detection accuracy of the self-structuring network is evaluated with discussed experimental setups, and compared with both traditional and neuromorphic detectors.

2.2 Related Works

Extensive studies on anomaly detection [10, 20, 21] have been carried out. Classification-based methods, such as support vector machine [47] and decision tree [1], learn a classifier from labeled training data and map a test subject into one of the classes. Density-based detectors assume anomalies are far from their neighbors. The local density has been defined by local reachability distance [13] or rank [43]. Online approaches [54, 77] are also

studied. Cluster-based techniques are unsupervised, and assume that normal data belong to some clusters while anomalous ones do not [18]. The statistical models fit the data with parametric distributions and consider anomalies occur in the low probability regions [85]. Graphic models such as conditional random fields [2] are studied to capture the spatial-temporal features [93]. AnRAD is also a graph-model-based approach. It differs from the previous works because our self-structuring algorithm enables a short and concurrent inference pipeline for parallel implementations.

Since AnRAD features bio-inspired detection mechanism, we are interested in other neuromorphic approaches for anomaly detection. Replicator neural network [37] trains symmetric hidden layers to reconstruct the input sample, and use the reconstruction error as the anomaly indicator. Self-organized map (SOM) methods [16, 83, 88] leverage competitive training to map the high-dimensional data into 2D neuron layers. Testing samples' abnormalities are ranked based on their distances to the best matching units. Growing hierarchy self-organizing maps [45, 74] are studied to overcome the static network structures. Hierarchical temporal memory (HTM) [36] is a neuromorhpic model based on cortical learning algorithm. Anomalies are identified by the percentage of active pooler columns that were incorrectly predicted [71]. Most existing methods are derivations of neural networks, but the studies on inference network are less.

2.3 Confabulation-based Anomaly Detection

2.3.1 Scoring Algorithm

We adopt *cogent confabulation* [38] as the computing model for probabilistic inference. Whenever an attribute is observed, the corresponding symbol (i.e. neuron) is activated, and an excitation is passed to the other symbols through the knowledge links (i.e. synapses). The excitation of a symbol t in lexicon l is calculated by summing up all incoming knowl-

edge links as in Equation (2.1):

$$y(t) = \sum_{k \in F_l} \{ \sum_{s \in S_k} [I(s) \ln \frac{p(s|t)}{p_0}] + B \}, t \in S_l \tag{2.1}$$

Here, t is one of the symbols of node l. F_l denotes the set of nodes that have connections to l, and S_k is the symbol set of lexicon k. $I(s)$ is the firing indicator, which takes value 0 or 1 given the occurrence of the source symbol s. p_0 is the minimum probability that is empirically selected to ensure that t has positive excitations. B is a constant band gap to favor the symbols that receive more activation from distinct lexicons. We use $B = 0$ in this work to let the abnormality determined only by the synaptic weights. The excitation level is essentially the log likelihood of symbol t given the rest of the observations. In the implementation, a knowledge link entry stores the *knowledge value* $v(s,t) = \ln \frac{p(s|t)}{p_0}$ for faster calculation during the detection. Based on this basic computation and inference model, the anomaly detection method is defined.

A set of features are selected and referred to as the *key lexicons*. They serve as the basic testing units. The other lexicons will not be tested, and are named *supporting lexicons*. Knowledge links are established from supporting lexicons to key lexicons and among key lexicons. The excitation levels of all possible symbols in a key lexicon are calculated according to function (2.1). The symbol with the highest excitation (i.e. the highest likelihood) is considered the reference symbol and will be denoted as t_{max}. Given the observed input symbol t, the *anomaly score* of a key lexicon is calculated using the following function (2.2).

$$\alpha_l(t) = \frac{y(t_{\text{max}}) - y(t)}{y(t_{\text{max}})}, t, t_{\text{max}} \in S_l \tag{2.2}$$

As shown in equation (2.2), the anomaly score is the normalized excitation difference between the observation t and reference symbol t_{max}, where t_{max} is the symbol in lexicon l that has the highest excitation level $y(t_{\text{max}})$. It reflects how low the observed symbol's cogency is compared to its context. The individual key lexicons are merged to compute the network

anomaly score by weighting the excitation levels with the priors $y^*(t) = y(t) + \ln(p(t)/p_0$. By substituting y in equation (2.2) with y^*, the network anomaly score A is obtained by averaging the prior-weighted node scores α_l^* as Equation (2.3).

$$A(t_{l=1\ldots L}) = \frac{\sum_{l=1}^{L} \alpha_l^*(t_l)}{L} \tag{2.3}$$

$$\alpha_l^*(t) = \frac{y^*(t_{\max}) - y^*(t)}{y^*(t_{\max})}, t, t_{\max} \in S_l \tag{2.4}$$

where L is the number of key lexicons. t_l is the observation symbol of node l, and the output score is ranged in $[0, 1]$.

The performance of such inference based anomaly detection largely depends on the quality of the knowledge graph, including the selection of key lexicons and the connection of knowledge links. To construct such a network manually is not trivial. First of all, it requires application specific information and such expert knowledge may not always be available. And furthermore, manual construction does not always ensure optimal network structure.

2.3.2 Algorithm Analysis

To develop self-structuring algorithm, we need first to understand the property of the confabulation model and its learning rule. As we discussed in Section 1.3, confabulation network assume independency between support lexicons, and it converges fast with smaller training set. In this section, we analyze the algorithm to justify these claims and see how the properties affect our design decision in the generic learning model.

While most existing neuromorphic methods are based on neural networks, we choose inference network instead. The motivation lies in the size of the training sample for anomaly detection. According to Zimek et al. [96], ensemble of models with smaller samples is preferable for anomaly detection compared to training a large model with all the data. Although confabulation network, who models $p(s, t)$, is supposed to have a higher

asymptotic error than that of neural network who directly learns $p(t|s)$, it also approaches the bound faster. Inference network provides the foundation for learning accurate models with small samples and performing temporal ensemble (section 2.5.5).

In equation (2.1), $p(s|t)$ is learnt by $\hat{p}(s|t) = c(s,t)/c(t)$, where $c(.)$ counts the occurrences of the event in the training set $\{s^i, t^i\}_{i=1}^{M}$. Here M is the training sample size. To simplify the analysis, let $B = 0$, $S_l = \{t, t'\}$ and $\sum_{s \in S_k} I(s) = 1, k \in F_l$. A threshold of 0 is used for equation (2.2), i.e. raising alarms whenever $t \neq t_{\text{max}}$. Let s_k denotes the only activated input symbol in the kth supporting lexicon. Assume that the observation t is abnormal, it can be detected if and only if inequity (2.5) hold true.

$$\varphi_l(s) = \sum_{k=1}^{|F_l|} \ln \frac{\hat{p}(s_k|t')}{\hat{p}(s_k|t)} > 0 \tag{2.5}$$

That said, under the input, t' is predicted as t_{max} (i.e. the expected symbol). If $\varphi_l(s) < 0$, a false negative error is generated. In the following, we focus our discussion only on how to bound the false negative error; the discussion of false positive error is similar by taking the reciprocals for each summation term in equation (2.5). With 2 symbols in the key lexicon, the anomaly detection problem is simplified to a binary classification problem. The classifier is defined $h_{\text{CFB}} : s \to S_l$ and the asymptotic version is h_{CFB}^{∞}, which is trained by infinite amount of data. We compare h_{CFB} with logistic regression h_{NN} as an example for neural networks, and use the same incoming connections F_l.

We use $\varepsilon(.)$ to denote the error rate of a mapping. According to previous studies [70], the asymptotic error rate is smaller for neural networks, i.e. $\varepsilon(h_{\text{NN}}^{\infty}) < \varepsilon(h_{\text{CFB}}^{\infty})$. However, for some constant ϵ_0, to make $\varepsilon(h_{\text{NN}}) \leq \varepsilon(h_{\text{NN}}^{\infty}) + \epsilon_0$ with high probability, we need sample size $M = \Omega(|F_l|)$, i.e. on the order of the learnt parameters.

In the case of confabulation, the learnt parameters $\hat{p}(s_k|t)$ can approach the asymptotic version $p(s_k|t)$ with less data. Let some $\epsilon > 0$, $c(t) = \beta M$ for $0 < \beta < 1$, $\delta = \epsilon\sqrt{\beta M}$, by

additive Chernoff Bound [68], for each knowledge link we have

$$\begin{aligned}
&\Pr[|\hat{p}(s_k|t) - p(s_k|t)| \geq \epsilon] \\
&= \Pr[\beta M|\hat{p}(s_k|t) - p(s_k|t)| \geq \sqrt{\beta M}\delta] \\
&\leq 2e^{-2\delta^2} = 2e^{-2\beta M\epsilon^2}
\end{aligned} \tag{2.6}$$

Since there are $2|F_l|$ such parameters, to make the union bound of the error $2|F_l|\cdot 2e^{-2\beta M\epsilon^2} \leq \rho$ for some constant $\rho > 0$, it suffices to pick $M = O(\ln|F_l|)$. In other words, with high probability, $\hat{p}(s_k|t)$ is within $\omega = O(\sqrt{\ln|F_l|/M})$ of $p(s_k|t)$.

To bound the error rate, we consider the case when h_{CFB} makes a false negative and h^{∞}_{CFB} does not (i.e. $\varphi^{\infty}_l(s) > 0$ and $\varphi_l(s) < 0$). This happens when $\varphi^{\infty}_l(s)$ (obtained by replacing $\hat{p}(s_k|t)$ with $p(s_k|t)$ in equation (2.5)) is within $(0, \omega|F_l|)$. So the difference with the ideal model can be represented by equation (2.7)

$$\varepsilon(h_{\text{CFB}}) \leq \varepsilon(h^{\infty}_{\text{CFB}}) + \Pr(\varphi^{\infty}_l(s) \in (0, \omega|F_l|)) \tag{2.7}$$

Given the normal pattern t', by the non-negativity of KL-divergence, each term of $\varphi^{\infty}_l(s)$ has positive mean and $\Omega(1)$ of them are far away from 0. So the expectation $E[\varphi^{\infty}_l(s)] = \Omega(|F_l|) = \gamma|F_l|$ for some $\gamma > 0$. Based on the assumption of confabulation model that all $p(s_k|t)$ are independent, by Chernoff Bound [68], let $\delta = (\gamma - \omega)/\gamma \in (0, 1)$, we have

$$\begin{aligned}
\Pr[\varphi^{\infty}_l(s) \in (0, \omega|F_l|)] &\leq \Pr[\varphi^{\infty}_l(s) < (1-\delta)\gamma|F_l|] \\
&< e^{-\delta^2\gamma|F_l|/2} \leq O(e^{-(\gamma-\omega)^2|F_l|})
\end{aligned} \tag{2.8}$$

which is exponentially small with respect to $|F_l|$ when ω is a constant. Therefore, by picking $M = \Omega(\ln|F_l|)$, with high probability, we have $\varepsilon(h_{\text{CFB}}) \leq \varepsilon(h^{\infty}_{\text{CFB}}) + \epsilon_0$ for some constant $\epsilon_0 > 0$.

From the analysis, we draw design insights. First of all, in order to satisfy the as-

sumption of lexicon independence, we have to de-couple the lexicons with their feature distance during the network self-structuring procedure (section 2.5). Furthermore, confabulation has higher asymptotic error for classification, but can approach it $\Omega(\ln|F_l|)$ faster. So it is beneficial to apply incremental learning where the network coefficients are learned by merging the trained results from multiple short episodes of training sequences (section 2.5.5). Before we start with the self-structuring algorithm, we first investigate a manually configured network and show the detection concept empirically. Then we will introduce the self-structuring method that extend the applicability of the inference network.

2.4 Case Study: Manually Configured Network for Vehicle behavior

2.4.1 Problem and Preprocessing

In this application, vehicle traces are obtained from an area road network. The preprocessor extracts 10 primary features, of which 5 are related to an individual vehicle (latitude, longitude, speed, direction, and vehicle type) and 5 are related to vehicle interactions (the neighbor vehicle's distance, speed, relative position, direction difference and type). The traffic records are generated at one-second sampling intervals. we focus on developing an abstract-level autonomous anomaly detection model that provides continuous monitoring of vehicle behavior. Taking advantage of the advanced sensing and imaging capability of today?s digital camera systems, our model may enable anomalous traffic situation detection for wide area traffic monitoring which is not achievable solely by human observers.

The area is partitioned into 342 detection zones. The benefits of this step are manifold: Firstly, partitioning the monitoring area into smaller zones and processing each zone independently can effectively reduce the complexity in extracting neighboring information between vehicles. Secondly, the number of possible attributes of certain features, e.g. vehi-

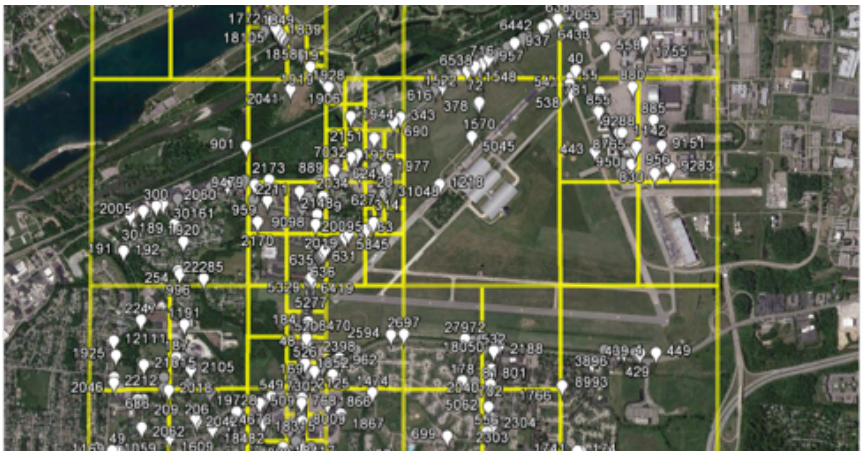


Fig. 2.1: A zone partition example

cle coordinates, is directly proportional to the area of the zones. Therefore the partitioning method effectively reduces the number of symbols in a lexicon, and reduces the complexity of the confabulation model. Thirdly, the traffic situation varies from location to location, zone partitioning helps to improve the accuracy of the model.

Because the number of vehicles directly determines the computation workload of the training and recall processes, it is used as the criteria for zone partitioning. In general the algorithm divides the zones based on the average traffic density and ensures that none of the monitoring zone has more than a maximum number of vehicles appearing at the same time slot in the training set. A training set are sent to the preprocessor frame by frame. And the counters of each zone is increased as the vehicles fall into the zone. On violating the density threshold, a zone is partitioned into four child zones. The resulting zones are organized in a sibling tree structure. An example of the resulting zone partitioning is shown in Fig. 2.1. The grids with yellow borders are detection zones.

2.4.2 Confabulation Network Structure

we consider the behavior of a vehicle within the context of its neighbors during the current and previous observations. If we define all observations made in the same time slot as a frame, the detection method involves three consecutive frames. Four classes of objects

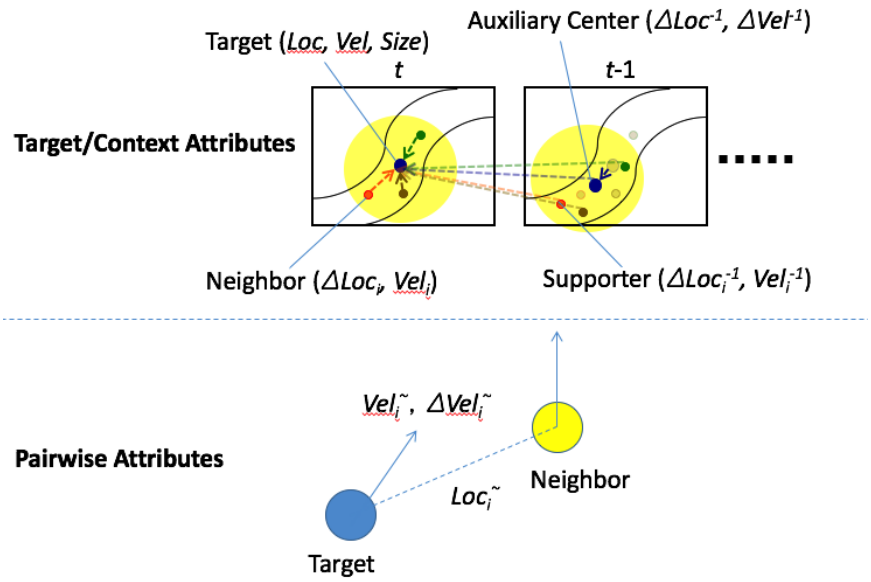


Fig. 2.2: Vehicle record and lexicons

are defined: target, neighbor, auxiliary center, and supporter. Each vehicle appearing in a detection zone at frame t is considered as a target. The ten vehicles closest to the target in the same frame are defined as neighbors. Based on the current location and speed of target, we can estimate its location in the previous frames. The estimated targets in frames $t-1$ and $t-2$ are referred as the auxiliary centers. The nearest ten neighbors of the auxiliary center in the corresponding frame are called the supporters. Fig. 2.2 shows an example of the four types of vehicle records. Network is generated based on the observations of each target within the context neighbors, auxiliary centers and supporters.

Three lexicons are used to describe the basic attributes of a target vehicle, target location Loc, target velocity Vel, and target size $Size$. The target location is expressed in geographic latitude and longitude and is discretized to levels of approximately ten meters. The target speed is expressed as the combination of ground speed and direction. The target size contains five different categories: sedan, truck, SUV, moving truck and 18-wheeler.

Two lexicons are associated with each auxiliary center, center displacement ΔLoc^{-t} and center acceleration $\Delta Vel^{-t}, t = 1, 2$. The center displacement is the distance between the target and an auxiliary center represented by the displacement in latitude and longitude.

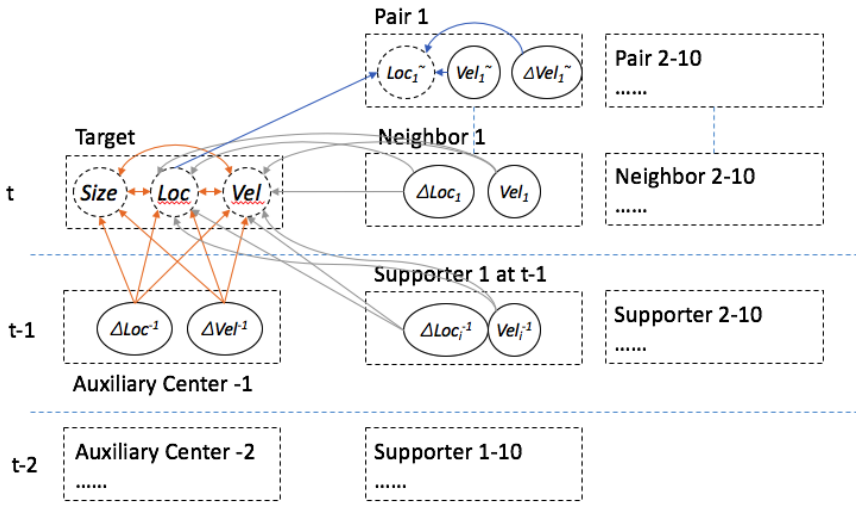


Fig. 2.3: Vehicle network structure

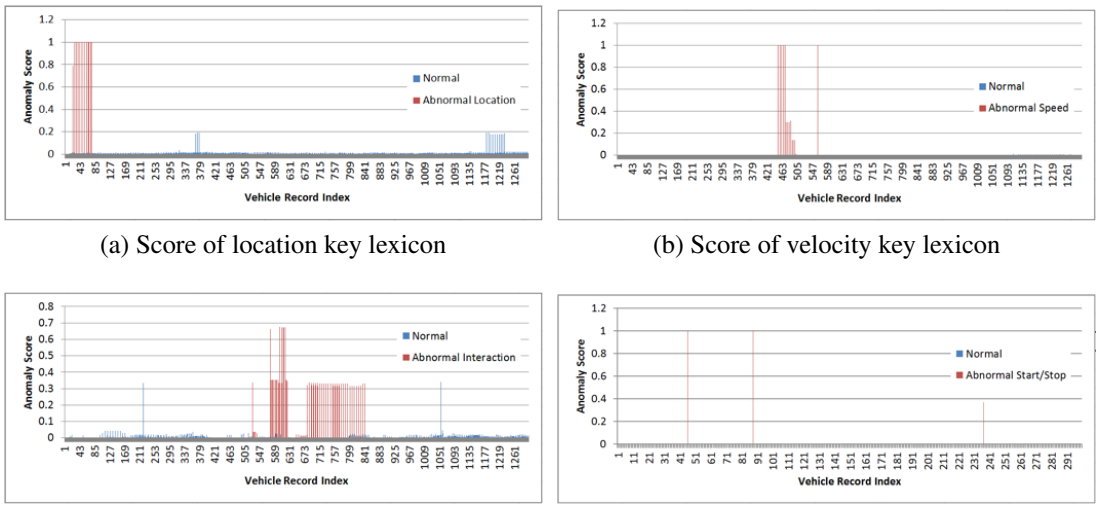
It describes how far the target moved in the last 1 and 2 frames. The center acceleration specifies the change of velocity of the target during the last 1 and 2 frames.

Two lexicons are associated with each neighbor or supporter. The relative location lexicon (denoted as ΔLoc_i for the ith neighbor and ΔLoc_i^{-t} for the ith supporter in frame $-t$) gives the relative position of the neighbor (or supporter) with the respect to the target. The velocity lexicon (denoted as Vel_i, for the ith neighbor and Vel_i^{-t} for the ith supporter in frame $-t$) specifies the velocity of the neighbor (or supporter) as a combination of the speed and direction.

Three lexicons are used for pairwise attributes that describes the relation between the target and each of its neighbors. Pairwise location lexicon Loc_i specifies the distance and direction between the target and the ith neighbor. Pairwise speed lexicon Vel_i specifies the target?s absolute speed and relative moving direction with respect to the neighbor?s direction. Pairwise speed changes lexicon ΔVel_i captures the target relative speed and relative direction with respect to the ith neighbor.

In total, 97 lexicons are used. Every vehicle in the detection zone is treated as a target; therefore there is one round of recall for each one of them.

Fig. 2.3 shows the overall confabulation model with lexicons and the knowledge links



(a) Score of location key lexicon

(b) Score of velocity key lexicon

(c) Score of the 1st neighbor pair lexicon

(d) Score of velocity lexicon: sudden stop/start

Fig. 2.4: Anomaly score trace for vehicles

among them. Lexicons S, L, V and L_i are represented using dashed circles. Each one of them corresponds to a general category of abnormal behavior of the target vehicle, such as abnormal location or speeding, inconsistency between vehicle size and its status, and abnormal interactions with neighbors. We make these lexicons as the key lexicons and others as the support lexicons. Only the excitation levels of the key lexicons need to be evaluated. All other lexicons only provide supporting context for them.

2.4.3 Detection Results

In this test, one detection zone of $500 \times 500m^2$ with moderate traffic density is selected from the monitoring area. The training data has 240 minutes of normal traffic. The testing data include 10 minutes of normal traffic data with manually inserted abnormal events representing typical hazardous vehicle activities. The abnormal events include cars deviating from the road, speeding, tailgating, 18-wheelers running at abnormal speed, and cars unexpectedly stop-and-go in the middle of the road.

Fig. 2.4 show the anomaly scores of the key lexicons for all vehicles in the testing data when abnormal events appeared. The X-axis of all the plots gives the indices of vehicles.

The Y-axis gives the magnitude of the anomaly scores. Each figure corresponds to a type of abnormal activities. The anomaly scores of the manually-inserted abnormal targets are highlighted in red in each figure. As we can see, the anomaly scores in red are significantly higher than the normal ones, indicating anomalies can be detected by a decision threshold. Furthermore, the anomaly scores demonstrate obvious temporal continuity for most categories of abnormal events, except that of abnormal stop-and-go of vehicles, which give short spikes only when the moving status changes.

From the case study, we see that confabulation model is able to detect different kinds of anomalies, given proper network configuration. However, it is also seen that the configuration of lexicons and their connections is an arduous process. We need the expert knowledge, e.g. the definition of speeding or tailgating to construct the useful lexicons. But such knowledge is not always available. What's worse, even with expert knowledge, the model is still not scalable in that it can only detect certain kinds of anomalies while novel classes of violations are not guaranteed to be captured. In the next, we will develop a data-driven structuring method to automatically define the network configurations. In this way, the model will be much more adaptive to new dataset and new anomalies.

2.5 Network Construction and Learning

The mechanism of cogent confabulation [38] is similar to that of a probabilistic graphic model. Activating the power of such simplified model usually requires carefully tuned network architecture. Instead of going deep and complicated in the network inference pipeline, AnRAD pushes the complexity to the initial structuring stage and builds hierarchical structure in the lexicon design space. This approach results in a very succinct network configuration. In fact, the anomaly inference only propagates two layers (support lexicons to key lexicons and key lexicons to score), and every excitation integration can be parallelized. The overall model is a simple but highly concurrent network built on top

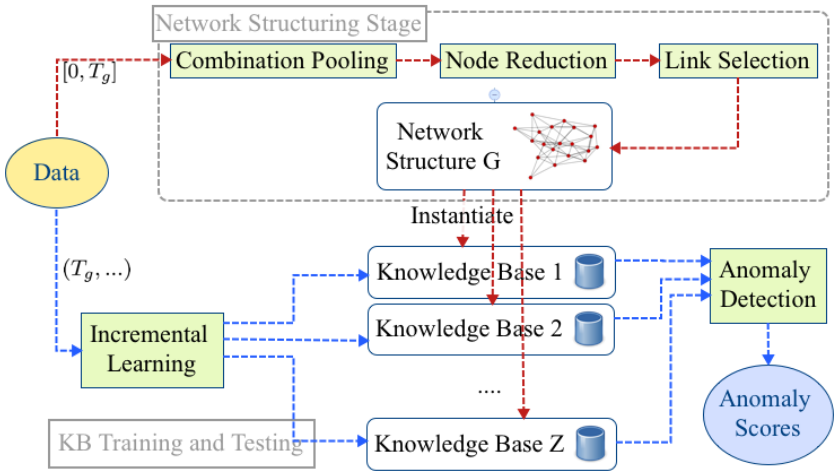


Fig. 2.5: AnRAD workflow

of unified processing elements (i.e. neurons), which is analogous to the massive parallel structure of biological neural system. The method enables fast online learning and highly concurrent detection.

AnRAD is an inference-based anomaly detection framework (Fig. 2.5), whose inputs can be represented as N streams $\{\{x_1^1, x_1^2, ..., x_1^t, ...\}, \{..., x_2^t, ...\}, ..., \{..., x_N^t, ...\}\}$. Here x_n^t represents a record tuple of the nth stream at time frame t, and consists of Q features denoted by $x_n^t(q)$. During the structuring stage, a span of the data at frame $[0, T_g]$ is sampled to construct the confabulation network G using our self-structuring algorithm. The topology captures the general correlations between the lexicons. Combination pooling finds those potentially useful feature combinations from an enormous number of possible ones; node reduction then selects a succinct set of key lexicons from the pooled candidates; the link selection connects lexicons to learn knowledge associations.

After the network is constructed, we let multiple knowledge contexts share a global network structure, and train separate knowledge bases using their local data samples. Take vehicle behavior monitoring on a large area for example, we break the area into hundreds of small zones. Data streams (individual vehicle trajectories) at time $(T_g, T_0]$ are directed to the local zones $\Psi = \{1, 2, ..., Z\}$ and used to train the initial knowledge bases

$\Theta_z^G(T_g : T_0), z \in \Psi$ (i.e. modeling $p(s|t)$'s). The knowledge bases are applied to streams from time T_0 to generate network anomaly scores for each sample. At the same time, the new incoming data continuously refine the knowledge bases $\Theta_z^G(T_g : t)$ by performing incremental learning. Typically, a moving window with size W, $\{x_n^{t-W}, ..., x_n^{t-1}, x_n^t\}$ is applied to the input stream at frame t. The anomaly detection module is accelerated by the state-of-the-art multicore processors for real-time processing. The performance of the inference network largely depends on the quality of the knowledge graph. In this section, we introduce the self-structuring procedure to construct the confabulation network.

2.5.1 Key Node Hierarchy

The confabulation model only capture the first order relation between features. A higher order relation has to be considered by adding new lexicons corresponding to feature combinations. The final structure of confabulation network consists of hierarchical lexicons where higher-level nodes are formed as the compositions of lower-level nodes as shown in Fig. 2.6. Lexicons at the bottom layer represent single primary features. These predefined primary features provide a basic description of the input data. The higher-level lexicons assemble multiple primary features; they represent more abstract meanings and combinational patterns. Since the confabulation network works at the symbolic level, continuous features are discretized using equal-width bins before mapping to symbols in lexicons. The composition process is applied to both the feature and the temporal domains. For example, there may be feature composition $\langle x_n^t(q), x_n^t(q')\rangle, q, q' \in Q$, or temporal composition $\langle x_n^t(q), x_n^{t-\Delta t}(q)\rangle, \Delta t < W$. Hence, temporal patterns are also learned and checked.

Such layered feature composition provides direct mapping from the feature space to nodes in the knowledge graph, but its complexity would quickly scale to an intractable size as Q and W increase. To reduce the complexity and improve the accuracy, AnRAD adopts the pooling-and-reduction procedure to construct network structure.

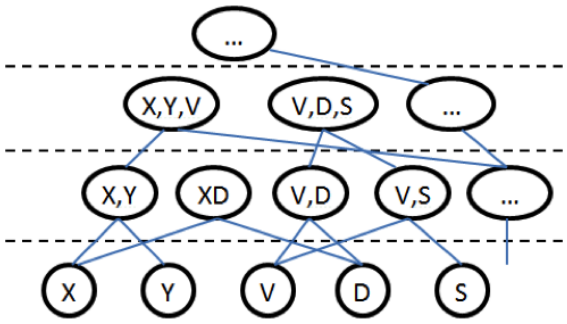


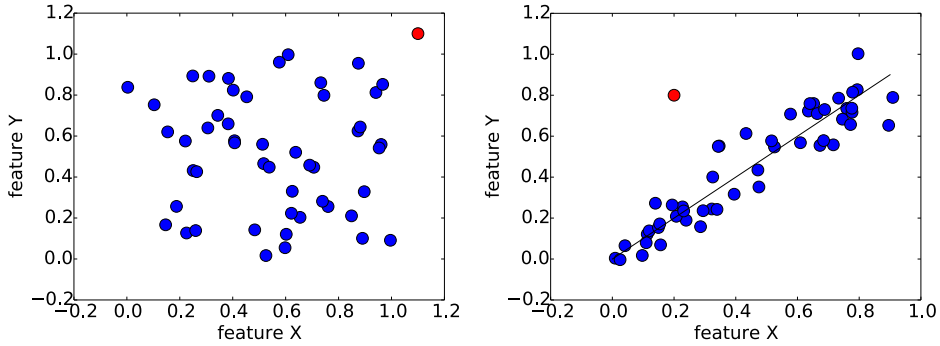
Fig. 2.6: Hierarchical Structure Example

2.5.2 Feature Combination Pooling

As mentioned previously, we complement the primary features with a set of composite features to capture higher order associations. We refer this step as feature pooling. The pooling stage generates a set of lexicon candidates, which will be reduced as discussed in the next section.

Take a simple two-feature combination for instance, the first question to ask is whether such combination provides more information for anomaly detection than the individual feature components. Consider the example scatter plot in Fig. 2.7, where the X and Y axes represent the dimensions of the two primary features. If the two features are distributed independently in their feature space as those blue dots in Fig. 2.7a, a potential outlier (the red dot) in this subspace can be detected by considering only one of the components. Therefore combinations of non-correlated features do not offer additional information. However, if the two features are sufficiently relevant as shown in Fig. 2.7b, the red dot, which is originally indistinguishable from any single axis, will be detected by their combination. Based on this observation, the pooling procedure is design to keep the combination of highly correlated features.

To extend this concept to more general cases, feature distance $d(q_i, q_j) = [1-\text{MI}(q_i, q_j)] \in [0, 1]$ is defined, in which $\text{MI}(.)$ calculates the normalized mutual information between feature q_i and q_j. The smaller the distance is, the more correlated the two features q_i and q_j are. For combination Q_l consisting of two or more features, a simple relevancy test defined in



(a) Uncorrelated Components (b) Correlated Components

Fig. 2.7: Relevant Feature Example

equation (2.9) is performed to determine whether it should be included in the set of lexicon candidates:

$$\text{RT}(Q_l) = \prod_{q_i, q_j \in Q_l} I[d(q_i, q_j) < d_{prox}] \tag{2.9}$$

where $I(.)$ is the identity function that equals to 1 when the test result is true.

This test requires all the component pairs in Q_l to be sufficiently close to each other. And d_{prox} is a constant proximity distance that defines the largest distance that is considered relevant. Algorithm 1 is used to pool the features for lexicon generation. The algorithm first adds all the single features into the candidate set. Then in the second for-loop, each subset of Q whose cardinality is less than *max_order* is inspected. If the subset passes the relevancy test, a new lexicon candidate will be added for it. Not all candidates will be key lexicons whose anomaly score will be calculated. A reduction stage is used to select the key lexicons from the candidate set.

2.5.3 k-NN Node Reduction

Although the pooling process excludes most of the irrelevant combinations, the number of possible candidates may still be large if Q has many features. Therefore, a reduction procedure is used to further compress the candidate set to generate key lexicons. The redundancy among candidates selected in the pooling stage should be removed during the

Algorithm 1 Feature Combination Pooling

1: **procedure** pool(Q, max_order): # Q: the complete feature set; max_order: the maximum combination order
2: $CS \leftarrow empty\ set$
3: **for** each feature $q \in Q$:
4: add $\{q\}$ to CS
5: **for** each $Q_l \subset Q$ and $|Q_l| <$ max_order:
6: **if** all $Q_{l'} \subset Q_l$ was accepted and $RT(Q_l)$ passed:
7: add Q_l to CS
8: **return** CS # feature combination candidate set

reduction. Because labels are not available in the training set, a similarity-based method [67] is modified to preserve the most representative combinations.

The general idea of the reduction procedure is to cluster the candidate feature combinations by their similarity, and then select one representative from each of the clusters. Again, normalized mutual information is employed to measure the distance $d(Q_{l1}, Q_{l2})$ between the combinations. The clustering process is accomplished by k-NN (k nearest neighbor) principle. While the most compact candidate is selected from a cluster, its neighbors will be discarded. This operation repeats until the remaining candidates cannot form any cluster. The reduction procedure is described in Algorithm 2. The algorithm first initializes the set *KEY* with all candidates. Then it calculates the distances from each combination to its nearest neighbors. The center of the compact cluster has its k-distance selected as the upper limit of cluster radius. Then in the following while-loop, the combination with minimum k-distance is selected and has its K neighbors removed from the *KEY* set. Then the K value is reduced until the next cluster would have a smaller radius than the radius limit. The neighbor-removing process repeats until K reaches 1. The remaining candidates in *KEY* set are the final nodes selected.

Although we use the features in Q as an example to explain the pooling-reduction procedure, the concept in Section 2.5.2 and 2.5.3 can be applied to temporal domain as well. When the data inputs are not just single points in the feature space but multivariant time series, the definition of anomalies may extends to historical patterns. To

Algorithm 2 Node Reduction

procedure knn(CS, K): # CS: pooled candidate set; K: the initial K
KEY $\leftarrow CS$
for each combination $Q_l \in CS$:
    for $Q_k \in K$ nearest neighbor of Q_l:
        Q_l.dist[k] $\leftarrow d(Q_l, Q_k)$
find Q_0 who has the smallest k-distance
max_err $\leftarrow Q_0$.dist[K-1]
while $K > 1$:
    find Q_r who has the smallest k-distance
    remove Q_r 's K nearest neighbor from KEY
    radius $\leftarrow$ min(Q_x.dist[$K-1$]) for $Q_x \in$ KEY
    while radius $>$ max_err:
        $K \leftarrow K-1$
        radius $\leftarrow$ min(Q_x.dist[$K-1$]) for $Q_x \in$ KEY
return KEY # key node set

capture such potential outliers, the key lexicons must include not only different features, but also feature projections in different frames. This can be accomplished by performing feature-wise selection followed by temporal selection. If multiple frames are considered after the key lexicons being built, each lexicon along with its historical readings form a new temporal feature set $Q_l^W = \{Q_l^0, Q_l^{-1}, ..., Q_l^{-W}\}$. The same pooling-reduction algorithms can then be applied directly on these feature sets to generate informative and succinct key lexicons. A key lexicon is represented as a two-dimensional pattern $R_l \sim [(q_{l_1}, q_{l_2}, ..., q_{l_i}, ...)^{-t_1}, (..., q_{l_i}, ...)^{-t_2}, ... \ (..., q_{l_i}, ...)^{-t_j}, ...]$. Furthermore, the number of correlated frames W is usually much smaller than the feature number in Q, so the reduction process may sometimes be omitted in temporal selection.

2.5.4 Link Selection

Now that the key lexicons are identified, the next step is to find the supporting lexicons that can be used to infer the key lexicon symbols. To do so we follow a max-similarity, min-redundancy principle. For instance, to infer the shape of an object, touching is preferable compared to color (max-similarity). But if touching is already selected, weighting

might not be necessary as they share some information (min-redundancy). Generally, we want to maximize the correlation between key lexicons and their supporting lexicons, and meanwhile, minimize the correlation among the supporting lexicons that are connected to the same key lexicon. The supporting lexicons are chosen from the primary features since the key lexicons have already handled the combinational patterns.

Algorithm 3 Link Selection

```
1: procedure select_links(R, F):        # R: target node; F: set of single features
2: SUPP ← empty set
3: ranks ← F.sort(key=d(R, q ∈ F))
4: for feature q ∈ ranks:
5:     if q ∈ R or d(R, q) > (1 − d_prox):
6:         continue                     # low similarity
7:     if any p ∈ SUPP has d(p, q) < d_prox:
8:         continue                     # high redundancy
9:     add q to SUPP
10: return SUPP                         # support nodes for R
```

Heuristic Algorithm 3 finds a group of features at certain time offset, $\{q^{-t}, q \in Q, t < W\}$ which infer the observation at key lexicon R_l. The algorithm first sorts the supporting features by their distances to the target key lexicon. Then it traverses the sorted features, adds a primary feature to the supporter set only if: (1) it is not one of the components of the key lexicon; (2) it is highly correlated with the key lexicon; and (3) it has low correlation with supporting features that has already been selected for the same key lexicon.

At this point, the network is properly configured. Note that the structuring stage only configures the connections among the lexicons, but the knowledge links of the connections (between symbols) are established and strengthened during online training to build local knowledge bases.

2.5.5 Incremental Learning

Given the configured network structure, AnRAD constructs the knowledge bases from the input data. It could simply count co-occurrences $c(s, t)$ from all the training samples, which

works fine for prediction tasks. However, increasing training set size does not necessarily give equally good performance for anomaly detection. Based on the analysis in section 2.3.2, our framework uses incremental learning with episodes, in which the co-activation counters reset after every time period T. The knowledge values stored in the knowledge bases are updated by merging the new conditional probability into the previous episodes.

$$v^{\Lambda+1}(s,t) = \frac{v^{\Lambda}(s,t)\Lambda + \ln[p(s|t)/p_0]}{\Lambda+1} \tag{2.10}$$

$$v^{0}(s,t) = \ln(\frac{p(s|t)}{p_0}) \tag{2.11}$$

v^{Λ} is the stored knowledge value at episode Λ. It can be substituted to excitation $y(t) = \sum_{k\in F_l}\{\sum_{s\in S_k}[I(s)v^{\Lambda}(s,t)]+B\}$. In the first training episode, i.e. v^0, the knowledge value is calculated the same as in equation (2.1). Essentially, this updating function works as an ensemble of temporal sub-samples.

2.6 Evaluations

2.6.1 Datasets

To evaluate the effectiveness of the framework, we investigate three different datasets.

Vehicle traces dataset This dataset is the same one that we used in Section (2.4). Because we upgrade the confabulation network to a self-structured one, the manually constructed feature combinations are no longer necessary. Instead, only the primary features are fit the model for structuring and knowledge base learning.

DARPA intrusion detection dataset [59]. The dataset is generated from DARPA 1998 tcpdump files. For each IP address pairs, traffic statistics are recorded per 300ms-frame. In total, 21 primary features are extracted from the raw dump files. Some examples of the features are bytes from client (or server) to server (or client), service ports, etc. We do not use the session-oriented KDD 99 dataset [97] because we would like to investigate

concurrent data streams rather than session-oriented data points, and our processing also leverages less attack specific domain knowledge. For training, normal streams from the seven weeks of training data are randomly sampled and about 20000 frames are selected. The negative class for test has another 7000 streams, and all the attacks (422 streams, 24 categories) in the seven weeks form the positive class.

ADFA-LD [28,29]. The dataset contains system call sequences of benign and malicious programs on Linux workstations. A clean training set containing around 10000 system calls are sampled. Also, we build another tainted training set of about 50000 system calls, among which 1/5 of the samples are randomly extracted from the attacks and treated as normal during training. This is to evaluate the detection performance under non-ideal or compromised training set. The testing data have about 6000 programs from the normal validation set and 746 programs from the attack set.

2.6.2 Comparison Methods

Incremental local outlier factor (LOF) [77]. It is the incremental version of the classical density-based LOF detector [13]. Given a testing subject, the method uses the ratio between the neighbors' local reachability distances and that of the testing sample's as the anomaly indicator.

Cross-feature analysis (CFA) [15]. This is a fast rule based unsupervised detector. For each feature, the method builds a CART decision tree and use the other features to infer the probability of the target feature. Then the probabilities from all trees are summed to indicate the abnormality.

For neuromorphic baselines, we reproduce the followings.

Replicator neural network (RNN) [37]. Similar to an auto-encoder, the method builds a symmetric 5-layer neural network and use back-propagation to minimize the reconstruction error. The mean squared error between the input features and the reconstruction outputs are used for the anomaly score.

Self-organizing map (SOM). The neural network uses competitive learning to map the high-dimensional data to a 2D neural layer. It has used for anomaly detection in various fashion [16, 83]. In this baseline, we use the sum of input's distances to its nearest BMU's (Best Matching Unit) to detect anomalies [88].

Hierarchical temporal memory (HTM) [36]. The emerging neuromorhpic model based on cortical learning algorithm and sparse coding identifies anomaly by the percentage of active spatial pooler columns that were incorrectly predicted by the temporal pooler [71]. The method works in streaming fashion and does not require sliding windows for the input sequences. For multi-featured dataset (DARPA), we train predictive models for each feature, and generate the anomaly score by summing up scores from each feature model. All hyper-parameters are selected using their in-package swarm algorithm.

All evaluation methods output anomaly scores for each data frame. We use leaky bucket to make anomaly decision from the score sequences. Whenever a score exceeds a threshold, we add 1 unit to the bucket. Otherwise, 1 unit is leaked from the bucket. An anomaly sequence is reported if the bucket overflowed. We vary the score threshold to tradeoff between detection rate and false alarms and also analyze the sensitivity of the bucket size. The comparative studies are conducted on the fully labeled public datasets (DARPA and ADFA-LD). For methods requiring continuous features (LOF, RNN and SOM), one-hot encoding is applied to the system calls. Different AnRAD configurations are tested on all three datasets.

2.6.3 Vehicle Behavior Detection

The self-structuring procedure picks 44 key lexicons out of 2548 possibilities given $max\_order = 5$ (feature-wise pooling) and $= 3$ (temporal pooling). The training stage consumed 240 minutes of traces and another 10-minute trace is used as the test set. Among the test data, there are 179 vehicles without intentional modification used as negative cases, and 22 manually created anomalies of different categories are used as positive cases.

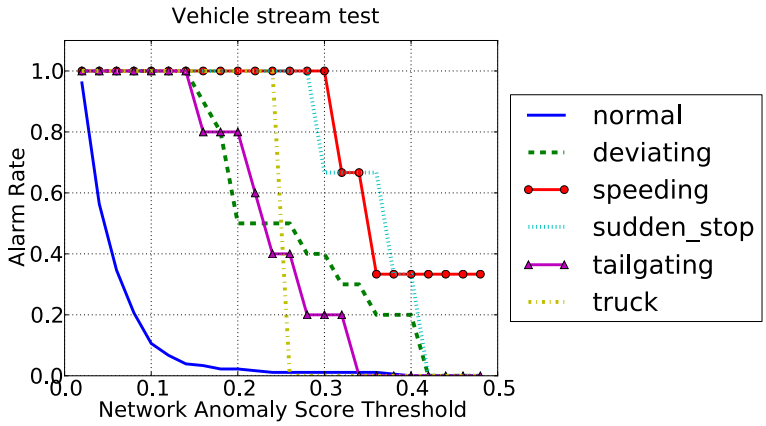


Fig. 2.8: Vehicle Detection Result

Table 2.1: Correlation between anomaly types and outstanding nodes

Anomaly	**Top 3 Outstanding Nodes**
sudden stop	**1.** ⟨speed⟩; **2.** ⟨neighbor(1).speed⟩; **3.** ⟨neighbor(1).distance⟩
speeding sedan	**1.** ⟨speed⟩; **2.** ⟨neighbor(1).speed⟩; **3.** ⟨neighbor(1).distance⟩
tailgating	**1.** ⟨speed, neighbor(1).distance⟩; **2.** ⟨longitude, latitude, speed, direction⟩; **3.** ⟨longitude, speed, direction⟩
deviating from driveway	**1.** ⟨longitude⟩; **2.** ⟨longitude, latitude, neighbor(1).direction⟩; **3.** ⟨longitude, latitude, direction⟩
speeding truck	**1.** ⟨vehicle_size, longitude, longitude^{-2}⟩; **2.** ⟨latitude, speed, neighbor(1).distance, neighbor(1).speed⟩; **3.** ⟨vehicle_size, longitude, longitude^{-1}⟩

Fig. 2.8 shows the detection results of anomaly classes. The Y-axis is the alarm rate and the X-axis is the network anomaly score threshold. It is observed that the normal vehicles generate a much lower alarm rate compared to abnormal ones. When threshold is 0.14, all abnormal vehicles are labeled positive by AnRAD while the false positive rate is at the order of 10e-2. Therefore, for vehicle anomaly detection tasks, the framework leaves a wide margin to trade between detection and false alarm.

AnRAD provides the reasoning ability, in that the positive labelings can be explained by introspecting the anomaly scores of the key lexicons. For instance, Table 2.1 shows the

Table 2.2: AUC scores for local knowledge bases

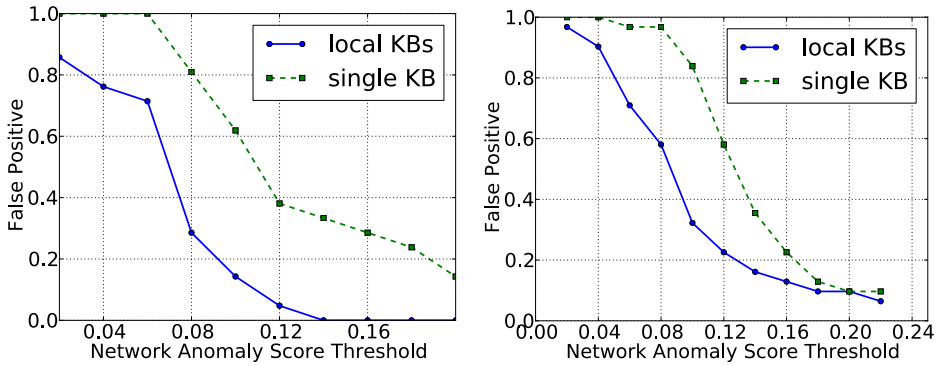
Context	**Zone 1**	**Zone 2**	**Zone 3**	**Zone 4**
AUC	1.0	0.968	1.0	1.0

relationship between the key lexicon scores and the manually annotated anomaly classes. In this example, key lexicons generating an anomaly score higher than 0.8 are defined as *"outstanding"*. We count the outstanding occurrences and sort the top three lexicons for each annotated class. For speeding and sudden stops, such anomalies are closely related to vehicle speed; our analysis also shows that the most outstanding lexicon for this type of anomaly is $\langle$speed$\rangle$. Tailgating happens when one vehicle fast approaching another, so it can be explained by that the composite lexicon of speed and distance to the first neighbor has an increase in its anomaly score. Anomalies such as deviating from the road are usually coupled by high anomaly scores in coordinates-related lexicons. Finally, trucks may be caught speeding even if this speed was normal for a sedan. Such behavior causes high scores at the composite lexicons of vehicle size and the displacement in consecutive frames. The example shows that the AnRAD framework can provide explanation to its positive results without training labels or domain knowledge.

AnRAD trains local knowledge bases for different zones. To evaluate the effectiveness of the design, we pick four zones out of the whole area. A zone contains 20 to 70 normal vehicles during the 10-minute test period. To each zone, 5 or 6 synthetic anomalies (speeding, deviating from roads, tailgating, etc.) are inserted randomly.

We first build separated local knowledge bases for each zone using their own vehicle traces. We collect the ROC (Receiver Operation Curve) AUC (Area Under Curve) scores generated from the test set. As shown in table 2.2, scores of 1.0 or almost 1.0 are achieved in the four zones. The closer the AUC is to 1.0, the better the method is during the tradeoff between true detection and false alarm. The localized training is very effective with the vehicle traces.

We then compare building a single large knowledge base using traces from all four



(a) Zone 1 (b) Zone 2

Fig. 2.9: Comparison between local and single knowledge bases

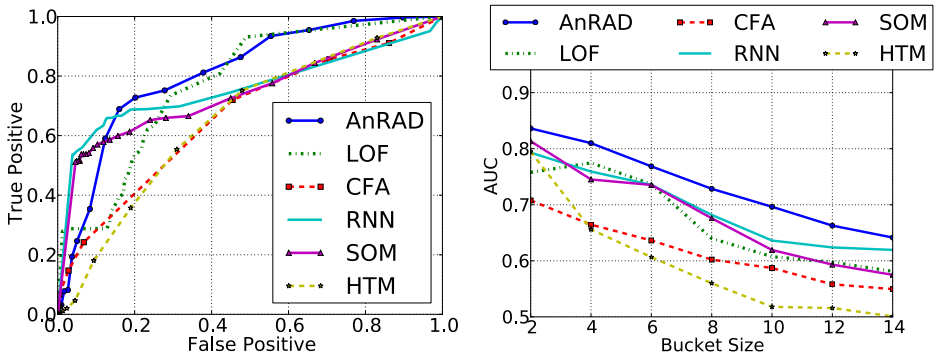
zones. While the single knowledge base can also achieve high detection rates on the synthetic anomalies, the localized training method responds better to the normal vehicles in all zones. In Fig. 2.9, The Y-axis gives the false positive rates when the detection rate is 1.0, and the X-axis specifies the network anomaly score threshold. The false positive rates using the local knowledge bases are about 40% lower than that of the single knowledge base at the same thresholds.

2.6.4 Comparative Evaluations

For the DARPA dataset, the ROC's for the comparison methods are plotted in Fig. 2.10a with X-axis representing the false alarm rate and Y-axis representing the true detection rate. Note that the true positive rates are averaged across anomaly categories to prevent the result from being biased by some large classes. The decisions are generated from the scores using leaky bucket of size 4 for all methods. LOF outperforms CFA because it works better with continuous features. Both neural network methods, RNN and SOM obtain similar curves and perform better at low-false-positive region. HTM does not adapt well because the original models are tuned for single features, and they probably should not be linearly combined here. AnRAD outperforms the baselines especially at operation region with high detection rates. It also achieves the best AUC score as shown in Table 2.3 column 2.

Table 2.3: AUC scores for different detectors

Methods	DARPA	ADFA-LD clean	ADFA-LD compromised
LOF	0.775	0.849	0.770
CFA	0.665	0.814	0.824
RNN	0.759	0.799	0.734
SOM	0.745	0.820	0.746
HTM	0.656	**0.905**	**0.872**
AnRAD	**0.810**	0.888	**0.872**



(a) ROC curves with bucket size 4 (b) AUC with different buckets

Fig. 2.10: Results on DARPA dataset

Since leaky bucket are used for all approaches, the bucket size plays an important role in the output. We also analyze the sensitivity of the detection with this parameter. In Fig. 2.10b, the X-axis varies the bucket sizes and the Y-axis marks the AUC scores of the comparison methods. For DARPA dataset, all approaches favor a smaller bucket, because the abnormal pattern of an intrusion usually occurs within a short time. The relative standings between the methods do not change much with different buckets.

For ADFA-LD dataset, the moving window size is set to 20 consecutive calls as suggested in Hofmeyr et al. [41]. Fig. 2.11 plots the mutual information (Y-axis) between a system call and its previous calls with different offsets (X-axis). The red line denotes the mutual information of the call with offset 20, after which the mutual information does not vary much. So the window size is acceptable since the work does not focus on building an optimal intrusion detection system.

Fig. 2.12a shows ROC curves when using clean training set and bucket size 4. HTM

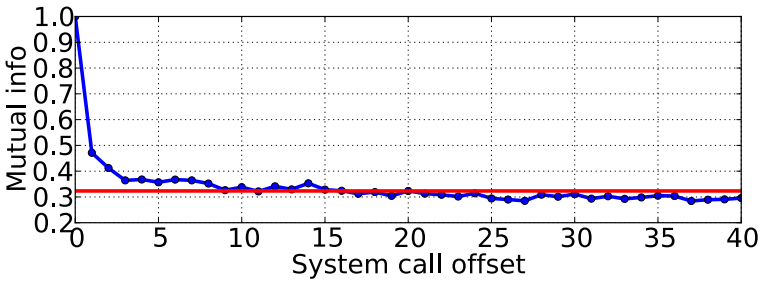
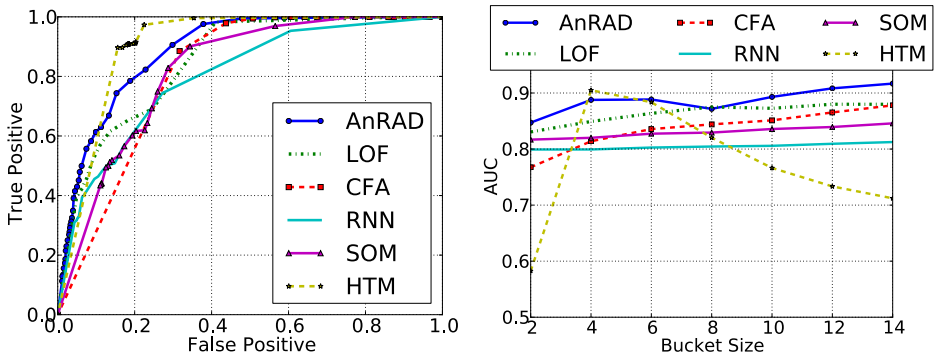


Fig. 2.11: Mutual Info between ADFA-LD system call and its previous calls



(a) ROC curves with bucket size 4

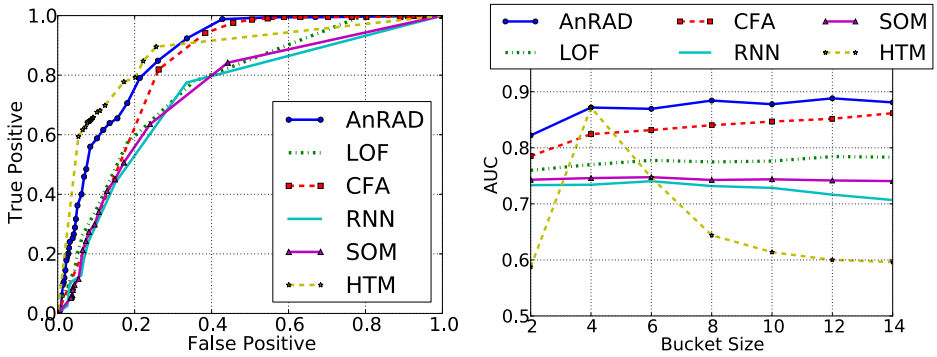
(b) AUC with different buckets

Fig. 2.12: Results on ADFA-LD using clean training data

has the best result in this case. Such univariate data do not fully exploit the lexicon compositions, but AnRAD still achieves competitive result in Table 2.3 column 3. Against different bucket sizes in Fig. 2.12b, AnRAD has overall best AUC scores with most of the bucket settings. HTM outputs bi-polar scores (either near 1 or near 0) alternatively in this dataset, thus more sensitive to the buckets.

For compromised ADFA-LD training data, AnRAD uses the incremental learning algorithm to counter the effects. In Fig. 2.13a, the neural networks and LOF do not perform well in tainted training data. AnRAD only suffers a small accuracy degradation according to Table 2.3, and it is advantageous at high-true-positive region. In Fig. 2.13b, AnRAD is stable and leads the AUC scores over different bucket sizes.

The results show that AnRAD's detection accuracy is competitive or superior to the comparison methods.



(a) ROC curves with bucket size 4 (b) AUC with different buckets

Fig. 2.13: Results on ADFA-LD using tainted training data

2.6.5 Effects of Self-structuring

In this section, we present experiment data to show the advantage of using self-structured network. In this experiment, the reference designs are a set of randomly generated networks with the same size (number of key lexicons and connections) as the self-structured AnRAD network. The reference design is trained and tested in the same way as in AnRAD.

For the DARPA dataset, the self-structured network consists of 123 key lexicons and 2421 connections. 10 networks with this same size are randomly generated and their average performance is reported. It is observed in Table 2.4 that the self-structuring algorithm has better AUC scores that outperform random network by around 17%. Similar experiments have been carried out for ADFA-LD dataset. The networks have 40 key lexicons and 410 connections. Table 2.4 shows that the self-structured network has slightly better performance than the random network with both clean and compromised training data. The advantage is smaller than that of DARPA because system calls within the windows are more correlated, so even random generated feature combinations could often benefit the detections.

Finally, Table 2.5 summarize the impact of the self-structuring algorithm to the complexity of the network. The first two rows gives the maximum order of composition in the feature and temporal domain. The rest of the rows give the potential number of nodes

Table 2.4: AUC scores for different network structures

Methods	DARPA	ADFA-LD clean	ADFA-LD compromised
Random	0.711	0.821	0.810
AnRAD	**0.810**	**0.888**	**0.872**

Table 2.5: Network complexity impact of self-structuring

Datasets		DARPA	ADFA-LD	Vehicle
Composition max order	Feature	5	1	5
	Temporal	5	5	3
Key node selection	Potential	446320	21700	2548
	Selected	123	40	44
	Reduction	99.97%	99.82%	98.27%
Connections for selected nodes	Potential	12915	800	1320
	Selected	2421	410	570
	Reduction	81.25%	48.75%	56.82%

and connections without reduction and the actual number of nodes and links after reduction. In all three datasets, our self-structuring technique achieves significant network size reduction.

2.7 Conclusion

We have presented a self-structured confabulation framework that provides real-time anomaly detection for data streams. The framework learns the application-specific configuration of network hierarchy from the data. Results show competitive detection accuracy and reasoning ability without the aid of training labels.

CHAPTER 3
PARALLEL OPTIMIZATION FOR INFERENCING CONCURRENT ANOMALOUS DATA STREAMS

3.1 Introduction

Data streams are continuous and non-stopping, which requires the machine learning algorithm to learn from the new data in one pass and to perform real-time recall at the same time. In this section, we continue to studies in confabulation network based anomaly detection system, while we shift the focus to the parallel implementation of such network. Although many anomaly detection techniques have been proposed to address the quality of computation issues, few of them are specially tuned to harness the ever-growing computing power of the parallel architectures.

This chapter present the parallel implementation of the AnRAD framework on heterogeneous multicore platforms. It provides prompt anomaly detection in the data streams, and it exploits the massively parallel structure of the neuromorphic inference model. In the previous chapter, we address the self-structuring problem of building the inference net-

work. It should be noted that our method does not only produce a network configuration that efficiently captures the data distribution, but also offers desirable characteristics to the parallel processing. By pushing the complexity to the feature (lexicon) space during the self-structuring stage, AnRAD is able to perform accurate detection with simple network topology, which enables fast online learning and high concurrency. From the implementation perspective, the framework is inherently parallel: networks constructed for different testing instances can be processed independently; within each network, the inconsistency tests of different key nodes can be processed in parallel; for each key node, the likelihoods of all possible observations are also assessed concurrently. The parallelism in different layers can be exploited by the state-of-the-art many-core processors to offer computation acceleration and model scalability.

We exploit high performance computing architectures to enable real-time performance and scalability. The main contributions are summarized as the following:

- We extend the self-structured network, which handles single data stream and single normal model, to a more general framework that monitors concurrent streams that follows diversified behavior patterns.

- The complexity of the recall algorithm is analyzed. Fine-grained parallelization as well as efficient memory layout designs on different multicore architectures are implemented and benchmarked. Over 1000x speedup over CPU program is achieved.

- The parallelization of workload is extended to support large volume of vehicles in hundreds of detection zones. Up to 16000 subjects can be handled in real-time with a commodity co-processor.

- The tradeoff between computation speed and power consumption is tested and analyzed.

3.2 Related Works

In recent years, the combination of machine learning and emerging computing architectures has received wide attention [7]. The Intelligent Text Recognition System [79] explored the confabulation [38] network's application of machine reading on heterogeneous platform with IBM cell processors. Ahmed et al. [3] further accelerated the pattern matching stage of the system with Intel Xeon Phi co-processor. However, the association stage still stayed with CPU-based multi-threading. Our previous work extends the theory to anomaly detection [24–26]. But it only considered single data stream with single knowledge context. The Bayesian network method, as a kin to cogent confabulation, was also studied from the HPC perspective. Linderman et al. [58] focused on accelerating the Bayesian network learning process on GPUs. Also, Naive Bayes classifier with testing-instance-level parallelism is studied [90]. Besides the efforts on HPC systems, neuromorphic chips featuring non-von Neumann architecture [32, 65] provide efficient cognitive computations.

3.3 Complexity Analysis

Chapter 2 demonstrates the framework's efficiency in anomaly detection. But from the aspect of sequential computation complexity, AnRAD is not superior to the other approaches during the recall. This section further leverages the highly concurrent inference structure of AnRAD to deploy the framework in a real-time and computation-intensive scenario using many-core systems.

Accelerating the algorithm requires understanding the bottleneck first. In table 3.1, the computation times consumed by different methods on the same DARPA data stream are collected. The programs are all single-threaded with median optimization effort. In terms of training, AnRAD is much faster than the others because at each frame, it only updates a single entry in the corresponding knowledge link matrix. Therefore, real-time processing and incremental training can be achieved without much optimization. However, the de-

Table 3.1: Complexity Analysis

Model	**Training time per frame**	**Training complexity N samples**	**Recall time per frame**	**Recall complexity per frame**
LOF	4854.9ms	$O(QN^2)$	684.1ms	$O(QNlogN)$
RNN	2916.9ms	$O(TNS)$	0.27ms	$O(S)$
CFA	4.30ms	$O(QHN)$	7.78ms	$O(QH)$
AnRAD	1.57ms	$O(NLF)$	243.1ms	$O(LDF)$

Q – number of features; N – number of training samples; T – neural network iterations; S – neural network connections; H – decision tree height; L – key lexicon numbers; D – average # of symbols in a key lexicon; F – average # of active knowledge links connected to a symbol.

tection function of AnRAD is merely faster than the incremental LOF, whose complexity scales with the volume of the training samples. Fortunately, the confabulation network has layered and massive parallel structure, which can be exploited for acceleration.

According to equation (2.1) and (2.2), the complexity of processing one testing instance is $O(LDF)$, where L is the number of key lexicons, D is the average number of symbols in one key lexicon, and F is the average number of activated knowledge links connected to one symbol. At the node level, each key lexicon works as an independent test, so the calculation of L anomaly scores can be distributed in parallel to multiple computing elements (e.g. CUDA blocks). At the symbol level, D and F were induced from accumulating the value of knowledge links connecting to each candidate symbols. Such computation can be made parallel by mapping different links to different CUDA threads or vector processors.

In terms of space complexity, the AnRAD model is dominated by $O(LFU)$ in which U is the average size of the knowledge link matrices. The actual space consumption can be lower. For example, features such as "shared links" in Section 2.4.2, which share the knowledge matrices for links with similar context meanings (e.g. links from different neighbor vehicles to the target vehicle), may be adopted to reduce knowledge base size. Because most of the connections are sparse matrices, a compact storage format is preferable.

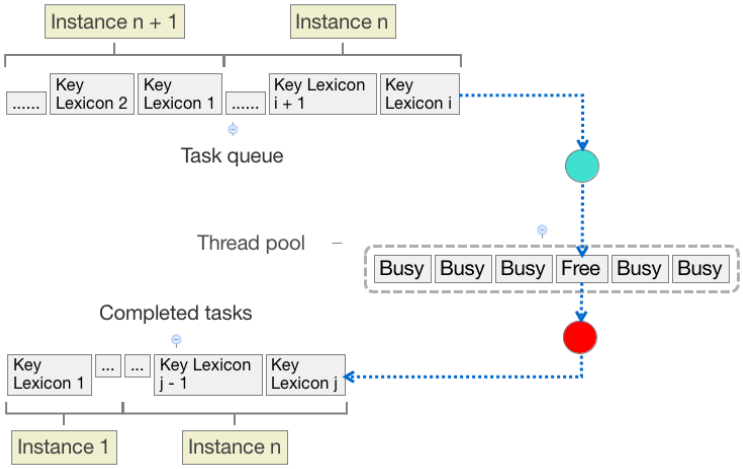


Fig. 3.1: Thread pool for CPU multi-threading

3.4 Acceleration with CPU Multi-threading

Today's general purpose processors usually have multiple cores with simultaneous multi-threading capability. A straightforward parallel implementation is to map each evaluation of Equation (2.4) to one thread. Because the computation of different anomaly scores are independent, such implementation requires almost no synchronization or inter-thread communications. Besides, the number of key lexicons would at least be a few dozens, so the amount of parallelism is usually sufficient to achieve a high utilization of the CPU cores.

As shown in Fig. 3.1, a thread pool is allocated with a maximum number of simultaneous threads. A workload dispatcher assigns key-lexicon computations to the available threads, or waits if all the worker threads were occupied. To prevent the overhead of context switching, the pool size is no larger than the maximum number of simultaneous threads that can be supported by the processor, i.e. (SMT*cpu_core_number).

Obviously, the limited number of CPU cores prevents us from fully exploiting the structural parallelism of AnRAD. Although the computation of key lexicons are parallelized, there are usually a few hundreds of such lexicons, which is already much more than the core number of a server-level CPU. Our thread pool approach avoids frequent context switching,

Algorithm 4 Naive Recall Kernel

```
1: procedure naive_recall(KB, IN):          # KB: knowledge base; IN: input symbols
2: ref ← 0
3: for symbol t ∈ KB.lexicons[threadIdx.x]:
4:     initialize y
5:     for kl ∈ KB.lexicons[threadIdx.x].links:
6:         y += kl[t][IN[link.input_idx]] + band_gap
7:     ref = max(ref, y)
8:     obs = y if t == IN[threadIdx.x]
9: Score[threadIdx.x] = (ref - obs) / ref
```

but we would expect higher performance to handle large workloads.

3.5 Fine-grained Parallelization on GPU and Xeon Phi

3.5.1 Inefficient Implementation

General Purpose Graphics Processing Units (GPGPUs) provide a potential option to fully parallelize the key-lexicon computations, because even a low-end GPU has more cores than a state-of-the-art CPU. A simple design is to directly replace each CPU thread with a GPU thread using kernel Algorithm 4. Each CUDA thread handles one key lexicon. This design may introduce following two major problems.

First of all, it gives inefficient knowledge base management. The knowledge base is a set of sparse matrices. Certain type of compression is required for efficient storage. In the original CPU implementation, the knowledge link matrices are stored in hash tables, which provide efficient memory usage and $O(1)$ lookup time. However, massive concurrent random accesses from GPUs will severely degrade the cache performance and induce a lot of stalls. Secondly, it will lead to imbalanced workloads distribution among threads. The number of symbols in different key lexicons are determined by the nature of the targeted application and they vary in a wide range. Such workload imbalance may produce serious control divergence since the CUDA threads are executed in warps. If the threads had to wait for their neighbors for outstanding workloads, the acceleration could be completely

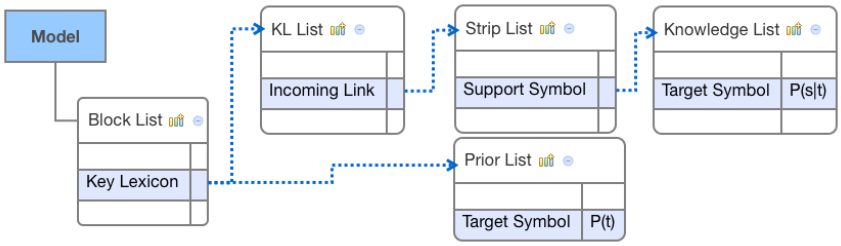


Fig. 3.2: In-memory knowledge base layout

diminished.

To address the above mentioned limitations, we further improve the GPU implementation from the perspectives of knowledge base storage and workload distribution. We also generalize the design to other parallel architectures such as the Xeon Phi co-processor.

3.5.2 In-memory Knowledge Base

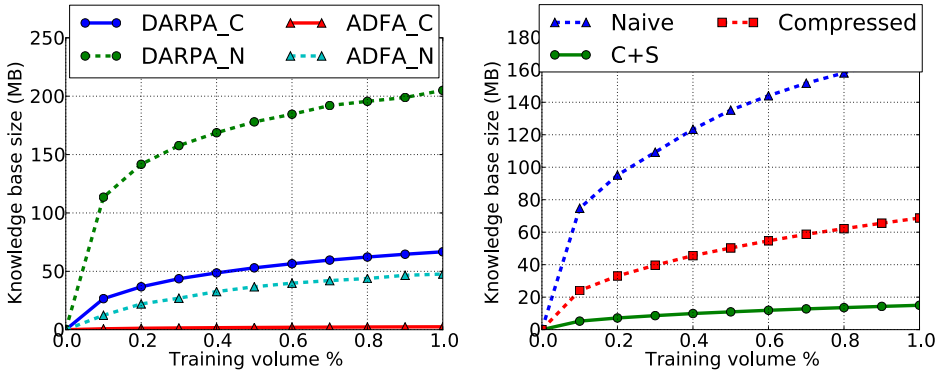
To fully utilize the computing resources on GPU, knowledge bases storage needs to be designed for both space efficiency and convenient querying.

The knowledge base of confabulation network is flattened and stored in the device memory. There can be multiple knowledge bases on the device, and Fig. 3.2 shows the structure of one knowledge base. It maintains a *Block List*. Each entry in the *Block List* is the record of a key lexicon. It contains a pointer to the list of all incoming connections (*KL List*) to this lexicon. The excitation levels of the candidate symbols in a key lexicon are stored in the shared memory, and the usage of shared memory determines how many blocks can be co-scheduled on a stream processor. Hence, a key lexicon may be divided into multiple blocks based on the size of its candidate symbols in order to optimize the GPU occupancy. A *KL entry* in the *KL List* has a pointer that points to the corresponding knowledge matrix. Because the matrix usually has very high sparsity, it is stored in a list of list format (LIL). Each entry of the LIL gives the conditional probability $p(s_j|t_i)$, where s_j and t_i are symbols in corresponding support lexicon and target lexicon respectively. Let each row of the knowledge matrix corresponds to a symbols s_j and each column corresponds to a

symbol t_i, the LIL is arranged in row-major order. Each entry in the *Strip List* represents a row in the matrix, and each entry in the knowledge list represents a non-zero entry in that row. Since different target candidates $t_i, t_i \in key\_lexicon$ need knowledge links from the same support symbol $s_j, s_j \in support\_lexicon$ to calculate their excitation level, this arrangement makes sure that the algorithm accesses *Knowledge List* sequentially for a better memory locality. Finally, the block entry also contains pointers to the prior probabilities (*Prior List*) of the key symbols for the calculation of equation (2.3).

The size of the trained knowledge bases for DARPA and ADFA datasets are plotted in Fig. 3.3a. The naive implementations (DARPA_N, ADFA_N) gives the potential size of knowledge base without compression. As we can see, the memory usages grow quickly as the size of the training data increases. The compressed implementation (DARPA_C, ADFA_C) compresses the sparse matrices using LIL and reduce the memory consumptions significantly. For DARPA and ADFA datasets, it gives 67% and 95% reduction in memory usage respectively. Such improvement is particularly notable for ADFA because features associated with the system calls are extremely sparse. For both implementations, we can see that the size of knowledge base gradually become stabilized.

Shared knowledge link is also implemented, where the knowledge links connecting different neighbor vehicles to the target vehicle are merged into the same probability matrices. This not only makes the nodes of interactive features more general and exposed to more training samples (e.g. $\langle neighbor(1).distance\rangle$ and $\langle neighbor(2).distance\rangle$ can share the same lexicon and knowledge links), but also reduces the memory usage. As Fig. 3.3b shows, using the compressed knowledge storage reduces 60% memory requirement, and further implementing the shared knowledge link (C+S) can give an additional 78% memory reduction.



(a) DARPA and ADFA usages (b) Vehicle usage of zone 190

Fig. 3.3: Memory usage of individual models

3.5.3 Workload Mapping and Anomaly Score Computation

Instead of mapping each key lexicon to individual threads, we map it to a CUDA block, which consists of up to 192 threads. Such implementation has two benefits: first, blocks can be dynamically scheduled, thus uneven workloads among different lexicons would no longer introduce control divergence; second, given the large number of threads in a CUDA block, different symbols in the key lexicon and different knowledge links associated to the symbol can be processed in parallel at thread level. Hence it reduces the runtime by factor of D and F as defined in Section 3.3).

Such workload partition and mapping is limited by the amount of hardware resource as will be discussed with the computation kernel. A large key lexicon with many symbols will have to be divided and mapped to multiple CUDA blocks, in order to fit all symbols in the shared memory. During the system initialization, the trained knowledge bases are flattened and loaded to the GPU. The input streams are then organized into corresponding format and dynamically sent to the devices at each frame. Each CUDA block corresponds to a key lexicon. For those large lexicons, multiple CUDA blocks are assigned to them.

The kernel for computing anomaly score follows the typical MapReduce style. Two stages are defined: excitation mapping and score reduction as shown in Fig. 3.4.

The mapping stage calculates the excitations of the symbols in a key lexicon, and stores

Algorithm 5 Optimized Recall Kernel

```
1: procedure optimized_recall(KB, IN):        # KB: knowledge base; IN: input symbols
2: shared memory: exbuf[U_max]
3: block ← KB.blocks[blockIdx.y];
4: N ← block.num_symbols
5: for t = threadIdx.x : blockDim.x : N:
6:     exbuf[t] ← KB.prior[t]
7: sync threads
8: —          # Knowledge mapping
9: for each kl ∈ block.KLs:
10:    if IN[kl.input_idx] is empty:
11:        continue
12:    strip ← kl.strips[IN[kl.input_idx]]
13:    for t = threadIdx.x : blockDim.x : strip.len:
14:        e ← strip.entries[t]
15:        atomicAdd(exbuf[e.key], e.value+B)
16: sync threads
17: —          # Excitation reduction
18: obs ← exbuf[IN[block.input_idx]]
19: b ← threadIdx.x
20: for t = b+blockDim.x : blockDim.x : N:
21:    exbuf[b] ← max(exbuf[b], exbuf[t])
22: sync threads
23: ref ← threadReduceMax(exbuf[0:blockDim.x])
24: Score[blockIdx.x] = (ref - obs) / ref
```

them in the shared memory buffer. Consider a key lexicon l with symbol set S_l and supporting lexicon set F_l. When the kernel receives a new input for support lexicon $k \in F_l$, it uses the input symbol $s \in S_k$ to locate the activated strip from the knowledge link LIL's. This strip contains the non-zero conditional probability $p(s|t)$, where t is a symbol in key lexicon. Traditional way to calculate the anomaly score of a key lexicon is to process its candidate symbols one by one. For each candidate symbol t, strips associated to all the support nodes are searched for the specific $p(s|t)$ and the values are accumulated. Such approach constantly loads different strips and hence has a low cache performance. We adopt a reversed approach that processes strip one by one. An active strip corresponding to symbol s in the support lexicon is read by multiple threads, which will then add the obtained

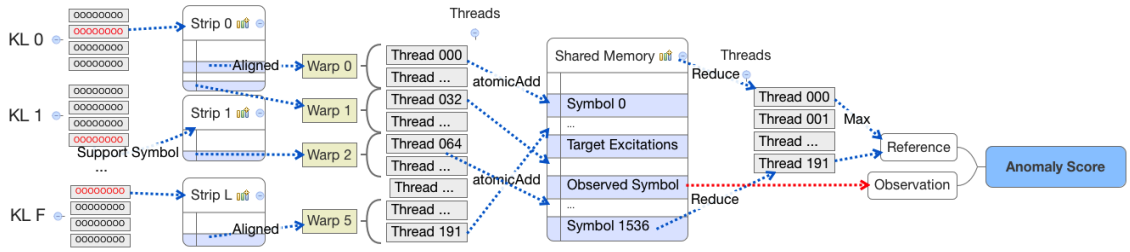


Fig. 3.4: Anomaly Score Computation

knowledge value $p(s|t)$ to the memory location that corresponds to the excitation level $y(t)$. Atomic add is used here since the same variable $y(t)$ is accesses by multiple threads. To prevent the control divergence, the strip-lengths are warp aligned so that the threads of a warp follow the same control flow. In this way, the cache performance is optimized because the strip access patterns are continuous. The excitations of the key symbols S_l are stored in the shared memory for efficient atomic addition and inter-thread operations. If a lexicon has more symbols than the pre-defined shared memory usage U_{max} (1536 in Fig. 3.4), it is partitioned into multiple blocks. The block dimension (192 in Fig. 3.4) is jointly chosen with U_{max} for higher device utilization (on Tesla C2075, the configuration offers 8 concurrent active blocks on one multi-processor and a theoretical occupancy of 100%). The synchronization required by the atomic addition does not cause much performance degradation. If the lexicon had many symbols, the possibility that multiple threads writing the same symbol would be low; if the lexicon had very few symbols, the computation of this lexicon itself is less time consuming.

In the reduction stage, all the excitations $\{y(t), t \in S_l\}$ buffered in the shared memory are compared and the most likely symbol t_{max} is selected. The excitation values of this reference symbol and the observed symbol generate the key lexicon anomaly score, which is then collected to calculate the network anomaly score based on Equation (2.3). The formal representation of the process is shown in Algorithm 5.

3.5.4 Implementation on Xeon Phi

In addition to NVIDIA GPU, we investigated the performance of AnRAD on another emerging multicore architecture, the 64-core Intel Xeon Phi processor. In this work, we use the co-processor in offload mode. The same memory layout as in Fig. 3.2 were adopted. The main algorithm is also similar as in Section 3.5.3. However, the workloads are mapped to different units of the co-processor architecture. Typically, Xeon Phi KNC chip has less physical cores than GPU does, but each of Xeon Phi's core are fully featured processor, and thus more powerful than the CUDA core in NVIDIA GPU. In particular, each Xeon Phi core is equipped with a 256-bit vector engine, which can be effective in the mapping-reduction process. Therefore, the lexicon-wised CUDA block computations are mapped to individual OpenMP threads, and within each OpenMP thread, vector operation is used to realize the parallelism originally enabled by the multiple CUDA threads.

3.6 Evaluations

3.6.1 Single Data Streams

We implemented AnRAD in the aforementioned parallel architecture, including CPU multi-threading, naive GPU acceleration, optimized GPU acceleration and Xeon Phi offloading. We compare the four designs using test data from vehicle monitoring, DARPA and ADFA datasets. For CPU multi-threading, we use Intel Xeon W5580 with 4 cores 16 logic processors running at 3.20GHz clock frequency. For GPU based implementations, we use NVIDIA Tesla C2075 with 448 CUDA cores running at 1.15GHz clock frequency and 6GB device memory. $U_{\text{max}} = 1536$ and blockDim.x$= 192$ is selected to achieve full theoretical occupancy. The Intel co-processor implementation is on a Xeon Phi 5100 with 60 cores running at 1.053GHz and 16GB memory capacity. A maximum of 240 threads are allocated.

Table 3.2: Single stream per-frame runtimes

Implementation		DARPA	ADFA-LD	Vehicle
CPU 1-thread	time	200.5ms	123.4ms	78.6ms
	speedup	1	1	1
CPU 16-thread	time	25.7ms	23.4ms	12.2ms
	speedup	7.8	5.3	6.4
GPU naive	time	146.9ms	44.5ms	150.1ms
	speedup	1.4	2.8	0.52
GPU optimized	time	0.210ms	0.0742ms	0.178ms
	speedup	**955**	**1663**	**442**
Xeon Phi KNC	time	4.04ms	1.02ms	3.27ms
	speedup	50	121	24

Table 3.2 compares the performance of those different implementations. As observed from the table, CPU with 16 threads achieves 5 to 8 times of speedup compared to serial implementation. It is not linear scaled with thread number mainly due to the memory stalls caused by concurrent memory access. The naive implementation on GPU, however, only provides marginal improvement or even runs slower than the single-thread baseline, because imbalanced workloads across threads produce control divergences. The optimized GPU implementation provides substantial speedup over the baseline methods. Finally, the improvement on Xeon Phi is also significant since the workflow follows the same principle of the GPU implementation. The reason that Xeon Phi implementation is not as fast as GPU is that the single testing stream does not generate enough workload for the maximal 240 threads. For example, the generated DARPA network has only 123 key lexicons, so can only utilizes about half of the thread capacity. Thus, GPU may offer better responsiveness on small and randomly arrived service requests, while the processing power of Xeon Phi would be sufficiently utilized by large workload batches.

3.6.2 Power and performance tradeoff

Furthermore, we evaluated the portability and performance of the AnRAD framework on high-performance workstation (NVIDIA Tesla K20c GPU) and embedded system (NVIDIA Jetson TK1). The former has 2496 CUDA cores running at 706 MHz clock frequency and

Table 3.3: Power and performance

Device		DARPA	ADFA-LD	Vehicle
K20 GPU	time	0.270ms	0.0622ms	0.231ms
	speedup	49.9	55.9	33.4
	active power	102.4W	98.87W	86.3W
	active energy	27.6mJ	6.1mJ	19.9mJ
Jetson TK1	time	13.48ms	3.477ms	7.709ms
	active power	2.5W	3.5W	2.6W
	active energy	33.7mJ	12.1mJ	20.0mJ

5 GB device memory; while the later has 192 CUDA cores running at 852 MHz clock frequency and 1.75 GB device memory. The power consumption of K20 workstation and Jetson board are 194.7W and 3.4W respectively when they are in idle state. We measure active power as the difference between average executing power and average idle power. Similar to the above analysis, we compare performance using data from vehicle monitoring, DARPA and ADFA.

Table 3.3 compares the performance of the implementations on both devices, and demonstrates that the K20 GPU achieves speedup of approximately 30 to 50 times compared to the Jetson GPU. However, the Jetson system on average consumes an active power of 2.87 W, while the K20 system's average active power is 95.86 W. Our results show that the execution on Jetson is effectively energy-neutral as compared to K20 despite the longer runtime per frame. However, it gives significant efficiency in active power consumption in all cases. Despite its low power, Jetson still exceeds the real-time requirements — for example, it achieves a 13.5ms per frame processing rate on DARPA stream whose input rate is only 300ms per frame.

3.6.3 Multi-stream Extension on Wide-area Monitoring

The previously discussed parallel implementation assumes that there is only one active knowledge base and one set of lexicons. In this section, we use wide-area monitoring of vehicle behaviors as a case study to demonstrate how the parallel implementation extend to

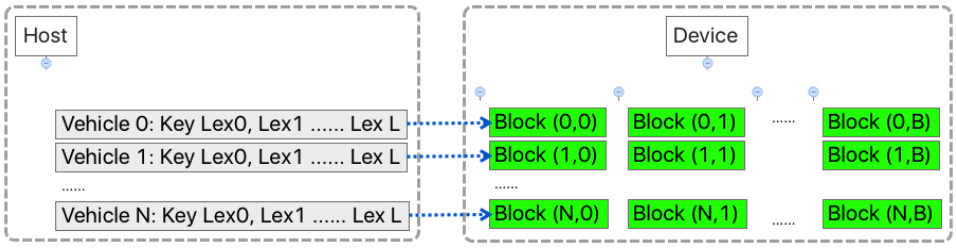


Fig. 3.5: 2D workload mapping

concurrent streams with multiple contexts.

After the universal network structure is built, separate knowledge bases are trained for each zone using their local traffic streams. A context selection module is used to associate the input vehicles with their corresponding zone knowledge base. Then the workloads are assigned to the computing platforms for anomaly scoring. The vehicles' key lexicons are mapped to OpenMP (CPU or Xeon Phi) threads or CUDA blocks (GPU). Using the GPU implementation as the example, during mapping, the abstract meaning of block-thread model is further exploited. Shown in Fig. 3.5, the concurrent streams (i.e. vehicles appeared in the same time frame) are mapped into a 2-dimensional CUDA grid, in which distinct vehicles are assigned across the first dimension of the grid (gridDim.x), and their key lexicons are mapped to the second dimension (gridDim.y). This design allows the pipeline to handle input streams with varying volumes. Finally, the results from the devices are collected to generate vehicle status reports.

To evaluate the performance of AnRAD in multi-stream multi-model scenarios, different training set volumes (small — 80min, medium — 160min, large — 240min) are evaluated in order to test the system performance under different levels of knowledge.

Fig. 3.6 shows the device memory consumptions for the knowledge bases. The X-axis is the number of the detection zones, and the Y-axis is the accumulated knowledge base size in megabytes. As the number of zones increases, the memory consumption increases nearly linearly, which indicates a balanced distribution of the knowledge base sizes. With different volume of training sets, the maximal total consumptions range from around 3GB

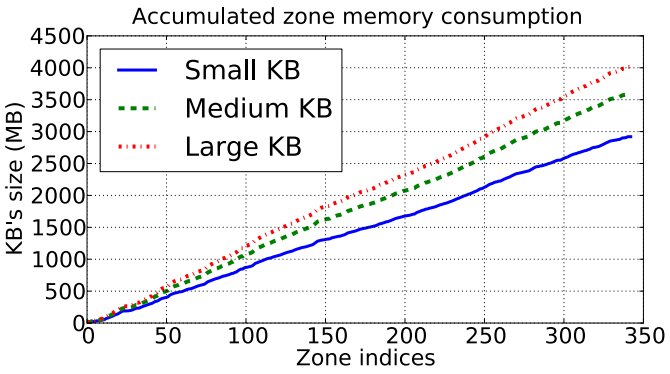
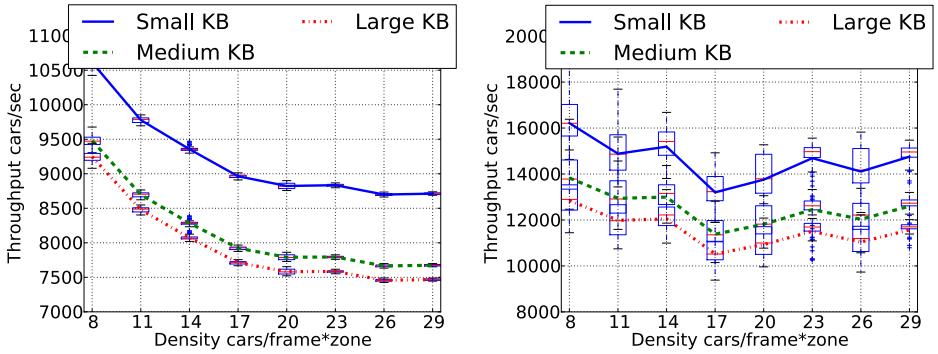


Fig. 3.6: Memory consumption of multiple zones

to 4GB, which is much lower than the available device memory of C2075 (6GB) or Xeon Phi (8GB). So a single GPU can support the entire 325 zones.

We define the throughput of the vehicle monitoring system as the number of cars that AnRAD can check (for abnormal behavior) per second. 100 zones are randomly picked and have their throughput collected with different vehicle densities. Fig. 3.7 illustrates the throughputs of the AnRAD recall on Tesla C2075 and Xeon Phi. The X-axis shows the density, i.e. the average number of vehicles that appear simultaneously in a zone. The Y-axis is the overall throughput of the 100 zones. On both devices, the throughputs drop as the vehicle density increases, because a denser neighboring among vehicles induces more processing of the interactive features. The throughputs get stable when the interaction-related key lexicons are saturated. GPU generally has smaller variations in different traffic patterns as in the box plots Fig. 3.7. Also, larger knowledge bases cause lower throughputs, but the degradation is smaller with knowledge bases approaching convergence. The throughputs of Xeon Phi 5100 on Fig. 3.7b are roughly twice as high as those in Fig. 3.7a of Tesla C2075. These generally fit the specifications that Xeon Phi 5100 peaks at 1011Gflop [44] and Tesla C2075 peaks at 515Gflop [72] of double precision performances.



(a) Tesla C2075 (b) Xeon Phi 5100

Fig. 3.7: Vehicle detection throughputs

3.7 Conclusion

We have presented an HPC-based neuromorphic anomaly detection framework that provides real-time processing for concurrent data streams. The method is significantly accelerated on GPUs and Xeon Phi processors with fine-grained and coarse-grained parallelization.

CHAPTER 4
LOW-POWER REALTIME DETECTION USING BURST CODE ON A NEUROSYNAPTIC PROCESSOR

4.1 Introduction

With the blooming of machine learning and neural networks, intelligent systems have been developed for various applications such as image recognition [87], multi-media retrieval [23] and intrusion detection [94]. Many of them process streaming data in real-time and imposes high demand in accuracy and computation throughput. Real-time anomaly detection is one of these applications that continuously monitors and processes incoming data streams for patterns that do not conform to normality. It is an extremely desirable feature for improving the autonomy of today's unmanned systems or mobile devices. However, to deliver the required performance under limited power constraint is a major design challenge.

The brain has unprecedented performance and energy efficiency in cognitive tasks [52]. It is believed that perception is a procedure of probabilistic inference, and the efficiency of

the biological neural system comes from its massive parallel architecture, spiking based communication and closely coupled computing and storage. Inspired by the biological structure, the IBM Neurosynaptic System with the TrueNorth architecture [65] provides a low-power platform for large-scale prototyping of Spiking Neural Network (SNN) based intelligent systems. Meanwhile, brain-inspired anomaly detection has been proposed [26] and shown to give superior detection quality than many traditional approaches. It performs inference based detection and features massive parallel structure that facilitates neuromorphic implementation. However, to implement the inference based anomaly detection on the TrueNorth processor is not straightforward.

A TrueNorth chip contains 4096 neurosynaptic cores, each of which has 256 axon inputs and 256 neurons. The synaptic connectivity is realized by a 256x256 crossbar. The connected core networks are encapsulated into Corelet [4] for abstraction and modular designs. While the processor provides potential to address the performance and power constraints for real-time embedded applications, challenges exist when mapping a signal processing flow onto this platform due to its hardware constraints. Firstly, each neuron (column) can only support 4 different input weights, while the weight of a connection is decided by the type of the axon (row). This limits all neurons who share an axon input to use the same weight rank at that row. However, most models' learned parameters are real numbers. Secondly, neurons communicate with each other using spikes, how to encode numerical value into spike trains is application specific. Thirdly, the crossbar's size of a core constrains the fan-in and fan-out of a neuron to 256. This hinders the direct mapping of big networks. Finally, some common arithmetic operations, such as division and maximum, are not as readily supported in TrueNorth as in traditional architectures. The neural circuit implementation of these operations also varies for different encoding schemes.

In this work, we focus on implementing a trained inference network on TrueNorth for real-time anomaly detection [26]. Instead of rate code, which has been widely used in many other TrueNorth applications [32], burst code is used because it gives higher computation

accuracy and allows very simple implementation of certain arithmetic operations, such as maximum. A Corelet library is developed, which consists of neural circuits for operations in the anomaly detection model. Our system first extract the topology and parameters of the network from the learned knowledge base to construct the network. Then, it flattens the network and map it to the Corelet library components for TrueNorth configuration. With a controllable clock driver input, the network is activated by streaming data in an event-driven way. Anomaly scores are calculated in real-time by probabilistic inferences of spatial-temporal features. To our best knowledge, this is the first work that applies neurosynaptic processor to the real-time anomaly detection.

The contributions of this chapter are the followings:

- We build a generic parser that transforms and maps inference-based anomaly detection network [24, 26] to a spiking neural network.
- A novel spike burst coding scheme is proposed for efficient representation, high accuracy and more convenient implementations. A Corelet library, *NeoInfer-TN*, is developed, which contains the neural circuit implementation of the network components for this coding.
- The adoption of bust code enables a two-phase pipelined processing for higher throughput. The throughput only depends on the spike encoding window configured to achieve a required data precision.
- A tunable accuracy factor is provided to enable tradeoff between detection accuracy and throughput.
- The network is evaluated with real-time intrusion detection data stream, and the accuracy, throughput and power performance are reported.

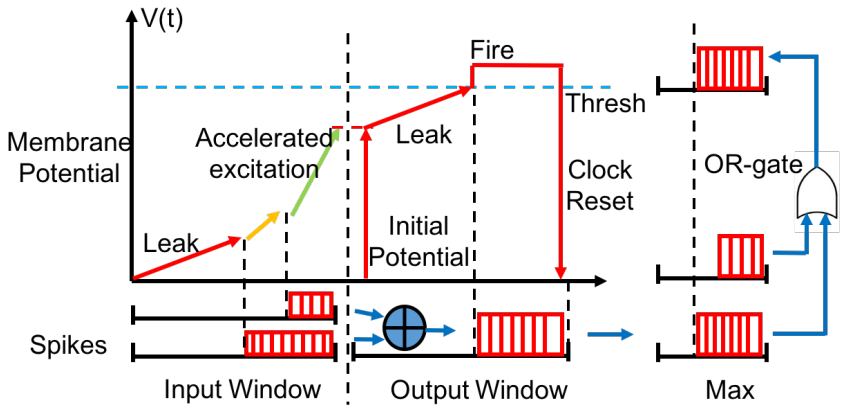


Fig. 4.1: Burst Code Neuron Dynamics

4.2 Spike Burst Coding

4.2.1 Encoding Mechanism

An important feature of the spiking neural network is that it encodes non-binary information into binary spike trains. A number of code scheme has been investigated [32], including binary code, rate code, population code and time-to-spike code. Among these schemes, rate code, which represents signal amplitude by the spiking frequency, is the most popular one and have been studied in many works [14, 73, 76].

The selection of coding scheme must lead to low-cost implementation of operations in the model and accurate representation of variables. In addition to integration and accumulation, which are common operations in many neural network models, our detection model has two special operations, maximum, which finds the maximum excitation level among all neurons in the same key lexicon; and division, which calculates the anomaly score as in Equation (2.2). None of the previously mentioned coding schemes give efficient implementations of these operations. The widely used rate code also suffers from sampling error, which reduces its accuracy. In this work, we propose a burst code that represents the numerical value of variables using the number of spikes that burst in a window.

Our burst code works in two phases alternatively as shown in Fig. 4.1: a burst neuron

integrates spikes from the axons during the *input phase* and emit spikes in the *output phase*. While the neuron has a constant positive leak, the presence of the input bursts accelerates the neuron's membrane potential raise, and result in a higher initial potential at the beginning of the output phase. The higher this initial level is, the faster the neuron reaches the firing threshold. It then stays on the threshold, fires until a negative input, i.e. the clock reset, is received. A binary code input can be considered a burst of only one spike, but the full input phase can be saved. The burst code is similar to a time-to-spike code in the sense that they both encode information into temporal representations. However, it is more fault tolerant than the time-to-spike code, as it relies on a set of spikes instead of the timing of a single spike. Essentially, the burst code works like the temporal version of a population code [32], but it occupies only one neuron.

The burst code has two advantages over the more widely used rate code. Firstly, the code is more compact. For example, to represent 100 distinct values, the burst code only needs $K = 100$ ticks code window. In the case of a stochastic rate code, the spikes fire as a Bernoulli process and we use $\dot{p} = \#\text{spikes}/K$ to represent the information. The approximately normal 95% confidence interval $\dot{p} \pm 1.96\sqrt{p(1-p)/K}$ needs around $K = 10000$ to represent 0.01 granunarity. Please refer to Section 4.5.4 for the comparison of precisions of the two codes. Secondly, the burst code can max-pool values much more efficiently: a simple OR-gate will find the maximum as shown in Fig. 4.1, while the rate code needs a more complicated winner-takes-all (WTA) [73] circuit and scaling. We find the burst code a better option for the detection network.

4.2.2 Detection Error Analysis

Using similar assumption as in Section 2.3.2, we analyze the anomaly detection by simplifying it into a binary classification problem, and the abnormal observation t can be detected

if and only if inequity (4.1) holds.

$$\varphi_l(s) = \sum_{k=1}^{N} \ln \frac{\dot{p}(s_k|t')}{\dot{p}(s_k|t)} > 0 \tag{4.1}$$

Here, t' is the normal symbol and should be predicted as $t_{\max}$. $\dot{p}(.)$ is the knowledge link value represented by the spike codes. N is the number of knowledge links. If $\varphi_l(s) < 0$, a false negative error is caused, and the formulation for false positive error is similar. In the following, we use w to denote the size of the code window.

In the case of burst code, the value representation is deterministic and equivalent to an equal-width quantization on the original learnt parameter $p(s|t)$. Therefore the error caused by spike encoding is approximately of uniform distribution with value range $p(s|t) \pm 1/w$. So for some constant error ϵ, each knowledge link has

$$\Pr[|\dot{p}_{\text{burst}}(s_k|t) - p(s_k|t)| \geq \epsilon] \leq 1 - (2\epsilon \frac{w}{2}) \tag{4.2}$$

Since there are $2N$ such parameters, to make the union bound probability $P_{\text{burst}} = 2N(1 - \epsilon w) \leq \rho$ for some constant $\rho > 0$, the window size must suffice $w \geq 1/\epsilon - \rho/2\epsilon N$, which is a convergent function with respect to N.

On the other hand, for the rate code, the encoding process emits spikes randomly using the excitation level as a firing probability, which is a Bernoulli process. For constant error $\epsilon \ll \mu$, let $\delta = w\epsilon/\mu$ where $\mu = E(\#spikes \text{ in window } w)$. By Chernoff Bound, the link error bound subject to

$$\begin{aligned} &\Pr[|\dot{p}_{\text{rate}}(s_k|t)| \geq \epsilon] \\ &= \Pr[w|\dot{p}_{\text{rate}}(s_k|t)| \geq \delta\mu] \\ &\leq 2e^{-\mu\delta^2/3} = 2e^{-w^2\epsilon^2/3\mu} \end{aligned} \tag{4.3}$$

Let $\mu = w/2$, again with $2N$ parameters, for the error probability $P_{\text{rate}} = 2Ne^{-2w\epsilon^2/3} \leq \rho$

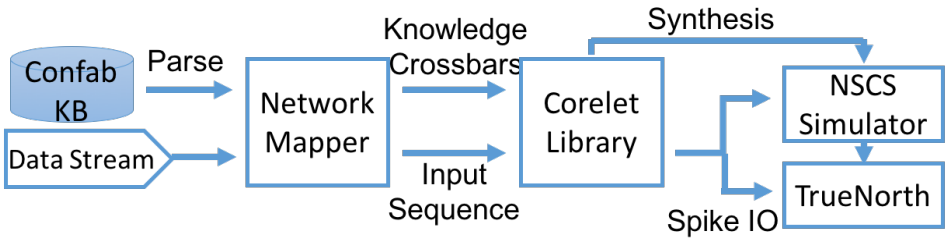


Fig. 4.2: System Workflow

for constant $\rho > 0$, the window size must be $w = O(\ln N)$, which scale with larger N.

Based on the analysis in Section 2.3.2, to make the false detection rate bounded by constant with respect to parameter number N, the excitation level's error rate P_{burst} or P_{rate} must be constants. Since the burst code error is convergent while the rate code has lnN windows size requirement, the choice of burst code could results in a lower false detection given similar window sizes.

4.3 System Design

Based on the self-structured inference network, we seek to implement it on the neurosynaptic system. This section introduces how the unstructured network is mapped to the crossbars for spiking network realization.

The aforementioned detection flow has four layers, (a) the support lexicon layer that collects input from the environment, (b) key lexicon layer that calculates neuron excitations using Equation (2.1), (c) anomaly score generator, which performs Equation (2.2) for each key lexicon, (d) and anomaly score accumulator, which merges all key lexicon anomaly scores using Equation (2.3). To convert the detection network into the Corelet configuration of TrueNorth, the overall workflow is shown in Fig. 4.2.

The network mapper reads in a trained confabulation knowledge base (KB) and maps the connections between support and key lexicons to a set of crossbar matrices considering the hardware constraints. It also maps the neuron observations to the input of the processor

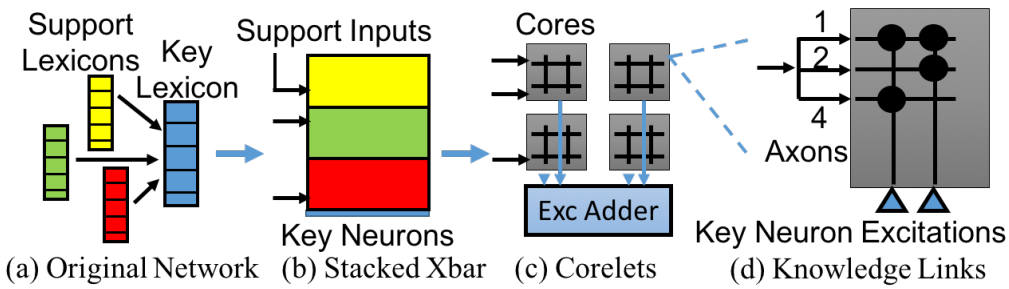


Fig. 4.3: Network Mapping

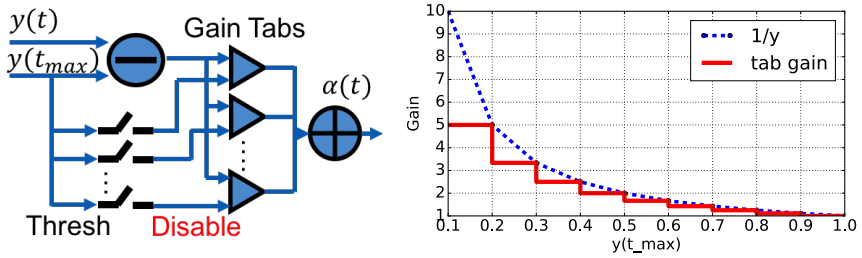
and synthesizes the input spikes from the given data stream. In the second step, the synthesizer maps each crossbar matrix into Corelets using our NeoInfer-TN library in the Corelet Programming Environment (CPE). It also maps the score generation and score accumulation layers into library components. At last, the network and its inputs are tested on the NSCS simulator [78] and the TrueNorth chip.

4.3.1 Network Mapping

The connection between neurons in the support and key lexicons forms a bipartite graph and can naturally be implemented as crossbar arrays, where each row corresponds to a neuron in the support lexicon and each column to a neuron in the key lexicon. However, a neurosynaptic core can only implement crossbar up to 256x256, while the number of neurons in the support and key lexicons reaches up to 3000 depending on the feature size. Matrix partition must be considered to implement one connection using multiple cores.

How to accurately represent synaptic weight in the crossbar is another challenge. Due to the hardware limitation, each column can only support 4 different weights, and all weights in the same row must have the same rank in their corresponding columns. We resolve this problem by decompose each row into three rows with different weights, and the binary combination of the three rows provides more flexibility in weight representation.

Fig. 4.3 shows each step performed by the network mapper. From the original network, large connection matrices are generated for key lexicons as shown in Fig. 4.3b. Each row



(a) Scorer Structure (b) 10-tabs Divider Response

Fig. 4.4: Key Lexicon Anomaly Scorer

represents a support neuron and each column represents a key neuron. As shown in Fig. 4.3d, for each support neuron input, 3 axon lines with corresponding strengths of 1, 2 and 4 are used. Using different combination of them, we can represent synaptic weight in the range of [0, 7]. We choose to use 3 lines because TrueNorth only supports 4 ranks in each column, and one of the ranks is used to implement an input clock, whose functionality will be explained in Section 4.2. Then this large matrix is partitioned into multiple 256x256 crossbars that fits in single cores as shown in Fig. 4.3c. Additional Corelets are also inserted to merge the results.

To reduce the core usage by the network, we follow a "train-then-constrain" [32] approach. During the self-structuring stage [26], we limit the support connections of each key lexicon to be less than 15, and then train the knowledge base accordingly. This significantly reduces the number of required synapses with only marginal impact ($< 1\%$) on the detection quality. More detailed results will be given in Section 4.5.4.

4.3.2 Divider and Key Lexicon Burst Scorer

With the new encoding scheme, the hardware implementation of some arithmetic operations should be redesigned, and one example is division, which is used for the computation of Equation (2.2). We design a segmentation-based scorer for the lexicon anomaly score as in Fig. 4.4a. The basic idea is to scale the excitation difference using multiple fixed-gain neurons.

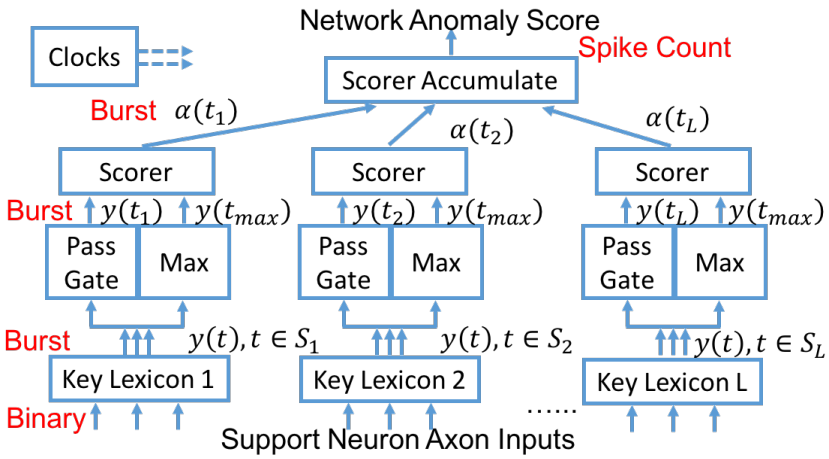


Fig. 4.5: Corelet Architecture

While differentiating the normalized excitations $y(t)$ and $y(t_{\max})$, the later is also directed to an array of trigger neurons with different spike count thresholds. The triggers can fire to disable their corresponding gain tabs, which are linear neurons with fixed weight to threshold ratios. The gains are precomputed to approach $1/y(t_{\max})$ using Equation (4.4).

$$1/y(t_{\max}) \approx \sum_{i=1}^{M} \text{gain}_i I[\text{thresh}_i \geq y(t_{\max})] \tag{4.4}$$

Here, M is the number of the gain tabs, and $I(.)$ is an indicator function that takes value $\{0, 1\}$. Those tabs having thresholds larger than $y(t_{\max})$ are kept on. Increasing $y(t_{\max})$ disables more tabs, results in a smaller amplifier to the signal. M decides the precision of the division. In this work, a 10-tabs scorer is used to emulate divisor with 0.1 precision as shown in Fig. 4.4b. The red lines indicate the scaling response of the neurons. Finally, the differential signal $y(t_{\max}) - y(t)$ passes the gain tabs and generate the anomaly score.

4.3.3 Corelet Library and Architecture

A set of Corelet configurations are designed to realize the aforementioned integration, maximization, and division operations in burst code and they form the NeoInfer-TN library. The components are instantiated to construct the detection flow as in Fig. 4.5. From the bot-

tom layer up, the binary activations of the support neurons are sent to the "Key Lexicon" Corelet, who contains multiple Knowledge Crossbars and Excitation Adders to flexibly map arbitrary key lexicons. The excitations of all key neurons are computed and passed to the "Max" Corelet to find $y(t_{\max})$ using OR-gates. Also, the "PassGate" Corelet uses the actual observation t to clamp $y(t)$. The "Scorer" receives both signals and compute Equation (2.2) as described in Section 4.3.2. Finally, an "Accumulator" collects the scores and represent the network-wide score of Equation 2.3 using spike count. The network also creates "Clocks" Corelet to control the code windows and the neuron timing.

4.4 Inference Pipeline

Using the architecture constructed in Section 4.3, we carefully design the pipeline to hide the delay and improve the throughput. This section introduces the timing of the neurons.

4.4.1 Timing for Real-time Processing

To hide the latency introduced by the burst code operations, we mesh the input window and output window as in Fig. 4.6. Integrations of lexicon excitations takes one window since they are parallel additions. Lexicon scores requires two windows for divisions. In this way, the processing delay is 5 windows, and the detector handles one sample every 2 code windows (input phase and output phase). The throughput of the system does not depend on the scale of the network, but only the length of the code window.

An input clock fires at the end of the code windows to reset the neurons. We can use this signal to dynamically adjust the throughput of the pipeline. When the window size is smaller than the range of the values, the code set an lower-bound to the excitations, which is not desirable. We would like to further the code window without affecting the spike representation range. Section 4.4.2 uses the accuracy factor to address this.

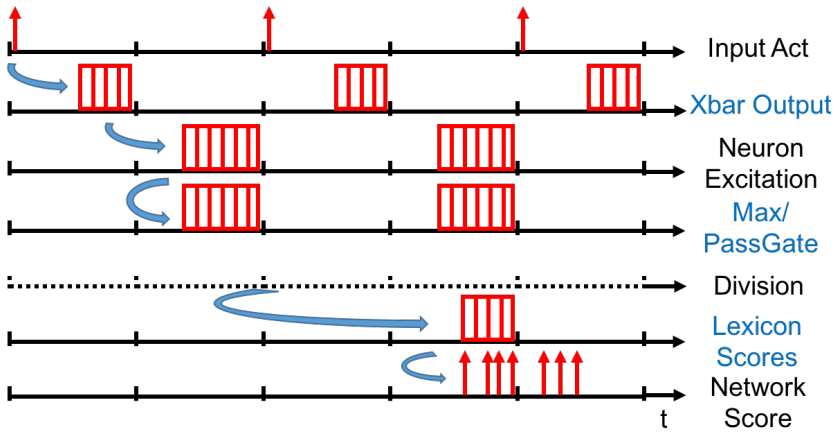
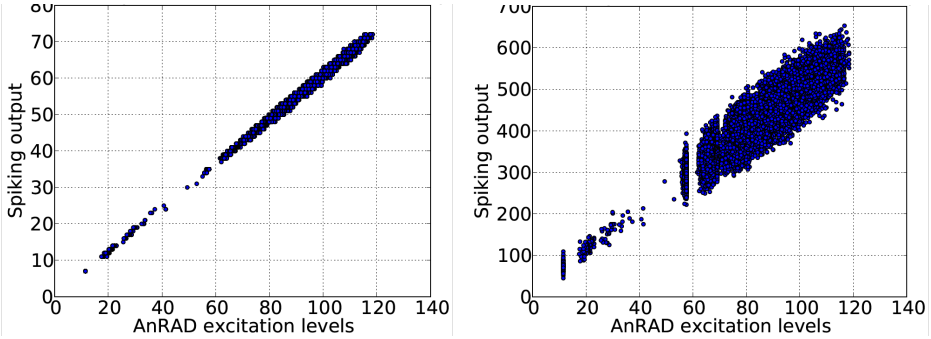


Fig. 4.6: Detection Pipeline Timing

4.4.2 Accuracy Factor

When the value range is fixed, reduced window length is associated with a lower precision of the computations. This may not be feasible for classification tasks as the reduced accuracy affects the ranking of the predictions significantly. However, anomaly detection is less sensitive to a reduced precision because the relative excitations, rather than rankings, are used to score the unlikely events. Therefore, we introduce an *accuracy factor* to the neurons to make tradeoff between the throughput and the accuracy.

The accuracy factor (AF) is actually a multiplier to the neuron's positive leak, which shorten the duration that the membrane potential needs to reach the threshold. AF is applied throughout the network to make the burst code to represent a wider range, so that a smaller code window (higher throughput) can be used. The larger AF is, the less precise is the computation.



(a) Burst Code (b) Rate Code

Fig. 4.7: Excitation Correlation between Spike Code and Reference Program

4.5 Evaluation

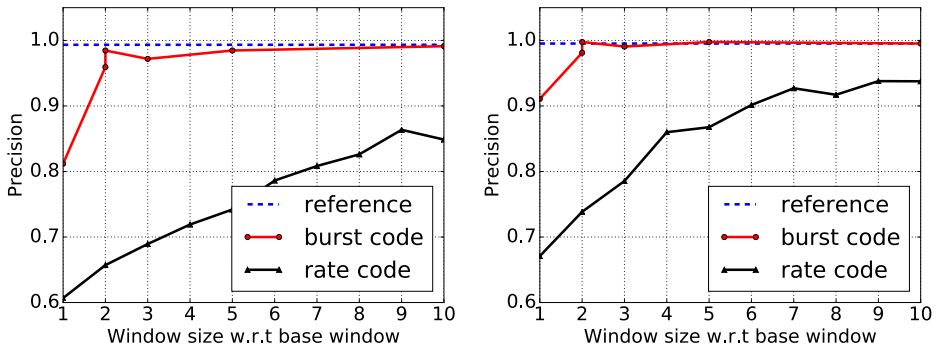
4.5.1 Experiment Setups

For evaluation, we structure and train the confabulation network using the DARPA intrusion detection dataset [59]. For each IP address pairs, traffic statistics are recorded per 300ms-frame. A random sample of 20000 frames are used for training. The test data contains about 80000 frames of 7000 normal samples, and 60000 frames of 38 attacks. The reference network and baseline results of the confabulation network are generated by the AnRAD framework [26].

The hardware development platform is IBM NS1e, which has a TrueNorth processor and its peripheral devices installed. The processor runs on 1ms/tick, therefore to represent data in the range [0, 99] we need a burst code window 100ms wide. There are 4096 neurosynaptic cores, and 1 million hardware neurons on the chip, which consumes 50 to 100mW during typical operation. The whole board consumes 2 to 3.5W power depending on the utilization.

4.5.2 Burst Code vs. Rate Code

We first compare burst code and rate code for their computation accuracy. The window for burst code is 100 ticks, while the window for rate code is 1000 ticks. One key lexicon



(a) Anomaly "nmap", Lexicon "service" (b) Anomaly "warezmaster", Lexicon "push"

Fig. 4.8: Precision of lexicons on anomalies

is selected from the DARPA network and has all its neuron excitations collected over 500 frames of data. Both are used to calculate the excitation level of the key lexicon neurons and compared to the baseline value calculated by the reference program. Fig. 4.7 is the scatter plots for both codes. The X-axis gives the baseline excitation computed by the reference program, and Y-axis denotes the spike counts at the neuron excitation pins from either code windows. Compared to rate code, our burst code shows better linear correlation with the reference computation, with a correlation coefficient of 0.998, while the rate code has 0.924. The burst code is able to achieve higher precision with a code window that is only 1/10 of the size of the rate code window.

Then we collect precision in detecting different classes of anomalies using both burst code and rate code. For abnormal samples, the "nmap" and "warezmaster" attacks are selected, and the correlated lexicons for service port and push flag are monitored. The base window size for burst code is 10-ticks, and the actual window sizes on the X-axis of Fig. 4.5.2 are picked as $10 * x$. The rate code uses a larger 100-tick base window to make the curves more comparable. Anomaly score threshold of 0.1 is selected for these tests. Observed from Fig. 4.8a and Fig. 4.8b, in detecting both types of anomaly, the burst code pipeline significantly outperform the rate code in terms of precision, while it only uses 1/10 smaller encoding windows. Therefore, the results generally comply with the analysis in Section 4.2.2.

Table 4.1: Network complexity impacts of constraint

Networks	Synapses	Cores for Key Lex	Total Cores
Original	3373K	5232	6116
Constraint	1322K	2169	2918
Reduction	60.8%	58.5%	52.3%

4.5.3 Network Construction

Using the self-structuring algorithm [26], we build the confabulation network, which is referred to as the "origina" network. We also constrain the support lexicon connections to be less than 15 (the original network ranges from 3 to 30), and refer to the new network as the "constrained" network. Both networks have 123 key lexicons. These lexicons represent features or feature combinations, such as server, packets, etc., extracted from network traffic. With constraint, each key neuron has much less incoming synapses, and thus consume less hardware resources. From Table 4.1, it is seen that the total number of synaptic weights is reduced by 60% by imposing the constraint on support connections. This reduces the neurosynaptic core usage by 50%, in which 74% of the usage is for the key lexicons and their support synapses. With such configuration, a single TrueNorth processor can handle the full confabulation network.

4.5.4 Detection Quality

Next we test the whole network for its performance in anomaly detection. Two configurations are tested: high accuracy with 100-tick burst window (TN-100) and high efficiency with 10-tick window (TN-10). The result is compared to three software implemented reference detection methods, including self-organizing map (SOM), replicator neural network (RNN) [9] and the reference detector using the full confabulation network. The metric is AUROC (Area Under Receiver Operation Curve) score, which measures the tradeoff between false alarm and true detection, the higher the better. Shown in Table 4.2, TN-100 outperforms SOM and RNN by around 5%. It is even slightly better than the reference

Table 4.2: Detection Qualities of Comparison Models

Methods	**SOM**	**RNN**	**Reference**	**TN-100**	**TN-10**
AUROC	0.879	0.898	0.933	**0.943**	0.914

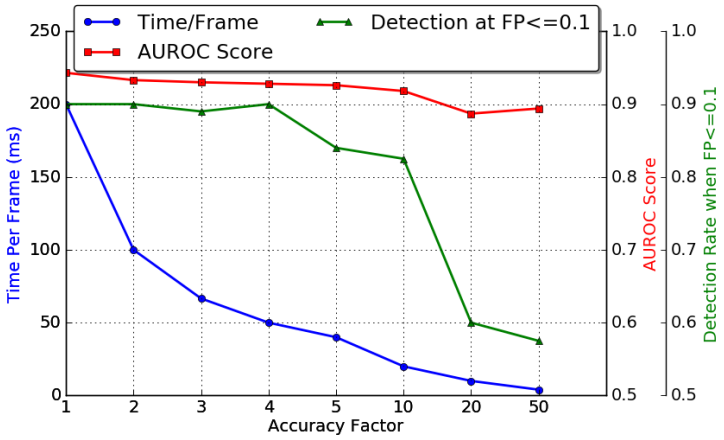


Fig. 4.9: Tradeoff between Quality and Speed

program. TN-10 provides 10X speedup over TN-100, and still generates better results compared to SOM and RNN.

4.5.5 Throughput and Accuracy Tradeoff

Computation speed is critical for real-time anomaly detection. With 100-tick windows, the spiking neural network uses 200ms for each input (alternative input/output windows). It can already achieves real-time detection as the network data stream was collected with sample intervals of 300ms. We would like to further investigate the potential performance increase for larger scaled data by trading off the accuracy.

Fig. 4.9 shows how the performance and detection quality vary with the change of the AF. The X-axis shows the accuracy factor. The Y-axises show the processing time (blue), AUROC score (red) and the best detection rates (green) when the false positive is less than 10%. As we increase the accuracy factor, the AUROC score only drops slightly from 0.943 (AF=1, 100-tick window) to 0.918 (AF=10, 10-tick window). However, the throughput of the system is improved significantly: with AF=10, the processing speed is

Table 4.3: Power and Performance of Different Platforms

Devices	Time	Power	Energy/Sample
Xeon W5580	25.7ms	68.0W	1747.6mJ
Tesla K20	0.270ms	102.4W	27.6mJ
Jetson TK1	13.48ms	2.5W	33.7mJ
TN-10 1.0V	20ms	104.1mW	2.1mJ
TN-10 0.8V	20ms	49.22mW	0.98mJ

reduced to 20ms/frame from the 200ms/frame base case. When the code window further shrinks to shorter than 10 ticks, the network still achieves good AUROC scores (> 0.88), but it generate so few spikes to the anomaly score that only a limited selection of thresholds can be used to tradeoff the detection and the false positives. Therefore, when we impose the false positive to be less than 0.1, the detection rate drops quickly for AF$>$10.

4.5.6 Power and Performance

Finally, we compare the power consumption of the confabulation network running on different platforms. For the baselines, we have a 16-threaded program on Intel W5580 quad-core CPU, whose active power is estimated using PowerAPI [12]. Also, a CUDA program is tested for active powers on an NVIDIA Tesla K20c workstation and the embeded GPGPU system Jetson TK1 using NVIDIA-smi and external Power meter. All baselines are supported by the AnRAD framework. The TrueNorth power is tested by the method in Cassidy et. al. [19]. To find the actual power consumption for the chip utilization, first the leakage power P_{leak} is measured when the system is idle, and then the total power P_{total} is measured with the network running. The active power is computed as $P_{\text{active}} = P_{\text{total}} - P_{\text{leak}}$. The leakage power is scaled by the number of cores actually used $P_{\text{leak\_scaled}} = P_{\text{leak}} N_{\text{cores}}/4096$. The final power is calculated as $P = P_{\text{active}} + P_{\text{leak\_scaled}}$. TrueNorth is capable of operating on 1.0V or 0.8V with the same 1ms ticks.

Table 4.3 shows the results of the baselines and the TrueNorth networks with 10-tick code windows. The spiking network is not only running faster than the CPU program, but is also 800/1700 times more energy efficient: it only consumes 2.1mJ at 1.0V and

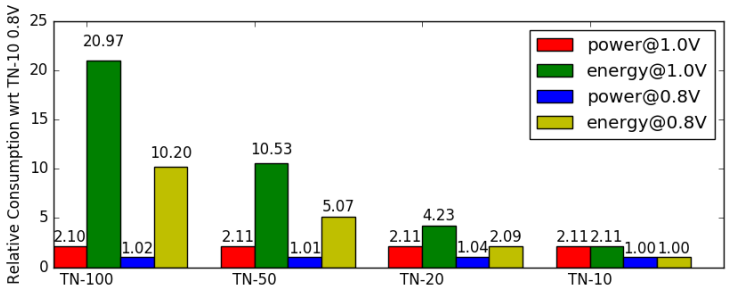


Fig. 4.10: Power/Energy Consumptions for Different Window Lengths

0.98mJ at 0.8V to process each sample. Although K20 is capable of achieving a higher throughput, it runs on a much higher power and consumes up to 30X more energy compared with the TrueNorth chip. The Jetson board also consumes low power, but overall the energy consumption is still much higher than that of the spiking networks. The DARPA network has around 50000 key neurons, each of which integrates 11 support activations per sample on average. Take TN-10 0.8V for example, the power efficiency is estimated as (#neurons$\times$#supports)/(time_per_sample$\times$power) $\approx 6 \times 10^8$ operations per watt-second. Note that the cores are not fully occupied, and smaller code window offers higher efficiency.

Finally, we test the power consumptions with different accuracies. In Fig. 4.10, we vary the code window from 10 to 100 and report the power and energy of TrueNorth implementation normalized with respect to the power and energy of TN-10 at 0.8V voltage. Obviously, high supply voltage results in about 2X higher consumptions. All accuracy settings are basically power-neutral because the spike frequencies do not change much given the same amount of time. However, since the processing time for each frame reduces with smaller windows, the energy usage is also less at high-efficient setting such as TN-10.

4.6 Conclusion

This chapter presents a streaming anomaly detection network using TrueNorth neurosynaptic processor. A trained confabulation network is mapped to Corelet with our NeoInfer-TN

library. The network uses an efficient burst code and features a highly concurrent architecture. The implementation achieves state-of-the-art detection precision, real-time processing and high power efficiency.

CHAPTER 5
BI-DIRECTIONAL ASSOCIATION BETWEEN DEEP IMAGE REPRESENTATIONS AND LOOSELY COUPLED TEXTS

5.1 Introduction

Learning to associate images with loosely correlated texts is an important feature for many retrieval applications. From textual input, we can search for images with natural language, or intelligently assign illustrations to news articles. Given an image, it is possible to generate caption automatically, or to locate relevant documents from a text database. In this chapter, we seek to improve the cross-modal matching performance given that the text-image parallel datasets are not specially constructed for query purposes.

There has been extensive researches on bidirectional mapping between images and words/sentences [34, 48–51, 61]. Learning an embedding space of different modalities is proved to be effective for high-quality datasets with descriptive sentences. But the task

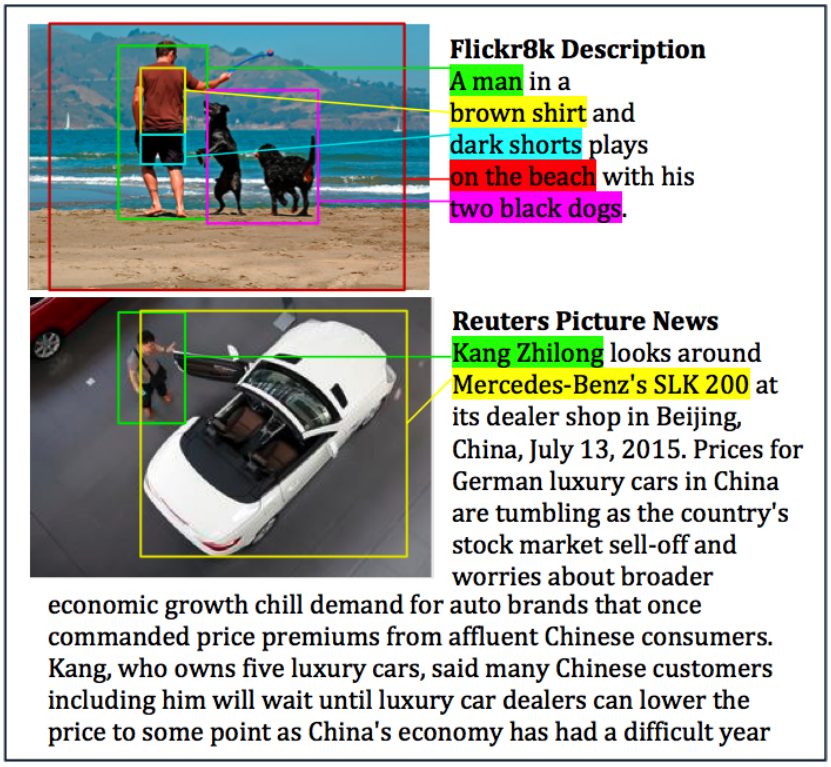


Fig. 5.1: Comparison between descriptive text-image pair and picture news

becomes much more challenging when tightly coupled text-image pairs are not available. The parallel text may contain many contents that do not directly describe the image, or conversely, the image could show objects that are otherwise not discriminative without proper contexts. Compare the example text-image pairs in Fig. 5.1, while most of the words in the Flickr8k [40] sentence are densely corresponded to the objects in the image, only a small portion of the Reuters Picture News [82] paragraph is explicitly describing the contents of its illustration. The rest of the paragraph is co-occurring just for the news background and may even cause overfitting to learning-based methods. Topic modeling has been studied to summarize text corpus [11, 17, 33], but they are mostly tuned for automatic annotation applications and are difficult to associate with continuous image features as embedding-based methods do. Even if the descriptive part of the news is somehow extracted, it is still hard to accurately map the words to the image components without recognizing the person's identity or the car model. Therefore, in order to successfully match the noisy text-

image pairs, we need to extract the useful portion of the paragraphs and to enrich the image representations.

The approach in this chapter follows the method of learning an embedding layer between texts and images. An image is partitioned into multiple regions of objects and has the region features extracted using convolutional neural networks (CNN) [53, 56, 75]. For a text paragraph, its dependent word pairs are used as the semantic fragments. Both image regions and word pairs are treated as bags of fragments and matched in the embedding space [48, 49]. To aid the mapping between images and noisy paragraphs, we propose two improvements to the fragment space. First, instead of learning a single level of embedding, we cascade embedding optimizers. The result from the upstream embedding is analyzed to determine the relevancy and discriminative power of the text fragments. Then the information is forwarded to the downstream embedding to suppress the noisy text portion and improve the secondary learning process. Second, we integrate multiple CNN's that are tuned for different contexts to construct the image fragments. The new fragments help the embedding not just match for the object-level image features, but also adopt diversified information such as facial characteristics of persons.

The contributions of this work are as the followings. In section 5.3, we analyze the problem of matching noisy paragraph and images, select the proper optimization objectives, and also propose an equivalent implementation of the alignment scores for accelerating the computation. In section 5.4 we design an cascade configuration of text-image matchers to refine the discrimintive set of text. In section 5.5, we integrate multiple CNN's to enrich the image representations and use facial recognition network as the demonstration. Finally in section 5.6, we evaluate the proposed methods with both synthetic dataset and real datasets of picture news collected from Reuters Picture News [82].

5.2 Related Work

It is an emerging topic of learning to bridge the gap between image and natural languages. Some works [55, 63, 92] have focused on generating novel captions from query images. A typical pipeline in Vinyals et al. [91] was that the image was first passed to the CNN [87] and had its compact representation extracted. Then the image representation was treated as the initial word input to the semantic space and used to generate a sentence label using a long-short term memory (LSTM) [39] predictor. Other works [49–51, 61] have focused on learning an embedding space for bidirectional mapping. Frome et al. [34] converted the whole images and the word labels into a common embedding space and defined a hinge rank loss to align the correct pairs. Instead of using a common embedding space, Karpathy et al. [48] broke the images into multiple objects using regional CNN [35] and the sentences into dependent word pairs using Stanford CoreNLP toolkit [62], and then learned to compute the similarity scores based on the visual-semantic fragment embedding. Most of the existing works have been focused on query-like text-image datasets such as Flickr8k [40] and Pascal1k [81], and achieved state-of-the-art accuracy. Only a small body of studies considered loosely correlated pairs, such as picture news.

To obtain neural descriptors of images, many studies have been conducted for different applications. For instance, the networks in Krizhevsky et al. [53] and Szegedy et al. [87] were dedicated to object classification for ImageNet challenge [30]. The VGG Face Descriptor [75] was tuned for celebrity identifications. Zhou et al. [95] specialized for scene classification. For the text representation, works have been conducted to convert words or sentences into vector space [8, 42, 66].

Topic modeling such as Latent Dirichlet Allocation (LDA) [11] has been an effective way to extract the essential part of large text bodies. There has been studies based on LDA for word sense disambiguation [57] and semantic category classification [22]. As for news media, Cano et al. [17] explored different methods in finding keywords from Twitter messages. Feng et al. [33] connected the image and text modalities by clustering the SIFT

features [60] of image regions into discrete words, and building a mixed LDA model with both visual and semantic words. They performed image annotation on BBC news dataset. The discretization of images may impose information loss compared to the embedding-based methods, but the key idea of extracting essential texts could be beneficial.

5.3 Visual-Semantic Embedding

The bidirectional retrieval task is essentially a ranking problem. For each text-image pair, an alignment score is calculated to indicate how closely correlated a text sample and an image sample are. The scores of all pairs in the searching space are ranked among the image peers or the text peers. The top-ranked images are the search result of a text query, or vice versa. For the datasets which are targeted by this chapter, the text queries are not short descriptive sentences that frequently refer to the image contents. They could be long paragraphs with only parts of them strongly connected to the images.

5.3.1 Text and Image Representations

Following the deep embedding approach [48,49], both the texts and images are broken into fine fragments. For images, RCNN [35] and Caffe [46] are used to detect the object regions. Each region forms an *image fragments*. The network is pre-trained with ImageNet [30] data and fine-tuned towards 200 object classes. Every image is represented by a bag of regions containing the whole image and up to 19 RCNN detections. The detection regions are selected by highest classification probabilities. The embedding of the i^{th} image fragment v_i is calculated as in equation (5.1).

$$v_i = W_v[\text{CNN}(R_i)] + b_v \tag{5.1}$$

where R_i is the pixels in region i and CNN(.) outputs the 4096-dimensional features of the fully-connected hidden layer (*fc7*) immediately before the RCNN classifier. W_v and b_v are learnt parameters. When the size of the embedding space is d, W_v is a $d \times 4096$ matrix.

The text paragraphs are analyzed using Stanford CoreNLP [62] and have their word dependencies extracted. Each pair of dependent words is grouped as a *text fragment* and the paragraph is represented by a bag of such fragments. The embedding of the t^{th} fragment s_t is computed by equation (5.2).

$$s_t = f(W_s \begin{bmatrix} {w_t}^p \\ {w_t}^c \end{bmatrix} + b_s) \tag{5.2}$$

where ${w_t}^p$ and ${w_t}^c$ are the 200-dimensional vectors of the parent and child words of the dependent pair. The vector representations are learned by unsupervised objective [42]. W_s is a $d \times 400$ matrix that transforms the lumped word pairs to the embedding space. The activation function f is the Rectifying Linear Unit ($ReLU$).

5.3.2 Selection of Objectives

The correlation between text fragment t and image fragment i is computed as the dot product of their embedding vectors, $v_i {s_t}^T$. One way of defining the alignment score [49] between the j^{th} image and k^{th} text sample is in equation (5.3), and the global alignment objective in equation (5.4) drives the optimization.

$$A_{j,k} = \sum_{t \in T_k} \max_{i \in I_j} v_i {s_t}^T \tag{5.3}$$

$$\begin{aligned} loss_G = \sum_j [\sum_k & \max(0, A_{j,k} - A_{j,j} + \Delta) \\ & + \sum_k \max(0, A_{k,j} - A_{j,j} + \Delta)] \end{aligned} \tag{5.4}$$

Here, T_k is the set of dependent word pairs of the k^{th} text sample and I_j denotes the regions of the j^{th} image. Δ is a constant margin that valued 40 in our experiments. The loss function essentially maximizes the correct alignment against the other images and texts. Compared with their former objective [48], the formulation simplifies the model and improves the ranking performance. However, such formulation assumes that each text fragment can only align to one image region with the highest dot product as in equation (5.3). This assumption works well for descriptive sentences because they are always directly referring to image regions. However, noisy paragraphs do not hold the same property. From our observation, it is possible for a word in the news article to align with multiple image regions. Therefore, we choose to use the original formulation [48] that combines the local objective and the global objective. The loss is defined by equation (5.7) with the alignment calculation (5.5).

$$A_{j,k} = \sum_{t \in T_k} \sum_{i \in I_j} v_i {s_t}^T \tag{5.5}$$

$$loss_L = \sum_i \sum_t \max(0, 1 - y_{i,t} v_i {s_t}^T) \tag{5.6}$$

$$loss = \alpha loss_G + \beta loss_L \tag{5.7}$$

This formulation of alignment score allows a text fragment to align with multiple regions. In the early training epochs, $y_{i,t}$ is defined as +1 when v_i and s_t occur together in a correct image-text pair (i.e. $j = k$ for $i \in I_j, t \in T_k$), and -1 otherwise. In the later epochs, $y_{i,t}$ is adjusted by Multi-Instance Learning (MIL) [27]. $y_{i,t}$ is +1 only if in a correct pair, $v_i {s_t}^T > 0$ or $i = \text{argmax}_{i' \in I_j}(v_{i'} {s_t}^T)$. The overall loss function is a weighted linear combination of the local loss (5.6) and the global loss (5.4) with biases $\alpha = 0.5$ and $\beta = 1.0$.

For testing, the alignment scores $A_{j,k}$ are calculated using the trained parameters (W_v, b_v and W_s, b_s). The image search (i.e. use a text sample to query the most likely image) is done by fixing a text sample k and ranking the alignment scores of all candidate images.

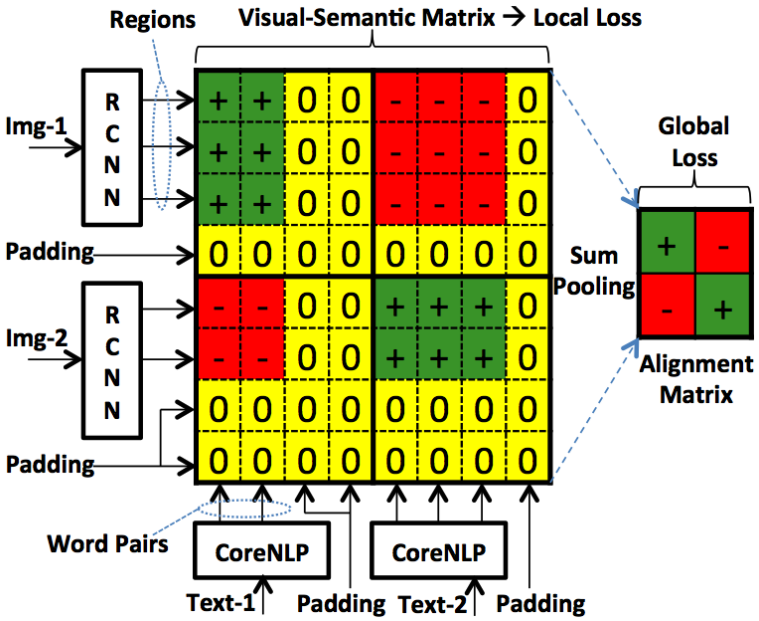


Fig. 5.2: Computation of Alignment Matrix

And the text search is similar by ranking the text candidates with a fixed image j.

5.3.3 Speed-up with Fragment Padding

For an optimization mini-batch H, the inner products of all image and text fragments ($v_i {s_t}^T, i \in I_j, t \in T_k$ and $j, k \in H$) form the *visual-semantic matrix*. We call the stacked $A_{j,k}$ of all text-image pairs the *alignment matrix*, from which the global loss can be quickly obtained with a few matrix operations. Essentially, an entry in the alignment matrix ($A_{j,k}$) is computed as the sum of all elements in its corresponding visual-semantic sub-matrix (a *patch*). Since images may have different number of regions and paragraphs are also diversified in the number of words, the sizes of the patches differ from each other. To calculate these alignments using theano [9], a straightforward implementation is to use a *scan* node to loop over the dimensions. However, it results in slow computation.

To improve the performance, we insert padding fragments (Fig. 5.2) to both the images and the texts. The j^{th} image fragment bag ${B_j}^v$ and the k^{th} text fragment bag ${B_k}^s$ are padded with zero fragments as in equation (5.8), in which all the texts and images will have the

same number of fragments. The resultant patches in the visual-semantic matrix are of size $(N^v \times N^s)$.

$$
\begin{aligned}
{B_j}^v &= \{R_i | i \in I_j\} + \{0\} \times {N_j}^v, \text{ s.t. } |{B_j}^v| = N^v \\
{B_k}^s &= \{ \begin{bmatrix} {w_t}^p \\ {w_t}^c \end{bmatrix} | t \in T_k\} + \{0\} \times {N_k}^s, \text{ s.t. } |{B_k}^s| = N^s
\end{aligned}
\tag{5.8}
$$

These padding fragments produce inner products of zeros, and thus will not contribute to the local loss or the global loss. But with the equally sized patches, we can use a standard sum-pooling process supported by theano to obtain the alignment matrix. The pooling operations are optimized in software implementations and better for vectorization than the loops do. Therefore, the padding helps accelerate the computation by removing the need of handling differently sized patches.

5.4 Text Fragment Filtering

A single stage of match embedding works well for short sentences that densely correlate with the images. However, loosely coupled texts such as picture news pose new challenges to the model. Since a lot of words in the news articles are not explicitly describing the images, they may cause overfitting and divert the optimization from those text fragments that really differentiate. In order to filter out the interfering fragments, we propose a fragment importance measure, and use a sequential architecture to improve the text-image association.

5.4.1 Fragment Importance Measure

The i, t^{th} entry in the visual-semantic matrix indicates how well the image fragment i correlate with the text fragment t. When the text body contains noises, the dot products $(v_i {s_t}^T)$ may not produce the optimal matching, but they are still valid indicators of whether a frag-

ment is useful for the association optimization. We define p_t in equation (5.9) the importance measure of the t^{th} text fragments.

$$p_t = g_{j \in \text{Imgs}}(\sum_{i \in I_j} v_i {s_t}^T) \tag{5.9}$$

Here, $g_{j \in Imgs}$ is a *Reduction Function* (e.g. $\sum_j$) that applies to the image population. The idea is that the larger the score is, the more likely the text fragment receives diversified matching results over different image regions. If we can make these informative fragments contribute more to the association optimizer, then we have a better chance to achieve an accurate text-image matching.

5.4.2 Cascade Embedding Stages

By equation (5.9), we know which text fragments are more useful in the association. Now a model is needed to integrate the importance measure to the optimizer. We propose to connect two fragment embedding stages as the yellow path in Fig. 5.3. The text-image fragments are passed to the first stage to train the *Filter Embedding*. The first stage does not produce the ranking, but outputs the fragment importance measures for the texts. The measures are converted to text weights that are applied to the fragments at the second embedding stage. The second stage, *Match Embedding* is trained with the filtered text fragments with weighted contributions to the loss. Match embedding produces the improved alignment matrix that generates the final ranking results.

Filter embedding is trained to identify the fragment importance. The importance measures are converted to text weights using equation (5.10).

$$m_t = \frac{|T_k|}{\sum_{t' \in T_k} p_{t'}} p_t, \forall t \in T_k \tag{5.10}$$

Here, the weight is essentially the normalized fragment importance measure with respect to

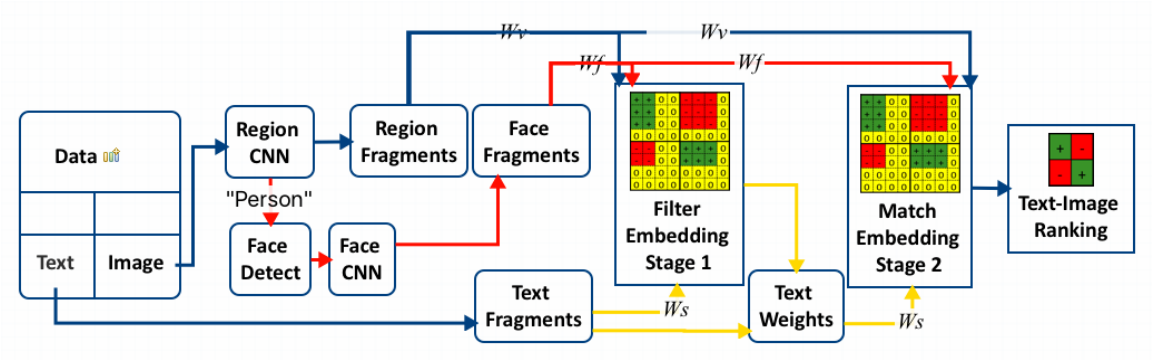


Fig. 5.3: Configuration of fragment filtering and fragment enrichment

the number of the fragments in the belonging text. While favoring those informative words, the normalization keeps the total "energy" of the text samples the same (i.e. $\sum_{t \in T_k} m_t = |T_k|$) to prevent large swings of the training loss.

The text weights are applied to the original text fragments. The second stage is then trained with the weighted text fragments defined in equation (5.11). Activation f is *ReLU*.

$$s_t = f[W_s(m_t \begin{bmatrix} {w_t}^p \\ {w_t}^c \end{bmatrix}) + b_s) \tag{5.11}$$

In this configuration, the word vectors are multiplied by the weight values m_t so that the important fragments are given higher weights and vice versa. The idea is that the non-informative fragments contribute less to the loss function in equation (5.7), so that the parameters W_s and b_s are able to "focus" on the important word fragments that are not discriminated optimally during the first stage. The dot products related to the noisy text fragments are forced to be near zero by the weights. Thus no matter how the parameters interact with the noisy words, they do not affect the final text-image ranking much.

5.5 Image Fragment Enrichment

The image features extracted by RCNN are tuned for object recognition. For text-image datasets with descriptive sentences, the level of knowledge is sufficient since the sentences are directly describing the objects in the images. However, in picture news, the images may contain different levels of meanings that cannot be captured by the object features. For example, the identities of the persons appeared in the news picture may help differentiate events, and thus improve the association learning, but simply recognizing the person objects doesn't provide such information. Therefore we use image fragments extracted by different CNN's to enrich the image understandings. Specifically, we extract the face features from the images.

Fig. 5.3 red path shows the workflow of extracting face fragments. The regions classified as "person" by RCNN are passed to the DPM face detector [64] to find the accurate face area. Then using VGG Face Descriptor [75], we extract the face features. The face features are then converted into the embedding space by equation (5.12).

$$v_l = W_f[\text{CNN}_F(R_l)] + b_f \tag{5.12}$$

The deep network $\text{CNN}_F(.)$ converts the pixels of the l^{th} detected face, R_l into a 4096-dimensional feature vector. Parameter W_f and b_f turns the face features into the image embedding v_l. The face fragments are placed along with the other image fragments to compute the alignment matrix. Because the number of faces detected in each image is small (usually less than 3), the face alignment score resembles equation (5.3). The final alignment score with face feature integrated is redefined in equation (5.13).

$$A_{j,k} = \sum_{t \in T_k} (\sum_{i \in I_j} v_i {s_t}^T + \text{max}_{l \in F_j} v_l {s_t}^T) \tag{5.13}$$

where F_j is the set of faces detected in the j^{th} image.

5.6 Evaluations

5.6.1 Datasets

Two datasets are investigated to evaluate the quality of the methods.

Pascal1k with noises. Pascal1k [81] dataset contains 1000 images, each of which is annotated by 5 independent sentences. We append to the sentences random texts grabbed from news articles, and the 5 sentences of each image have the same random text added. With this setup, we know that the first sentence of each text sample always contains the most information. This setup serves as a synthetic baseline for text filtering.

Reuters Picture News. We develop a crawler to download the thumbnail images along with their news articles from Reuters Picture News [82]. For each of the news categories (Scitech5k, Business5k and Politics5k), 5000 images with their associated articles longer than 70 words are collected. We also build a dataset of 15000 samples (Mixed15k) with news from all three categories. For face fragment evaluation, we construct a dataset of 1000 samples (People1k) with faces detected by RCNN [35] and DPM face detector [64].

5.6.2 Comparison Methods

For comparative study in image-text retrievals, we reproduce 4 baseline models.

Joint topics. We use K-means to cluster the RCNN [35] image regions into 1000 discrete visual words, and train an LDA [11] model with the joint corpus of both visual words and semantic words (similar to MixedLDA [33]). The LDA model is trained with 800 latent topics. Using LDA, the probability of each visual (semantic) word can be inferred from the latent topic distribution, which is inferred from the query bag of semantic (visual) words. We use the sum of the logarithm likelihoods of the visual (semantic) words as the alignment score.

DeViSE [34]. The work connects the modalities by minimizing the alignment loss between single words and images. It does not handle image or text as bag of fragments,

but it can be treated as a special case for the fragment embedding. The word vectors in a paragraph are averaged (L2-normalized) to one word fragment, and the regions detected in the same image are summed up to one image fragment. Only the global loss in equation (5.4) is applied during the optimization.

DeFrag [48]. The approach improves the performance by breaking the text and image into fragments. The fragment embedding is optimized by a mixed objective (global loss + local loss + MIL). We implement DeFrag with theano [9] for our customized configurations, and use it as the building blocks for our cascade configuration described in Section 5.4.

DepTree edges [49]. This method is the simplified extension of DeFrag. It removes the local loss, and uses the alternative alignment calculation defined by Equation (5.3).

5.6.3 Experiment Setup

For embedding optimization, we use stochastic gradient descent with momentum of 0.9. For Pascal1k and the noisy version, the dimension of embedding space (i.e. v_i and s_t) is 700, the mini-batch contains 35 text-image pairs and the reduction function g_j for equation (5.9) is variance var_j. For the picture news datasets, we use 1000-dimensional embedding, mini-batch size of 100 and the sum reduction function $\sum_j$. For all datasets, 80% of the samples are used for training and the rest two populations of 10% samples are used for validation and testing respectively. Take Pascal1k for example, we use 800 samples for training, 100 for validation and 100 for testing. Both DeFrag and our model use MIL [27] for the local losses.

For retrieval tests, we follow the description in section 5.3.2. The performance metrics are **R@K** and **Med** (Table 5.1 5.2). R@K is the percentage of the correct alignments that are ranked among the top K retrieval results (higher is better). Med is the medium rank of the correct samples (lower is better).

Table 5.1: Text-image retrieval results on Pascal1k

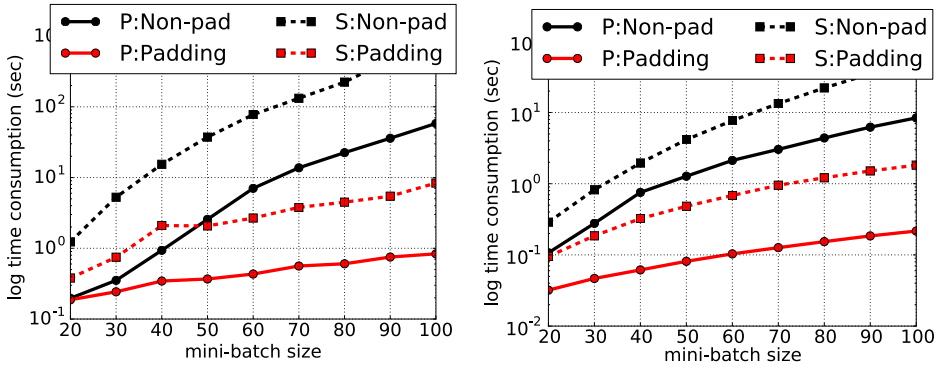
	Image Retrieval				Text Retrieval			
Implementation	**R@1**	**R@5**	**R@10**	**Med**	**R@1**	**R@5**	**R@10**	**Med**
Non-padding	27.2	62.2	82.2	3.0	25.0	67.0	80.0	2.0
Padding	25.6	66.6	84.0	2.0	27.0	60.0	74.0	3.0

5.6.4 Improvement in Computation Speed

In this experiment, we evaluate the computation performance boost brought by padding the fragments to equal patches. The implementation without padding uses a *scan* node [9] to loop over patches of different sizes. For equal patches with padding fragments, a sum-pooling operation based on *images2neibs* [9] is used. We test the per-batch time consumptions for the optimization of loss equation (5.7).

Two platforms are tested with the datasets. On a laptop with Intel Core i5 4250U at 1.3GHz, the padding-based implementation (*:Padding) outperforms the loop-based implementation (*-Non-pad) on both Pascal1k dataset (P:*) and Scitech5k news dataset (S:*), and provides 10X to 100X speed-up (Fig. 5.4a). The sum-pooling operation is more suitable for vectorization than the scan node does. On our server with Intel Xeon W5580 at 3.2GHz and NVIDIA Tesla C2075 with 448 CUDA cores at 1.15GHz, our fragment padding also accelerates the optimization process (Fig. 5.4b). At 100 batch size, the per-batch runtime on Scitech5k dataset is around 2 seconds with fragment padding, while the loop-based implementation consumes more than 50 seconds. The pooling operation can better utilize the GPU resources. Although fragment padding produces slightly larger visual-semantic matrix, the removal of scan nodes provides substantial improvement in training speed.

We also test the retrieval performance for padding-based implementation on Pascal1k dataset [81]. As shown in Table 5.1, fragment padding (Padding) does not degrade the accuracy. It achieves equivalent performance as compared to the scan-based (Non-padding) approach, while significantly improves the computation speed.



(a) Laptop (b) Server

Fig. 5.4: Computation Speed Comparisons

5.6.5 Results of Text-Image Retrievals

In this section, we evaluate the accuracy of comparison models on both synthetic dataset and picture news (Table 5.2).

We first perform retrieval tests on Pascal1k with noises. Fig. 5.5 shows the output weights of the filtering embedding obtained from an example piece of text. It is observed that the filter is able to capture the informative text fragments, i.e. the original description, and suppress the noises that we appended. The method correctly identifies the first part of the text as the most important, and assigns it high weights. The retrieval performances for both texts and images are improved significantly compared to the baseline methods. The R@1 measures are about 40% better than the second best model, DeFrag. This validates our assumption of using the filter embedding to extract the useful part of the texts.

Secondly, evaluations are done on the real picture news of different categories. Generally, Joint topics do not perform well because clustering regions into words causes loss of visual information. It has relatively better results on People1k as the images usually contain less types of objects. Also, DeViSE does not associate the text-image pairs as good as those fragment-based approaches, because using only the whole picture and averaged word vectors loses the details of the images and texts. The DepTree edges method uses the simplified alignment formulation and only considers the global objective [49]. This assumes

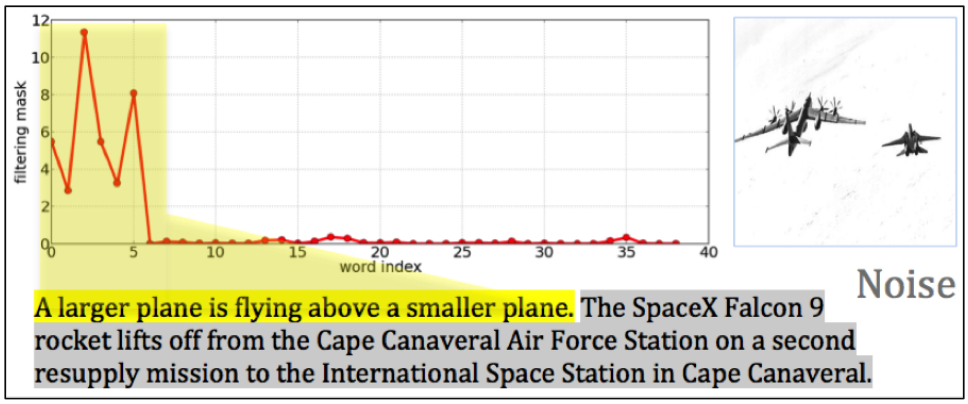


Fig. 5.5: Weight output from filter embedding on Pascal1k with noises

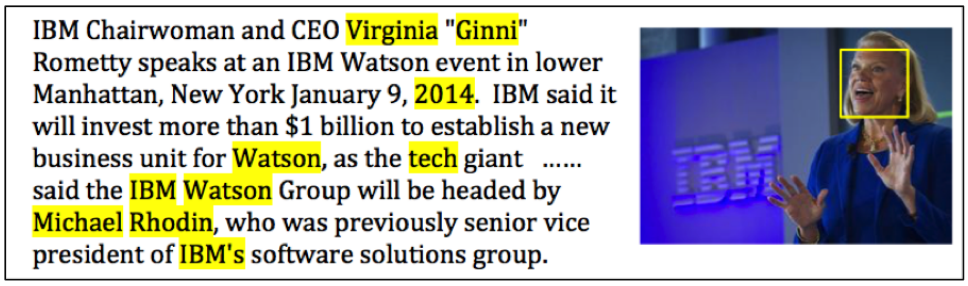


Fig. 5.6: Top 10 fragments with the highest dot products to the detected face

that each text fragment aligns to one image region. When many of the text fragments align to none or multiple regions, this assumption reduces accuracy. The ranking results (R@K) are worse than its more complete peer [48] with both local and global objectives.

The proposed method of text fragment filtering achieves substantial performance boost on the picture news mapping. On Scitech5k dataset, fragment filtering outperforms the second best method by around 10% in the ranking metrics. For Business5k dataset, our method produces better image ranking than those of the comparison methods, and competitive text ranking with DeFrag. On Politics5k dataset, fragment filtering still generates around 10% R@K improvement over the best baseline results. For Mixed15k, all approaches perform worse than they do on the smaller datasets, because Mixed15k is larger and more difficult. Our method adapts to the mixed news categories and produces the best ranking results among the comparison methods.

On People1k dataset, fragment filtering (F1) generates better ranking results over the

Table 5.2: Text-image retrieval results on noisy datasets

	Image Retrieval				Text Retrieval			
Model	**R@1**	**R@5**	**R@10**	**Med**	**R@1**	**R@5**	**R@10**	**Med**
Pascal1k with noises								
Joint topics	3.0	15.6	23.6	36.0	4.0	11.0	15.0	85.5
DeViSE	6.2	17.8	31.0	22.0	6.0	7.0	14.0	70.5
DepTree edges	4.8	20.0	36.2	16.5	6.0	20.0	23.0	38.0
DeFrag	12.6	42.2	63.6	6.0	11.0	31.0	43.0	14.5
Fragment filtering	17.6	51.2	68.8	4.0	16.0	43.0	55.0	8.0
Scitech5k								
Joint topics	4.6	12.0	15.8	119.5	4.2	8.8	12.2	183.0
DeViSE	3.2	14.0	24.0	38.5	5.0	18.4	30.4	30.0
DepTree edges	9.0	22.0	32.0	26.5	7.6	26.4	36.2	23.0
DeFrag	12.0	29.8	39.4	18.0	11.2	30.6	41.6	15.0
Fragment filtering	14.0	31.8	42.8	15.0	13.4	32.8	46.2	12.0
Business5k								
Joint topics	4.8	10.8	17.0	88.5	2.2	6.2	8.0	177.5
DeViSE	4.4	17.2	28.0	30.5	6.2	22.6	32.6	23.0
DepTree edges	6.4	22.6	33.4	23.5	7.2	26.8	37.0	17.0
DeFrag	11.6	31.2	41.2	14.0	12.8	36.2	48.4	10.5
Fragment filtering	11.2	33.0	45.8	12.5	13.0	36.2	47.2	12.0
Politics5k								
Joint topics	1.2	7.2	10.2	159.0	1.4	5.6	8.0	209.5
DeViSE	2.0	9.4	19.2	50.5	4.2	11.8	22.0	43.0
DepTree edges	4.2	15.4	21.4	46.0	7.0	19.8	31.6	35.0
DeFrag	8.0	22.2	31.0	25.0	1.8	22.8	33.2	22.0
Fragment filtering	9.0	25.6	35.2	19.5	8.2	26.8	36.2	19.5
Mixed15k								
Joint topics	1.7	4.7	6.6	426.0	1.4	2.7	3.7	579.5
DeViSE	2.0	8.7	14.1	88.5	2.5	10.4	17.3	69.5
DepTree edges	4.4	14.9	21.0	59.5	4.0	13.4	22.7	50.0
DeFrag	6.2	19.7	28.8	34.0	2.4	20.4	31.9	29.0
Fragment filtering	8.8	24.9	34.0	31.0	3.6	24.9	34.7	25.0
People1k								
Joint topics	13.7	25.5	31.4	27.0	10.8	21.6	29.4	27.5
DeViSE	2.9	13.7	25.5	23.0	4.9	23.5	41.2	17.5
DepTree edges	7.8	27.5	37.3	18.0	5.9	24.5	38.2	13.0
DeFrag	12.7	30.4	35.3	16.5	5.9	28.4	37.3	15.0
Fragment filtering (F1)	14.7	33.3	44.1	11.5	16.7	33.3	38.2	16.5
Face fragments (F2)	22.5	46.1	58.8	5.0	15.7	45.1	54.9	5.5
F1 + F2	31.4	46.1	59.8	6.0	22.5	47.0	58.8	5.5

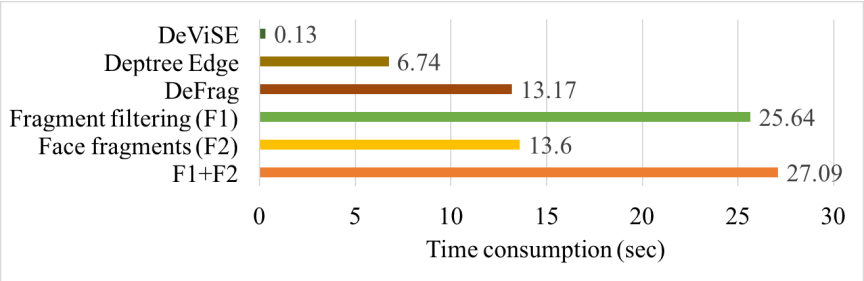


Fig. 5.7: Training time for 800 samples on People1k

first three methods. By integrating the deep face representations (F2), we outperform the best baseline approach DeFrag by 50% in the R@10 score. The face fragments provide another layer of context matching. The example in Fig. 5.6 highlights the child words of the dependent word pairs whose fragments produce the highest dot products $v_l {s_t}^T$ with the detected face. The face of IBM's CEO is strongly correlated with the company, her and her colleague's name, and "Watson". Some images that are previously not distinguishable can now be better identified by the person's facial characteristics. Finally, the combination of both text fragment filtering and image fragment enrichment (F1 + F2) obtains more accurate rankings compared to the two individual enhancements. It reaches 31.4 of R@1 for image retrieval.

Finally, Fig. 5.7 shows the times for training 800 samples on People1k data. DeViSE is fast because it does not handle fragments. So for a sample pair with 10 regions and 10 words, the size of the visual-semantic matrix is only 1/100 of the other methods'. DeFrag is slower than Deptree edge since the former optimizes both local and global costs. Fragment filtering sequentially connect the embedding stages, so the training complexity is about two times as that of DeFrag. Adding the face fragments only add a small overhead to the counterparts, because the number of faces in an image is usually small. The retrieval is accomplished by doing the forward pass with the network, so the time consumption is proportional to the training.

Fig. 5.8: Text Search Results using Example Image Queries

5.6.6 Qualitative Example of Article Search

In addition to the quantitative metrics that measure the quality of the embedding models, we also collect some example to show how the application works. In this section, both our sequential embedding method and DeFrag are tested with some image queries to locate the most relevant news article from the database. Basically, each input image is computed for alignment scores with respect to all candidate texts. Then the scores are ranked to see which text samples describe the image. The images and candidate texts are all from the Mixed15k dataset.

Fig. 5.8 demonstrates the examples. The left column shows the input images, and the two columns on the right shows the top 3 texts obtained by our text filtering method and the vanilla DeFrag method respectively. The text samples of red color are the ground truth results for the queries. On these three images of significantly different appearances, our model is able to rank the correct candidates in the first place, while the baseline method has degraded performance since the texts are very noisy. We observe that the baseline method are biased toward the same wrong candidate, which states that the Chinese president visited Seattle to meet the CEO's from a lot of American technology companies. We suspect that this article hits many keywords from sci-tech news and thus has a relative high alignment no matter what kinds of images are used as the query. When the baseline embedding cannot

differentiate the real answer, it picks the “safest” option.

5.7 Limitations

From the text side, the fragments rely on Stanford CoreNLP [62] to extract the dependency edges. This process is computationally expensive and may lose semantic information. BRNN [49] has used Recurrent Neural Networks to extract long-term concepts expressed by each word. Since our enhancements work in fragment level, it can be adapted to the BRNN fragments without much difficulty. Also, the proposed work only does reduction to the text fragments, but sometimes it is helpful to create richer text fragments (addition). For example, when seeing “Trump" and “Hilary" in the text, bringing up a new fragment such as “election" could improve the association learning. Therefore, inference-based model such as ITRS [79] can be integrated into the configuration. Finally, the improvement brought by the face descriptor is subject to the face detection, a more elastic approach is needed when the dataset lacks of facial information.

5.8 Conclusion

This chapter addresses the problem of associating images with noisy texts. We first modify the implementation by padding empty fragments to generate visual-semantic matrix with equally sized patches, which accelerate the speed of computation. Second, an embedding cascade configuration is designed to suppress the noisy part of the texts, so that in the second match embedding stage the optimization can be more effective in distinguishing the correct alignments. Third, we integrate face CNN to the image fragment generation in order to interpret richer information from the images. We show the improvements of our methods over the existing works on both synthetic dataset and real datasets of picture news.

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, we studied the neuromorphic learning systems and their implementation on non-traditional architectures. With both neural network and inference network, different network structuring methods were applied to fit supervised and unsupervised applications.

In Chapter 2, we discussed the confabulation based anomaly detection algorithms. We started by analyzing the sample complexity and its implications, and then developed the self-structuring procedure that automatically learn the network configuration. Comparison with classical anomaly detectors showed the ability of our method in capturing novel pattern from data streams and time series.

In Chapter 3, we exploited the parallel structure of the anomaly detection network, and designed fine-grained concurrent algorithms. The network inference was optimized using different computing platforms including CPU, GPU and Xeon Phi co-processor. Our concurrent algorithm provides significant speedup and scalability for realtime data streams.

In Chapter 4, the anomaly detection network was mapped to the bio-inspired TrueNorth chips. We designed novel spike burst coding scheme which provides compact representation and efficient implementations. The inference network, supported by our spike library,

offersed realtime processing while consuming very low power and energy.

In Chapter 5, we investigated the supervised learning problem for cross-modal retrievals. The gap between images and noisy texts are bridged with our fragment embedding network. We developed efficient network structure to handle non-ideal properties from both image side and text side. Our method achieved impressive retrieval accuracy compared to the state of the art.

6.2 Future Directions

6.2.1 Spiking Confabulation Network: Optimization of Hardware Resource Costs

Although our anomaly detection network achieved high computation performance and low energy/power consumption, it used almost 3000 neurosynpatic cores to handle the network intrusion data. We would like to reduce the core usage of the network and further reduce the power consumptions.

Currently, the knowledge matrices of the confabulation network are directly mapped to the crossbars. As most of the matrices are not fully populated, and the computation for anomaly detection can tolerate some degrees of imprecision, we propose to compress those knowledge matrices by row/column reordering and binning support symbols. By doing this, the crossbars number for each key lexicon can be reduced, and the excitation adder to handle fan-in larger than 256 can be eliminated. If proper prior knowledge about the data can be leveraged to the compression, we may preserve or even improve the anomaly detection quality.

All neurons in the key lexicons were activated during the inferences. But we observed that they are not necessary in some circumstances. As we know, the $t_{\max}$ symbol is only used for normalizing the anomaly scores with respect to context. Thus we don't need to locate the most likely symbol every time. We propose to randomly shut down a proportion

of the key neurons at each frame. In this way, the active power can be reduced since the many of the neurons are not firing. With efficient sampling method on the hardware, we can still provide accurate detections.

6.2.2 Text-Image Modeling: Generate Novel Sentence and Paragraphs

The current system assigned captions or articles for image queries using a candidate database. This was not always effective since such candidate set are hard to construct and they may not cover all the image descriptions.

The existing neural caption approach used recurrent neural network that treat image as the starting word. Such network usually requires large number of parameters and could suffer from overfitting. We propose to use fragment embedding method, which has less learnt parameters, to detect the word concepts from the image, and then use confabulation network to complete the sentences. In this way, we can release the neural networks from learning the language model, so that they can focus on the word-image associations.

Also, noisy text such as news articles are also hard to train for novel paragraph generation. Directly applying recurrent neural networks with very long article could cause weight vanishing. Using similar concept for short description, we propose to use the embedding network to find the topics the paragraph, use confabulation to construct the order of the topics, and then apply recurrent network on each topic to generate small pieces of sentences.

REFERENCES

[1] T. Abbes, A. Bouhoula, and M. Rusinowitch, "Efficient decision tree for protocol analysis in intrusion detection," *International Journal of Security and Networks*, vol. 5, no. 4, pp. 220–235, 2010.

[2] B. Ahmed, T. Thesen, K. Blackmon, Y. Zhao, O. Devinsky, R. Kuzniecky, and C. E. Brodley, "Hierarchical conditional random fields for outlier detection: An application to detecting epileptogenic cortical malformations." in *International Conference on Machine Learning (ICML)*, 2014, pp. 1080–1088.

[3] K. Ahmed, Q. Qiu, P. Malani, and M. Tamhankar, "Accelerating pattern matching in neuromorphic text recognition system using intel xeon phi coprocessor," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2014, pp. 4272–4279.

[4] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza *et al.*, "Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores," in *2013 International Joint Conference on Neural Networks*. IEEE, 2013, pp. 1–10.

[5] A. Andoni, R. Panigrahy, G. Valiant, and L. Zhang, "Learning polynomials with neural networks," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1908–1916.

[6] J. Backus, "Can programming be liberated from the von neumann style?: a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[7] E. Begoli, "A short survey on the state of the art in architectures and platforms for large scale data analysis and knowledge discovery from data," in *Proceedings of the WICSA/ECSA 2012 Companion Volume*. ACM, 2012, pp. 177–183.

[8] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *The Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[9] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a cpu and gpu math expression compiler," in *Proceedings of the Python for scientific computing conference (SciPy)*, vol. 4. Austin, TX, 2010, p. 3.

[10] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[12] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier, "Powerapi: A software library to monitor the energy consumed at the processlevel," *ERCIM News*, vol. 2013, no. 92, 2013.

[13] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *International Conference on Management of Data (SIGMOD)*, vol. 29. ACM, 2000, pp. 93–104.

[14] L. Buesing, J. Bill, B. Nessler, and W. Maass, "Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons," *PLoS Comput Biol*, vol. 7, no. 11, p. e1002211, 2011.

[15] J. B. Cabrera, C. Gutiérrez, and R. K. Mehra, "Ensemble methods for anomaly detection and distributed intrusion detection in mobile ad-hoc networks," *Information Fusion*, vol. 9, no. 1, pp. 96–119, 2008.

[16] Q. Cai, H. He, and H. Man, "Spatial outlier detection based on iterative self-organizing learning model," *Neurocomputing*, vol. 117, pp. 161–172, 2013.

[17] A. E. Cano Basave, Y. He, and R. Xu, "Automatic labelling of topic models learned from twitter by summarisation," in *Association for Computational Linguistics (ACL)*, 2014.

[18] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[19] A. S. Cassidy, R. Alvarez-Icaza, F. Akopyan, J. Sawada, J. V. Arthur, P. A. Merolla, P. Datta, M. G. Tallada, B. Taba, A. Andreopoulos *et al.*, "Real-time scalable cortical computing at 46 giga-synaptic ops/watt with," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*. IEEE, 2014, pp. 27–38.

[20] V. Chandola, A. Banerjee, and Kumar, "Anomaly detection for discrete sequences: A survey," *Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 823–839, 2012.

[21] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, p. 15, 2009.

[22] J. Z. Chang, R. T.-H. Tsai, and J. S. Chang, "Wikisense: Supersense tagging of wikipedia named entities based wordnet." in *PACLIC*, 2009, pp. 72–81.

[23] Q. Chen and Q. Qiu, "Enhancing bidirectional association between deep image representations and loosely correlated texts," in *2016 International Joint Conference on Neural Networks*. IEEE, 2016.

[24] Q. Chen, Q. Qiu, H. Li, and Q. Wu, "A neuromorphic architecture for anomaly detection in autonomous large-area traffic monitoring," in *International Conference on Computer-Aided Design (ICCAD)*. IEEE/ACM, 2013, pp. 202–205.

[25] Q. Chen, Q. Qiu, Q. Wu, M. Bishop, and M. Barnell, "A confabulation model for abnormal vehicle events detection in wide-area traffic monitoring," in *International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*. IEEE, 2014, pp. 216–222.

[26] Q. Chen, Q. Wu, M. Bishop, R. Linderman, and Q. Qiu, "Self-structured confabulation network for fast anomaly detection and reasoning," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015.

[27] Y. Chen, J. Bi, and J. Z. Wang, "Miles: Multiple-instance learning via embedded instance selection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 12, pp. 1931–1947, 2006.

[28] G. Creech and Hu, "A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns," *Transactions on Pattern Computer*, vol. 63, no. 4, pp. 807–819, 2013.

[29] G. Creech and J. Hu, "Generation of a new ids test dataset: Time to retire the kdd collection," in *Wireless Communications and Networking Conference (WCNC)*. IEEE, 2013, pp. 4487–4492.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[31] Economist, "http://www.economist.com/news/science-and-technology/21694540-win-or-lose-best-five-battle-contest-another-milestone," 2016.

[32] S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla *et al.*, "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2013, pp. 1–10.

[33] Y. Feng and M. Lapata, "Automatic caption generation for news images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 4, pp. 797–812, 2013.

[34] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov *et al.*, "Devise: A deep visual-semantic embedding model," in *Advances in Neural Information Processing Systems*, 2013, pp. 2121–2129.

[35] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014, pp. 580–587.

[36] J. Hawkins, S. Ahmad, and D. Dubinsky, "Hierarchical temporal memory including htm cortical learning algorithms," *Techical report, Numenta, Inc, Palto Alto*, 2011.

[37] S. Hawkins, H. He, G. Williams, and R. Baxter, "Outlier detection using replicator neural networks," in *Data Warehousing and Knowledge Discovery*. Springer, 2002, pp. 170–180.

[38] R. Hecht-Nielsen, *Confabulation Theory: The Mechanism of Thought*. Springer, 2007.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[40] M. Hodosh, P. Young, and J. Hockenmaier, "Framing image description as a ranking task: Data, models and evaluation metrics," *Journal of Artificial Intelligence Research*, pp. 853–899, 2013.

[41] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.

[42] E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng, "Improving word representations via global context and multiple word prototypes," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, 2012, pp. 873–882.

[43] H. Huang, "Rank based anomaly detection algorithms," Ph.D. dissertation, Syracuse University, NY, USA, 2013.

[44] Intel, "Intel xeon phi product family: Product brief," 2013.

[45] D. Ippoliti and X. Zhou, "A-ghsom: An adaptive growing hierarchical self organizing map for network anomaly detection," *Journal of Parallel and Distributed Computing*, vol. 72, no. 12, pp. 1576–1590, 2012.

[46] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*. ACM, 2014, pp. 675–678.

[47] I. Kang, M. K. Jeong, and D. Kong, "A differentiated one-class classification method with applications to intrusion detection," *Expert Systems with Applications*, vol. 39, no. 4, pp. 3899–3905, 2012.

[48] A. Karpathy, A. Joulin, and F.-F. Li, "Deep fragment embeddings for bidirectional image sentence mapping," in *Advances in neural information processing systems*, 2014, pp. 1889–1897.

[49] A. Karpathy and F.-F. Li, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3128–3137.

[50] R. Kiros, R. Salakhutdinov, and R. S. Zemel, "Unifying visual-semantic embeddings with multimodal neural language models," *arXiv preprint arXiv:1411.2539*, 2014.

[51] B. Klein, G. Lev, G. Sadeh, and L. Wolf, "Associating neural word embeddings with deep image representations using fisher vectors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4437–4446.

[52] D. C. Knill and A. Pouget, "The bayesian brain: the role of uncertainty in neural coding and computation," *TRENDS in Neurosciences*, vol. 27, no. 12, pp. 712–719, 2004.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[54] R. Laxhammar and G. Falkman, "Online learning and sequential anomaly detection in trajectories," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 6, pp. 1158–1173, 2014.

[55] R. Lebret, P. O. Pinheiro, and R. Collobert, "Phrase-based image captioning," *arXiv preprint arXiv:1502.03671*, 2015.

[56] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, 1995.

[57] L. Li, B. Roth, and C. Sporleder, "Topic models for word sense disambiguation and token-based idiom detection," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 1138–1147.

[58] M. D. Linderman, R. Bruggner, V. Athalye, T. H. Meng, N. Bani Asadi, and G. P. Nolan, "High-throughput bayesian network learning using heterogeneous multicore computers," in *International Conference on Supercomputing*. ACM, 2010, pp. 95–104.

[59] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," in *DARPA Information Survivability Conference and Exposition*, vol. 2. IEEE, 2000, pp. 12–26.

[60] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[61] L. Ma, Z. Lu, L. Shang, and H. Li, "Multimodal convolutional neural networks for matching image and sentence," *arXiv preprint arXiv:1504.06063*, 2015.

[62] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014, pp. 55–60.

[63] J. Mao, W. Xu, Y. Yang, J. Wang, and A. Yuille, "Deep captioning with multimodal recurrent neural networks (m-rnn)," *arXiv preprint arXiv:1412.6632*, 2014.

[64] M. Mathias, R. Benenson, M. Pedersoli, and L. Van Gool, "Face detection without bells and whistles," in *Computer Vision–ECCV 2014*. Springer, 2014, pp. 720–735.

[65] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[66] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[67] P. Mitra, C. Murthy, and S. K. Pal, "Unsupervised feature selection using feature similarity," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 301–312, 2002.

[68] R. Motwani and P. Raghavan, *Randomized algorithms*. Chapman & Hall, 2010.

[69] B. Nessler, M. Pfeiffer, L. Buesing, and W. Maass, "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity," *PLoS Comput Biol*, vol. 9, no. 4, p. e1003037, 2013.

[70] A. Ng and M. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 14, 2002, p. 841.

[71] Numenta, "The science of anomaly detection," 2014.

[72] NVIDIA, "Nvidia tesla c2075 companion processor," 2011.

[73] M. Oster and S.-C. Liu, "Spiking inputs to a winner-take-all network," *Advances in Neural Information Processing Systems*, vol. 18, p. 1051, 2006.

[74] E. J. Palomo, J. M. Ortiz-de Lazcano-Lobato, E. Domínguez, and R. M. Luque, "An anomaly detection system using a ghsom-1," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2010, pp. 1–7.

[75] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," *Proceedings of the British Machine Vision*, vol. 1, no. 3, p. 6, 2015.

[76] M. A. Petrovici, J. Bill, I. Bytschok, J. Schemmel, and K. Meier, "Stochastic inference with deterministic spiking neurons," *arXiv preprint arXiv:1311.3211*, 2013.

[77] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," in *Symposium on Computational Intelligence and Data Mining (CIDM)*. IEEE, 2007, pp. 504–515.

[78] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: A scalable simulator for an architecture for cognitive computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 54.

[79] Q. Qiu, Q. Wu, M. Bishop, R. E. Pino, and R. W. Linderman, "A parallel neuromorphic text recognition system and its implementation on a heterogeneous high-performance computing cluster," *Transactions on Computers*, vol. 62, no. 5, pp. 886–899, 2013.

[80] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 759–766.

[81] C. Rashtchian, P. Young, M. Hodosh, and J. Hockenmaier, "Collecting image annotations using amazon's mechanical turk," in *Proceedings of the NAACL HLT 2010*

Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk. Association for Computational Linguistics, 2010, pp. 139–147.

[82] "Reuters picture news," 2015, pictures.reuters.com.

[83] M. L. Shahreza, D. Moazzami, B. Moshiri, and M. Delavar, "Anomaly detection using a self-organizing map and particle swarm optimization," *Scientia Iranica*, vol. 18, no. 6, pp. 1460–1468, 2011.

[84] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[85] F. Simmross-Wattenberg, J. I. Asensio-Perez, P. Casaseca-de-la Higuera, M. Martin-Fernandez, I. A. Dimitriadis, and C. Alberola-Lopez, "Anomaly detection in network traffic based on statistical inference and alpha-stable modeling," *Transactions on Dependable and Secure Computing*, vol. 8, no. 4, pp. 494–509, 2011.

[86] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[87] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[88] J. Tian, M. H. Azarian, and M. Pecht, "Anomaly detection using self-organizing maps-based k-nearest neighbor algorithm," in *Proceedings of the European Conference of the Prognostics and Health Management Society*, 2014.

[89] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[90] F. Viegas, G. Andrade, J. Almeida, R. Ferreira, M. Gonçalves, G. Ramos, and L. Rocha, "Gpu-nb: A fast cuda-based implementation of naïve bayes," in *International Symposium on Computer Architecture and High Performance Computing*. IEEE, 2013, pp. 168–175.

[91] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *arXiv preprint arXiv:1411.4555*, 2014.

[92] K. Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *arXiv preprint arXiv:1502.03044*, 2015.

[93] J. Yin, D. H. Hu, and Q. Yang, "Spatio-temporal event detection using dynamic conditional random fields." in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 9, 2009, pp. 1321–1327.

[94] Z. Zhao, K. G. Mehrotra, and C. K. Mohan, "Ensemble algorithms for unsupervised anomaly detection," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2015, pp. 514–525.

[95] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in Neural Information Processing Systems*, 2014, pp. 487–495.

[96] A. Zimek, M. Gaudet, R. J. Campello, and J. Sander, "Subsampling for efficient and effective unsupervised outlier detection ensembles," in *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. ACM, 2013, pp. 428–436.

[97] “Uci machine learning repository,” http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

VITA

NAME OF AUTHOR: Qiuwen Chen

PLACE OF BIRTH: Longyan, Fujian, China

DATE OF BIRTH: January 14, 1987

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

M.Sc. 2012, Beijing University of Posts and Telecommunications, China

B.Sc. 2009, Beijing University of Posts and Telecommunications, China

PROFESSIONAL EXPERIENCE:

- Research Assistant, 2012 - Present, Syracuse University, Syracuse, US
- Software Development Engineer Intern, 2015, Microsoft, Redmond, US
- Software Development Engineer Intern, 2012, Alcatel-Lucent, Beijing, China
- Research Assistant, 2009 - 2012, Beijing University of Posts and Telecommunications, Beijing, China
- Baseband Engineer Intern, 2011, Potevio Institute of Technology, Beijing, China