Inferring Origin Flow Patterns in Wi-Fi with Deep Learning

Youngjune L. Gwon
Harvard University

H. T. Kung
Harvard University

Abstract

We present a novel application of deep learning in networking. The envisioned system can learn the original flow characteristics such as a burst size and inter-burst gaps conceived at the source from packet sampling done at a receiver Wi-Fi node. This problem is challenging because CSMA introduces complex, irregular alterations to the origin pattern of the flow in the presence of competing flows. Our approach is semi-supervised learning. We first work through multiple layers of feature extraction and subsampling from unlabeled flow measurements. We use a feature extractor based on sparse coding and dictionary learning, and our subsampler performs overlapping max pooling. Given the layers of learned feature mapping, we train SVM classifiers with deep feature representation resulted at the top layer. The proposed scheme has been evaluated empirically in a custom wireless simulator and OPNET. The results are promising that we achieve superior classification performance over ARMAX, Naïve Bayes classifiers, and Gaussian mixture models optimized by the EM algorithm.

1 Introduction

Machine learning plays an increasingly important role in the complex, data-intensive tasks required by today's sensing and computing systems. A flow is a sequence of data packets sharing the same context (*e.g.*, TCP connection, media stream) sent from a source to its destination. Accurate knowledge about the flow characteristics such as a burst size (in number of packets or bytes) and inter-burst gap, as were originated at the source, can be used beneficially to manage scarce networking resources. One motivating example would be software-defined networking (SDN) [3], which can leverage detailed flow knowledge to program routers and access points (APs) for scheduling a congested data traffic or mitigating wireless interferences more intelligently.

This paper describes our first work in developing inference schemes to learn the original properties of a flow from packet sampling at a receiver that is not necessarily the destination of the flow. We focus on the case where the source and the receiver are Wi-Fi nodes, and there are other Wi-Fi nodes that transmit their own flows. In particular, the receiver for our case is a network node such as a Wi-Fi AP that forwards or broadcasts packets, being a spot of aggregating different flows. The key challenge is how to unravel the work of CSMA that introduces a complicated mixture of competing flows. We believe that the approach of this paper can be extended for various other wireless and wired networks.

1.1 The Problem

Figure 1 explains our origin flow inference problem. Wi-Fi node A is the source of flow $\mathbf{f}_\mathrm{A}$ transmitted to Node B, a Wi-Fi AP. We denote $\mathbf{x}_{\mathrm{A|B}}$ a sample of $\mathbf{f}_\mathrm{A}$ measured by B. We use vector notation $\mathbf{f}$ to represent an origin flow pattern over time, and $\mathbf{x}$ its measurement. (Section 2 will explain how we describe patterns of a flow in detail; for now, consider $\mathbf{f}$ and $\mathbf{x}$ finite sequences of numbers.) Notice that there are other Wi-Fi nodes in the channel, namely nodes C, D, E, F, and G, that transmit own flows, creating contentions.

Distributed Coordination Function (DCF) provides the fundamental mechanism to access wireless media for the IEEE 802.11 Wi-Fi [1]. DCF employs Carrier Sense Multiple Access (CSMA) with a random backoff drawn from an exponentially growing window. Mixed with other transmissions, the sample $\mathbf{x}_{\mathrm{A|B}}$ could hardly preserve the original patterns in $\mathbf{f}_\mathrm{A}$. For example, the received packet burst lengths and gaps between bursts can be altered significantly. The exactness of such alteration is difficult to estimate, but there are both linear (*e.g.*, geometric increase of burst lengths) and nonlinear (*e.g.*, packet loss, retransmission, timeout) distortions. Among all possible causes, the main culprit should be CSMA.

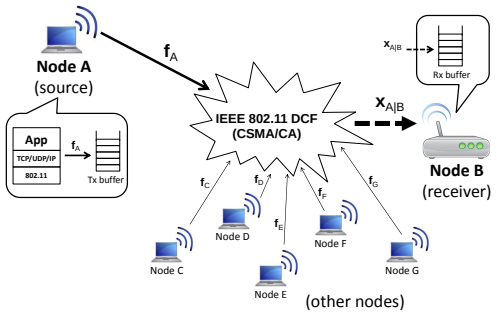

Figure 1: Flow inference problem illustrated

We aim to solve the following.

1. Classify received frame/packet pattern sampled at a receiver Wi-Fi node to the origin flow pattern;
2. Infer the original properties of a flow such as burst sizes and inter-burst gaps originated at the source.

We clarify several points. First, the *origin flow* pattern conveys context of application-level data. This is depicted under node A in Figure 1. $\mathbf{f}_A$ instantiates a pattern of the data formation mostly passed from the application layer (and the lesser from Transport/IP/MAC) to the transmit unit. Thus, our inference problem can lead to important understanding of application context mining. Secondly, our primary work domain is the MAC layer. We deduce observable patterns of a conventional TCP/UDP/IP flow from measuring directly the 802.11 MAC frames over time. Lastly, a sampled flow pattern at best is the origin flow pattern shifted (delayed) in time. For our inference to be effective, it is crucial to learn invariance such as some preserved original burst lengths that can spread widely over time.

1.2 Motivating Applications

Traffic monitoring capabilities are crucial for network management and security. Wireless bandwidth is among the most precious networking resources. Accurate origin flow inference can help derive efficient scheduling for wireless channels. The inferred information can also be used to classify legitimate traffic from malicious attacks.

Programming network nodes. Software-defined networking (SDN) is an emerging paradigm to build highly dynamic networks. Inferred origin traffic information can help program SDN nodes. For example, we can improve transmit-receive scheduling and avoid interferences.

Resource provisioning. The state-of-the-art networks can provision almost all networking resources elastically. The origin traffic inference will reveal the original properties of a flow that resource provisions such as communication bandwidth, flow cache, and compute cycles should strive to satisfy.

Queue management. A network node (*e.g.*, router, AP, switch) can leverage the source sending rates of large flows to manage its receive buffers and scheduling mechanisms dynamically. With knowledge on origin characteristics of a flow, networks can improve overall fairness.

1.3 Our Contributions

The main contribution of our work is to demonstrate the effectiveness of learning algorithms applied to an important networking problem mostly studied under parametric, model-based frameworks. Our approach is semi-supervised learning. We set up and train deep, unsupervised feature learning that constitutes multiple layers of sparse coding and pooling units. Given the learned feature mapping, we train classifiers in supervised learning. We have identified the key attributes for successful learning approaches to enhance our baseline such as forcing incoherency for sparse coding dictionary to extract more discriminative features, dense arrangement of sparse coding units, and max pooling on overlapping intervals. We have also explored and experimented with other learning methods from classical autoregressive time-series prediction and Naïve Bayes classifiers to the EM-optimized Gaussian mixtures. Our evaluation empirically confirms superior performance of the proposed learning methods in recovering the original properties of a flow.

1.4 Related Work

There is considerable work in model-based estimation for origin flow properties. Basu and Mukherjee [5] discuss numerous time-series models for Internet data traffic, including the autoregressive moving average process helpful for some of our formulation in Sections 2 and 3. Claffy *et al.* [9] present one of the earliest work to infer the original packet size distribution of a flow from packet sampling at routers. Duffield *et al.* [12] analyze methods to infer the original frequencies of flow lengths from sparse packet sampling.

The way sparse representations are used in computer vision and pattern recognition has inspired our method. Wright *et al.* [27] have developed a face recognition system that performs classification with sparse representation of features, which is based on a similar idea as ours. The idea of pooling sparse representations of features provides an important primitive to construct higher-level features as studied by Raina *et al.* [22], although pooling techniques date back to Riesenhuber and Poggio [24]. Coates and Ng [10] propose to pool over multiple features for deep learning.

Heisele, Ho, and Poggio [14] explain useful techniques of applying SVM for multi-class classification,

which is inherent in our origin flow pattern inference problem. There are a number of existing techniques to learn incoherent dictionary atoms. Ramirez *et al.* [23], Zhang & Li [28], and Lin *et al.* [18] have proposed similar ideas that force orthonormal dictionary columns as we maximize incoherency among dictionary columns in §5.1. Our idea of accompanying sparsity relaxation (§5.2) for sparse coding with forced incoherent dictionary atoms is new.

1.5 Outline

Section 2 explains the time-series representation and processing of a flow. In Section 3, we explore supervised learning methods for the origin flow inference. Section 4 describes our baseline semi-supervised learning method. We propose several enhancements to the baseline method in Section 5 and evaluate the learning methods with a custom simulator and OPNET in Section 6. The paper concludes in Section 7.

2 Time-series Representation of Flow

The runs-and-gaps model [16] gives a concise way to describe a flow. In Figure 2, characteristic patterns of an example flow are captured by packet *runs* and *gaps* measurable over time. As indicated earlier, we perform our flow measurements directly at the MAC layer rather than at the transport or IP layers by sampling and processing Wi-Fi frames.

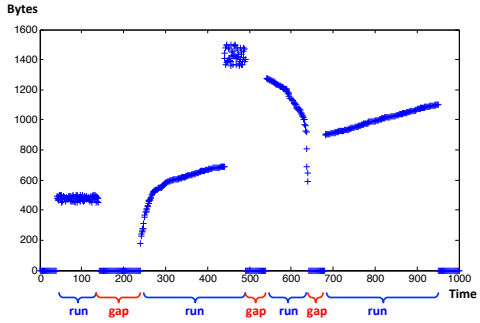

Figure 2: Runs-and-gaps model

Let $\mathbf{w} = [w_1\ w_2\ \ldots\ w_t\ \ldots\ w_N]$ be a vector containing the number of packets in a flow measured over N time intervals. Here, an important parameter is the *unit interval* T_s or *sampling period* during which each element w_t is sampled and recorded. The total measurement time or observation window is $N \times T_s$. Alternatively, we have vector $\mathbf{x} = [x_1\ x_2\ \ldots\ x_t\ \ldots\ x_N]$ for $\mathbf{w}$ where x_t is the corresponding byte count of the total payload at time t. Hence, a zero in $\mathbf{x}$ (or $\mathbf{w}$) indicates a gap. If $w_7 = 3$ and $x_7 = 1{,}492$, we have 3 packets for the flow at $t = 7$, and the sum of the payloads from the 3 packets is 1,492 bytes. We will call either $\mathbf{w}$ or $\mathbf{x}$ a measurable input vector for inference, which contains *extractable* features. While both $\mathbf{w}$ and $\mathbf{x}$ carry unique information, we mainly work with $\mathbf{x}$ throughout this paper.

We designate an origin flow (pattern) with another vector $\mathbf{f}$. Just like $\mathbf{x}$, $\mathbf{f}$ is a sequence of byte counts uniformly sampled, but the difference is that $\mathbf{f}$ reflects the initial pattern (or signature) originated at its source. Note $\mathbf{f} \in \mathbb{R}^M$ whereas $\mathbf{x} \in \mathbb{R}^N$, and M and N are not necessarily equal. We use notation $\mathbf{x}_{i,k}$ to refer kth measurement on flow i since there can be many measurements on $\mathbf{f}_i$. We also use $\mathbf{f}_{i,k}$ to designate the kth instance of origin flow i because there could be many origin patterns, or the pattern can be a stochastic process and changes dynamically over time. In summary, $\mathbf{f}$, $\mathbf{w}$, and $\mathbf{x}$ are all finite time-series representations of a flow.

Consider sampling and processing of three example flows in Fig. 3 at a receiver. The receive buffer first timestamps each arriving data frame and marks with flow ID. At $t = 1$, the received frame for flow 1 contains 2 packets whose payload sizes are 50 and 50 bytes, denoted in (2, 50/50B). At $t = 6$, flow 3 has two received frames. The first frame contains 2 packets with sizes 100 and 400 bytes whereas the second frame contains only one packet with 1,000 bytes. The example results in the following:

1. $\mathbf{w}_1 = [2\ 1\ 2\ 0\ 1\ 2]$, $\mathbf{x}_1 = [100\ 80\ 110\ 0\ 80\ 100]$
2. $\mathbf{w}_2 = [1\ 0\ 1\ 0\ 1\ 0]$, $\mathbf{x}_2 = [600\ 0\ 600\ 0\ 600\ 0]$
3. $\mathbf{w}_3 = [4\ 0\ 0\ 0\ 0\ 3]$, $\mathbf{x}_3 = [1500\ 0\ 0\ 0\ 0\ 1500]$

With $T_s = 10$ msec, each time series take 60 msec to measure. Flow 1 has 133.3 packets/sec, Flow 2 with 50 packets/sec, and Flow 3 with 116.7 packets/sec. In bit rates, they are 62.7, 240, and 400 kbps, respectively.

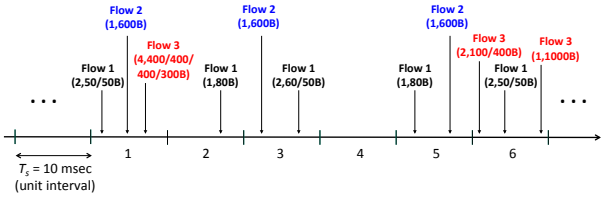

Figure 3: Time-series processing example

3 Origin Flow Inference with Supervised Feature Learning

The core of an inference system comprises a feature extractor (FE) and a classifier (CL) that need to be trained. Figure 4 describes the supervised learning framework. Supervised learning requires a labeled training dataset that consists of training examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ with corresponding desired output values (*i.e.*, labels) $\{l_1, \ldots, l_T\}$. There are two mappings, FE : $\mathbf{x} \rightarrow \mathbf{y}$ that

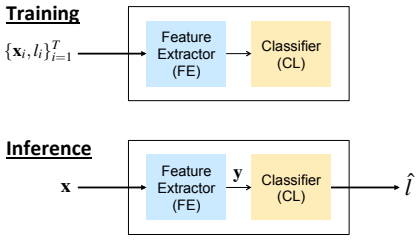

Figure 4: Supervised learning framework

maps an input $\mathbf{x}$ to its feature $\mathbf{y}$ and CL : $\mathbf{y} \to \hat{l}$ that performs classification on extracted features of the input. The inference system learns the mappings FE and CL from training examples and their labels. Once trained, when an unknown data $\mathbf{x}$ comes in, the system makes an inference by classifying it to a class $\hat{l}$.

Supervised learning for the origin flow inference problem considers a training dataset $\{\mathbf{x}_i, \langle \mathbf{f}_{l_i}, l_i \rangle\}_{i=1}^T$ collected at the Wi-Fi receiver of interest. $\mathbf{x}_i$ is a time-series representation of flow l_i sampled at the receiver. We also make $\mathbf{f}_{l_i}$ available, the corresponding origin flow time-series representation. So, when a measured sample $\mathbf{x}$ is classified as l_j, we can infer the original flow properties from looking up $\mathbf{f}_{l_j}$'s.

We now explore supervised learning methods. We note that most of these methods naturally lead to binary classifiers. Our origin flow inference problem, however, is a multi-class classification. We will revisit this issue in §4.3. For clarity of explanation, this section accompanies binary classification.

Autoregressive moving average with exogenous inputs (ARMAX) [19] is a widely-studied model for linear system identification. ARMAX models the current output of a system with the previous (delayed) output and input values. With ARMAX, we can directly estimate the origin flow time-series $\mathbf{f} = [f_1\ f_2\ \ldots\ f_{t-1}\ f_t]$ from the measurement $\mathbf{x} = [x_0\ x_1\ \ldots\ x_{t-2}\ x_{t-1}]$ in a linear difference equation: $f_t + a_1 f_{t-1} + \ldots + a_n f_{t-n} = b_1 x_{t-1} + \ldots + b_m x_{t-m} + \varepsilon$. Note that ε gives the model error, which itself can be written elaborately over time, *i.e.*, $c_1\varepsilon_{t-1} + \ldots + c_m\varepsilon_{t-m}$. The ARMAX matrix form is

$$\underbrace{\begin{bmatrix} f_t \\ f_{t-1} \\ \vdots \\ f_1 \end{bmatrix}}_{\mathbf{f}} = \underbrace{\begin{bmatrix} f_{t-1} & \cdots & f_{t-n} & x_{t-1} & \cdots & x_{t-m} \\ \vdots & & \vdots & \vdots & & \vdots \\ f_0 & \cdots & f_{1-n} & x_0 & \cdots & x_{1-m} \end{bmatrix}}_{\boldsymbol{\Phi}} \underbrace{\begin{bmatrix} -a_1 \\ \vdots \\ -a_n \\ b_1 \\ \vdots \\ b_m \end{bmatrix}}_{\boldsymbol{\theta}}$$

Under supervised learning, if we have many training examples $\{\mathbf{x}_i, \langle \mathbf{f}_{l_i}, l_i \rangle\}_{i=1}^T$, we will have a massively *over-constrained* system for our inference problem. Least squares can train $\boldsymbol{\theta}$. However, when the normal equation $\hat{\boldsymbol{\theta}} = (\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\mathsf{T}\mathbf{y}$ becomes unstable, the recursive least squares via Kalman filtering [8] can be used instead.

Naïve Bayes Classifier. The key for Naïve Bayes is to define a good feature extractor for a flow measurement $\mathbf{x}$. We use a feature $\mathbf{y} = \begin{bmatrix} \hat{\mu}_{\text{run\_size}} & \hat{\mu}_{\text{gap\_length}} \end{bmatrix}$ by computing the sample mean values of run size (bytes) and gap length (unit intervals) from $\mathbf{x}$. The classifier is constructed from computing empirical conditional distribution $p(\mathbf{y}|l_i)$ of the train dataset. In runtime, the trained classifiers infer the origin flow: l_i if $p(l_i|\mathbf{y}) \geq p(l_j|\mathbf{y}) \Leftrightarrow p(\mathbf{y}|l_i)p(l_i)/p(\mathbf{y}|l_j)p(l_j) \geq 1$. Here, we use a simple decision rule that compares the learned likelihood ratios $p(\mathbf{y}|l_i)$ and $p(\mathbf{y}|l_j)$ for binary classification.

Support vector machine (SVM). Boser, Guyon & Vapnik [6] first proposed SVM. SVM is a binary classification model searching for a hyperplane that maximizes separation between two classes. The hyperplane is orthogonal to the shortest line connecting the convex hulls of the classes. Support vectors are the data points along the shortest line. The hyperplane has the form $h(\mathbf{x}) = \sum_{i=1}^T \alpha_i l_i \mathbf{x}_i \cdot \mathbf{x} + b$, where $\mathbf{x}_i$ is a training example for flow i, and α_i, b the solution of a quadratic programming (QP) problem. Class label $l_i \in \{-1, 1\}$ for each $\mathbf{x}_i$ must be provided during the training. Classification of a runtime input $\mathbf{x}$ computes $\text{sign}(h(\mathbf{x}))$.

The SVM kernel trick can work with nonlinearity of data. A kernel function $K(.)$ (*e.g.*, radial basis, sigmoid) maps the input $\mathbf{x}$ to a higher dimensional feature space where a better margin is possible. The hyperplane with the kernel trick becomes $h_{\text{kern}}(\mathbf{x}) = \sum_{i=1}^T \alpha_i l_i K(\mathbf{x}_i, \mathbf{x}) + b$.

Gaussian mixture model (GMM). Strictly speaking, GMM is an unsupervised feature learning method that is later paired with supervised classifier training. GMM assumes the probability density of input data as a weighted sum of K Gaussian distributions. GMM can be thought as a model-based version for K-means clustering. Parameterized by $\{w_j, \mu_j, \Sigma_j\}_{j=1}^K$, the feature for an input $\mathbf{x}$ is a combination of posterior membership probabilities evaluated by the Gaussians. For jth Gaussian, we have

$$p_j(\mathbf{x}) = \frac{1}{(2\pi)^{N/2}|\Sigma_j|^{1/2}} \cdot \exp\left[-\frac{1}{2}(\mathbf{x}-\mu_j)^\top \Sigma_j^{-1}(\mathbf{x}-\mu_j)\right]$$

Expectation maximization (EM) [11] trains GMMs iteratively. In E-step, EM creates a function evaluating the expected log-likelihood with the current estimate of the parameters. M-step computes new parameter values that maximizes the expected log-likelihood of the E-step.

4 Origin Flow Inference with Deep Learning

Deep learning refers to multiple layers of extracting features and nonlinear aggregation of the extracted features.

This section presents our deep learning approach for the origin flow inference problem.

4.1 Overview

We propose an inference system based on semi-supervised learning. At the first stage, the system performs *unsupervised* feature learning over multiple layers.

1. Do sparse coding and dictionary learning with unlabeled training dataset
2. Pool sparse representations of the training dataset to reduce the number of features
3. Pass the resulting features (*i.e.*, pooled sparse representations) to next layer and repeat by treating current layer's features as input data for next layer

Given multiple layers of the learned dictionaries and features, the system next performs *supervised* learning.

1. Do multi-layer sparse coding and pooling with labeled training dataset
2. Train (linear) SVM classifiers with the final form of feature vector resulted at the top layer

At runtime, the system takes a sample measurement of a flow, performs the multi-layer inference (*i.e.*, sparse coding and pooling), and predicts the origin flow pattern and properties.

4.2 Unsupervised Feature Learning

4.2.1 Sparse coding

We use sparse coding [20] as the primary means to extract features from the sampled time-series data. Consider unlabeled dataset $\{\mathbf{x}_k\}_{k=1}^T$ with each $\mathbf{x}_k \in \mathbb{R}^N$. We pack $\{\mathbf{x}_k\}_{k=1}^T$ to the columns of $\mathbf{X} = \left[\mathbf{x}_1^\top\ \mathbf{x}_2^\top\ \ldots\ \mathbf{x}_T^\top\right]$. Note $\mathbf{X} \in \mathbb{R}^{N\times T}$. Sparse coding requires a dictionary $D \in \mathbb{R}^{N\times P}$ learned from $\mathbf{X}$. We adopt K-SVD [4] that learns D in the following optimization

$$\min_{D,\mathbf{Y}} \|\mathbf{X} - D\mathbf{Y}\|_F^2 \quad \text{s.t. } \|\mathbf{y}_k\|_0 \le K\ \ \forall k \tag{1}$$

Here, the columns of $\mathbf{Y} \in \mathbb{R}^{P\times T}$ or $\{\mathbf{y}_k\}_{k=1}^T$, are the sparse representations of $\{\mathbf{x}_k\}_{k=1}^T$. (Note $\mathbf{y}_k \in \mathbb{R}^P$.)

K-SVD is a fast iterative algorithm and requires to compute sparse code $\mathbf{y}_k$ for each $\mathbf{x}_k$ with current D. We use orthogonal matching pursuit (OMP) [21] for computing sparse codes. Our choice of OMP is merely based on its computational efficiency, and there are other algorithms such as LASSO [26] and LARS [13] that also work well.

The columns of D, $\{\mathbf{d}_j\}_{j=1}^P$, are dictionary atoms. Hence, each element in vector $\mathbf{y}$ reflects a degree of membership to the corresponding dictionary atom. To represent unbiased membership, dictionary atoms are normalized such that $\|\mathbf{d}_j\|_{\ell_2}^2 = 1$. To make every $\mathbf{d}_j$ meaningful, we need more training samples than dictionary atoms, so $T \ge P$. For discriminative convenience, D is overcomplete—*i.e.*, $P > N$. This means that sparse code $\mathbf{y}$ has a higher dimensionality than $\mathbf{x}$, but $\mathbf{y}$ is K-sparse, that is, only $K \ll N$ entries of $\mathbf{y}$ are nonzero.

4.2.2 Max pooling

Every input vector $\mathbf{x}_k$ is mapped to the corresponding feature vector $\mathbf{y}_k$ via sparse coding. If all feature vectors were used straightforwardly, we could overwhelm the unsupervised feature learning process. It is customary to reduce the number of extracted features by subsampling (or summarizing over) feature vectors—note, this is by no means to discard any useful information.

Pooling, popular in convolutional neural networks [17], operates over multiple (sparse) feature representations and aggregates to a higher level of features in reduced dimension. An important property of pooled feature representation is translation invariance [24]. We use *max* pooling [7] that takes the maximum value for the elements in the same position over a group of feature vectors. For example, consider max pooling of L sparse codes $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L\}$ that yields $\mathbf{z} = \mathsf{max\_pool}(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L)$ in Figure 5. Noting $\mathbf{y}_k = [y_{k,1}\ \ldots\ y_{k,P}]$ and $\mathbf{z} = [z_1\ \ldots\ z_P]$, max pooling results in $z_j = \max(y_{1,j}, y_{2,j}, \ldots, y_{L,j})$.

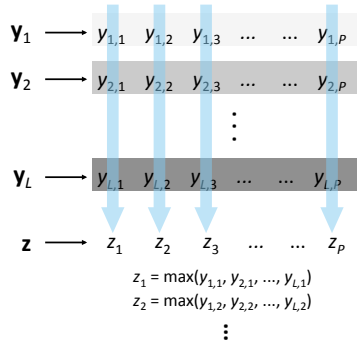

Figure 5: Max pooling of L sparse codes

4.2.3 Multi-layer deep learning

Figure 6 presents our multi-layer deep learning. We use 2 layers. Each layer trains own dictionary and has separate sparse coding and pooling units. Assuming input vector $\mathbf{x}_k$ has a moderate size N, we perform batch processing of multiple $\mathbf{x}_k$'s concatenated in series. Figure 6 showcases 4 input vectors with pooling unit configured at $L = 2$. If $\mathbf{x}_k$ has a very large N on the other hand, $\mathbf{x}_k$ can be divided

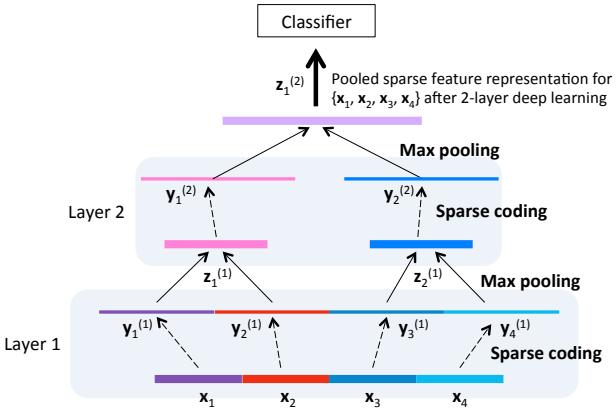

Figure 6: Baseline 2-layer deep learning with sparse coding and max pooling

into subpatches, and we perform sparse coding on each subpatch. The input vector length for sparse coding is an important system parameter for deep feature learning.

Max pooling is performed over sets of two consecutive sparse representations $\{\mathbf{y}_1^{(1)}, \mathbf{y}_2^{(1)}\}$ and $\{\mathbf{y}_3^{(1)}, \mathbf{y}_4^{(1)}\}$. We use notation $\mathbf{y}_k^{(1)}$ for kth sparse code at layer 1. The intermediate pooled features, $\mathbf{z}_1^{(1)}$ and $\mathbf{z}_2^{(1)}$, are sent to layer 2 for another round of sparse coding and max pooling. At the top, $\mathbf{z}_1^{(2)}$—obtained by pooling the layer 2 sparse codes $\mathbf{y}_1^{(2)}$ and $\mathbf{y}_2^{(2)}$—gives the final feature representation for $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ in this 2-layer deep learning.

In general, a depth or the number of layers reflects the coverage of deep learning. Layer 1 extracts small, local features over multiple intervals spanned by consecutive input vectors. The resulting feature representations are subsampled with max pooling before passed to layer 2. Layer 2 builds larger features using its own dictionary. Because the feature coverage by layer 1 coding and pooling is over a subregion for the layer 2 coverage, the features aggregated at layer 2 are novel and could not be seen at layer 1.

4.3 Supervised Learning

We embrace the supervised learning that largely consists of training SVM classifiers. The SVM classification framework is generic and can directly work on $\mathbf{x}$ without any feature extraction or pooling. SVMs could be trained with a single-layer sparse representation $\mathbf{y}^{(1)}$ subject to $\mathbf{x} = D^{(1)}\mathbf{y}^{(1)}$. Under our 2-layer deep learning setup, we train linear SVM classifiers using the final pooled feature vectors $\mathbf{z}^{(2)}$.

Considering there are many data patterns generated by applications, the origin flow inference is a multi-class classification problem. There are two approaches for SVM, which is inherently a binary classifier, to classify q origin flow patterns. The first approach is to train all $\frac{q(q-1)}{2}$ 1-vs-1 SVMs exhaustively. Each SVM is dedicated to distinguish between any pair out of q origin flows and infers the original flow properties mapped by the classification result.

The second approach is to train q 1-vs-all SVMs. For flow i, training 1-vs-all SVM will require two datasets $\mathbf{X}_i$ with label $l_i = 1$ consisting of measured patterns of flow i only and $\mathbf{X}_{\setminus i}$ with label $l_j = -1$ containing measurements for all other flows $j\ \forall j \neq i$. Ideally, $\mathbf{X}_{\setminus i}$ should contain unbiased mix of the other flows. Our empirical evaluation in Section 6 considers the 1-vs-all approach.

5 Enhancements

5.1 Incoherent Dictionary Learning

Dictionary learning algorithms address the performance of sparse coding in two aspects: 1) reconstructive accuracy and 2) discriminative ability of the learned dictionary atoms (*i.e.*, the column vectors of D). We emphasize the latter aspect because our primary objective is classification rather than compressing data. Discriminative ability of a dictionary is related to making its atoms as *incoherent* as possible. Sparse coding with a dictionary consisting of more incoherent column vectors should improve the margin of an SVM, which results in a better classification performance.

The maximally incoherent D is constrained such that $D^{\mathsf{T}}D = I$. In other words, an incoherent dictionary matrix has orthonormal columns. This is equivalent to minimizing $\left\|D^{\mathsf{T}}D - I\right\|_F^2$. We can also think of having the two conditions $\mathbf{d}_k^{\mathsf{T}}\mathbf{d}_j = 0\ \forall k \neq j$ (orthogonal columns) and $\left\|\mathbf{d}_k^{\mathsf{T}}\mathbf{d}_k\right\|_2 = 1$ (normalized).

Since we use K-SVD, we add the incoherence optimization term to Equation (1)

$$\min_{D,\mathbf{Y}} \|\mathbf{X} - D\mathbf{Y}\|_F^2 + \gamma\left\|D^{\mathsf{T}}D - I\right\|_F^2 \text{ s.t. } \|\mathbf{y}_k\|_0 \leq K\ \ \forall k \tag{2}$$

The new optimization here, however, is not a trivial task. For the time being, we propose a two-stage algorithm presented below instead.

In the outer **for** loop, we run K-SVD unmodified. The resulting D then enters the inner **while** loop that implements the gradient descent algorithm [25] to *regularize* the incoherence term in Eq. (2). γ is the step size for gradient search and decayed by $0 < \delta < 1$ within the inner loop until initialized back to the default value γ_0 in the outer loop after running K-SVD with the next training vector.

Algorithm 1 Two-stage incoherent dictionary learning

Require: training dataset $\mathbf{X} = \left[\mathbf{x}_1^\top \ \ldots \ \mathbf{x}_T^\top\right]$

initialize $D := I$
for $i = 1$ to T
  $D := \mathsf{ksvd}(D, \mathbf{x}_k, K)$
  initialize $\gamma := \gamma_0$
  while $\left\|D^\mathsf{T} D - I\right\|_F^2 > \varepsilon$
    $D := D - \gamma D(D^\mathsf{T} D - I)$
    $D := \mathsf{normalize\_columns}(D)$
    $\gamma := \gamma \cdot \delta$
  end
end

5.2 Sparsity Relaxation

Strictly speaking, the way we force the dictionary incoherence is flawed. The gradient descent takes over after K-SVD, but K-SVD and the gradient descent regularization have to take place jointly as Equation (2) suggests. For this reason, the resulting effect of the outer loop computation perturbs the K-SVD optimization. In other words, improving the dictionary incoherence comes with the cost of reconstructive accuracy.

As a result, using the same value of K for sparse coding may be too tight to meet the minimal error criterion. Therefore, we must relax the original sparsity K for sparse coding to K' such that $K' > K$. We do sparsity relaxation as follows. Let D' represent the incoherent dictionary resulted from Algorithm 1. We repeat sparse coding with D' to find $\mathbf{Y}'$

$$\min_{\mathbf{Y}'} \left\|\mathbf{X} - D'\mathbf{Y}'\right\|_F^2 \quad \text{s.t. } \left\|\mathbf{y}'_k\right\|_0 \le K' \ \ \forall k$$

5.3 Dense Sparse Coding and Overlapping Pooling

The baseline deep learning scheme in Figure 6 does batch sparse coding of $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ and max pooling of $\{\mathbf{y}_1, \mathbf{y}_2\}$ and $\{\mathbf{y}_3, \mathbf{y}_4\}$. We can enhance this baseline by performing sparse coding on shifted $\mathbf{x}_k$'s and max pooling over the resulting, overlapping sparse codes. This is illustrated in Figure 7. The overlapping intervals are formed by shifting (*i.e.*, delaying) the elements in $\mathbf{x}_k$'s altogether by τ. Note that $\tau = 1$ gives the fully overlapped intervals while there is no overlapping for $\tau = 4 \cdot N$, which equals the baseline. (In our evaluation, we use a 95% overlap between consecutive $\mathbf{x}$'s.) Dense sparse coding can substantially reduce chances to miss a feature with increased cost of computing. Overlapping pooling further improves on the translational invariance of a feature. Also, according to Krizhevsky *et al.* [15], overlapping pooling reduces overfit in classifiers.

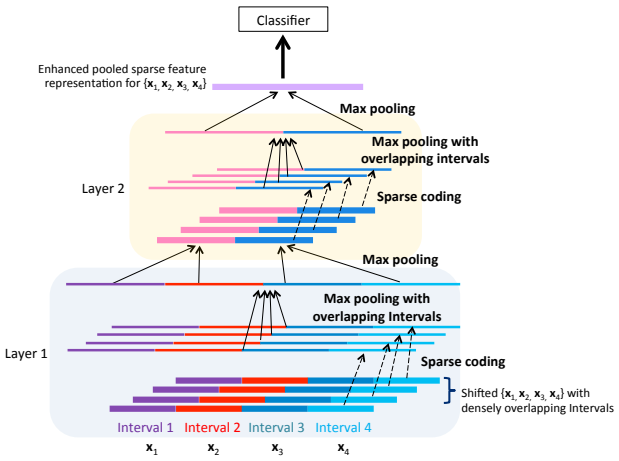

Figure 7: Enhanced 2-layer deep learning with dense sparse coding and overlapping max pooling

6 Evaluation

We evaluate the proposed baseline and enhanced inference schemes in comparison to classical machine learning approaches described in Section 3. We have implemented a simple setup featuring three Wi-Fi nodes in a custom MATLAB simulator and used OPNET Modeler to test more elaborated, seven Wi-Fi node scenario.

6.1 Methodology

6.1.1 Flow generation

We generate flows based on the runs-and-gaps model explained in Section 2. The triplet $\langle t_r, s_r, t_g \rangle$ describes the generative pattern of a flow, where t_r and t_g are the run and gap lengths in number of unit intervals (T_s), and s_r denotes the size of payload in bytes generated per each unit interval of a run. A flow type can be *constant*, *stochastic*, or *mixed*. A constant flow has deterministic t_r, s_r, and t_g values. For example, flow 1 with $\langle 2, 100, 4 \rangle$ creates the origin flow pattern (time series) $\mathbf{f}_1 = [100\ 100\ 0\ 0\ 0\ 0\ 100\ 100\ 0\ 0\ \ldots\ ]$. For a stochastic flow, t_r, s_r, and t_g are random variables. For example, flow 6 with $\langle \mathrm{Exp}(0.5), \mathrm{Pareto}(40, 1), \mathrm{Exp}(0.25) \rangle$ has exponentially distributed run and gap lengths (with mean of 1/0.5 and 1/0.25 unit intervals, respectively), and Pareto-distributed payload sizes. An instance of flow 6 could be $\mathbf{f}_6 = [518\ 97\ 0\ 0\ 0\ 0\ 0\ 0\ 32\ 0\ \ldots\ ]$.

We consider 10 origin flows summarized in Table 1. Notice we also use the normal (N) and uniform (U) distributions. We round fractions, discard negative numbers drawn from a normal distribution and regenerate. Using $T_s = 10$ msec, we generate 2,000 instances of time series for each flow. We use the first 1,000 instances for training and the other 1,000 for testing. Each instance is a vector

Table 1: Origin flows used for evaluation

Flow	Type	Generative triplet $\langle t_r, s_r, t_g \rangle$
Flow 1	Constant	$\langle 2, 100, 4 \rangle$
Flow 2	Constant	$\langle 2, 500, 2 \rangle$
Flow 3	Constant	$\langle 5, 200, 5 \rangle$
Flow 4	Constant	$\langle 10, 200, 10 \rangle$
Flow 5	Stochastic	$\langle$Exp(1), Pareto(100, 2), Exp(0.1)$\rangle$
Flow 6	Stochastic	$\langle$Exp(0.5), Pareto(40, 1), Exp(0.25)$\rangle$
Flow 7	Stochastic	$\langle$U(4, 10), Pareto(100, 2), Exp(0.5)$\rangle$
Flow 8	Stochastic	$\langle$N(10, 5), Pareto(40, 1), N(10, 5)$\rangle$
Flow 9	Mixed	$\langle$1, Pareto(100, 2), 1$\rangle$
Flow 10	Mixed	$\langle$1, Pareto(100, 2), Exp(0.25)$\rangle$

of 500 elements.

6.1.2 Preprocessing generated origin flow patterns

We precompute the mean run and gap lengths from the generated origin flow patterns in the training dataset. This is convenient because we enable simple lookup (of the precomputed values) based on the classification result of a measured flow in order to estimate the origin run and gap properties. In Figure 8, we have $\left[s_1^1\ s_2^1\ 0\ 0\ 0\ s_1^2\ 0\ 0\ 0\ 0\ 0\ s_1^3\ s_2^3\ s_3^3\ 0\ 0\ \ldots\right]$, where $s^1 = \sum_{k=1}^{2} s_k^1$, $s^2 = \sum_{k=1}^{1} s_k^2$, $s^3 = \sum_{k=1}^{3} s_k^3$ give total bytes of the three bursts. We can then compute the mean burst size for this pattern. We also compute $\{t_r^1, t_r^2, t_r^3, \ldots\}$, $\{t_g^1, t_g^2, t_g^3, \ldots\}$, and their mean values.

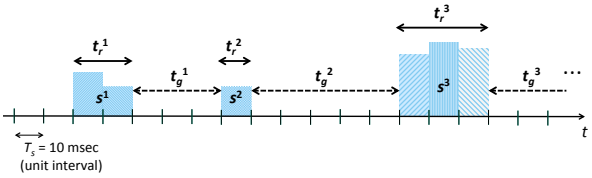

Figure 8: Computing generated flow statistics

6.1.3 Evaluation metrics

We are foremost interested in the accuracy of classifying a measured pattern $\mathbf{x}$ to its ground-truth origin flow pattern $\mathbf{f}$. We compute two metrics, *recall* (true positive rate) and *false alarm* (false positive rate), to evaluate classification performance:

$$Recall = \frac{\sum \text{True positives}}{\sum \text{True positives} + \sum \text{False negatives}}$$

$$False\ alarm = \frac{\sum \text{False positives}}{\sum \text{False positives} + \sum \text{True negatives}}$$

Without false alarm rate, we cannot truly assess the probability of detection for a classifier using a computed recall value because the classifier can be configured to declare positive only, automatically achieving to guess all positives correctly. Classification leads to inferring other important properties of a flow from its training dataset records. As our secondary evaluation metrics, we calculate errors in estimating the original mean burst size and mean gap length of the flow.

Table 2: Wi-Fi parameter configuration for Scenario 1

Parameter	Description	Value
aSlotTime	Slot time	20 μsec
aSIFSTime	Short interframe space (SIFS)	10 μsec
aDIFSTime	DCF interframe space (DIFS)	50 μsec
aCWmin	Min contention window size	15 slots
aCWmax	Max contention window size	1023 slots
tPLCPPreamble	PLCP preamble duration	16 μsec
tPLCP_SIG	PLCP SIGNAL field duration	4 μsec
tSymbol	OFDM symbol duration	4 μsec

6.2 Scenario 1: Three Wi-Fi Nodes

Figure 9 depicts Scenario 1. In this simple scenario, we infer the origin time series $\mathbf{f}_A$ sent by source node A, using $\mathbf{x}_{A|B}$ measured at receiver node B. Node C, another source, contends with node A by transmitting its own flow $\mathbf{f}_C$. We carry out cross-validation with all 10 flow datasets by setting $\mathbf{f}_A = \mathbf{f}_i\ \forall i \in \{1, \ldots, 10\}$, flow by flow at once. When $\mathbf{f}_A = \mathbf{f}_i$, we randomly set $\mathbf{f}_C = \mathbf{f}_j\ \forall j \neq i$. Node C can change its flow pattern from $\mathbf{f}_j$ to $\mathbf{f}_k$, while node A still running $\mathbf{f}_i$, but $\mathbf{f}_k$ is chosen such that $k \neq i$.

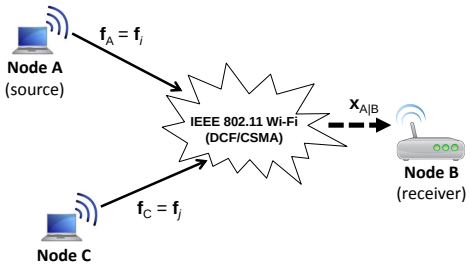

Figure 9: Scenario 1

Wi-Fi setup. We have implemented a custom discrete-event simulator in MATLAB, assuming the IEEE 802.11g our baseline Wi-Fi system. At its core, our CSMA implementation is based on an open-source wireless simulator [2]. The backoff mechanism works as follows. The contention window `CW` is initialized to *aCWmin*. In case of timeout, CSMA doubles `CW`, otherwise waits until the channel becomes idle with an additional DCF interframe space (DIFS) duration. CSMA chooses a uniformly random wait time between [1, `CW`]. `CW` can grow up to *aCWmax* of 1,023 slots. `CW` is decremented only when the media is sensed idle. RTS and CTS are disabled. The Wi-Fi configuration is summarized in Table 2.

Inference schemes. We have implemented all of the inference schemes in MATLAB. We consider ARMAX-

Table 3: Errors to estimate original mean burst size and mean gap length with 5-sec observation window (error percentages in parenthesis are for Scenario 2)

Scheme	Origin burst size estimation error	Origin gap length estimation error
ARMAX	39.3% (45.9%)	28.1% (36.7%)
Naïve Bayes	31.4% (37.5%)	15.8% (24.6%)
GMMs	23.2% (31.3%)	11.7% (18.1%)
DL (baseline)	18.7% (28.3%)	9.3% (16.2%)
DL (enhanced)	12.2% (22.8%)	6.8% (11.4%)

least squares, Naïve Bayes classifiers, Gaussian mixture models (GMM), and the two deep learning methods we proposed. The baseline deep learning method has 2 layers implemented as described in Section 4. The enhanced deep learning method also has 2 layers, but we additionally have implemented incoherent dictionary training, dense sparse coding, and overlapping max pooling as described in Section 5. We call the size of vector $\mathbf{x}$ *observed length* or *observation window size* and have varied 100, 200, 300, and 500.

Training. As mentioned in §6.1.1, the training dataset for each flow has 1,000 instances. For flow i, we transmit training examples $\{\mathbf{f}_{i,k}\}_{k=1}^{1000}$ serially. (Note the notation $\mathbf{f}_{i,k}$ for kth instance of flow i.) We consider 1-vs-all linear SVM classifiers for all inference schemes.

We train ARMAX with n previously *transmitted* origin flow patterns and m previous inputs—*i.e.*, for flow i, we feed $\{\mathbf{f}_{i,k-1},\ldots,\mathbf{f}_{i,k-n}\}$ and $\{\mathbf{x}_{i,k-1},\ldots,\mathbf{x}_{i,k-m}\}$—at time k. After trying out various configurations, we choose $n=2$ and $m=3$. The least squares directly estimate $\hat{\mathbf{f}}$ given $\mathbf{x}$. SVMs for ARMAX are trained with $\hat{\mathbf{f}}$s.

For Naïve Bayes, we extract the statistic $\mathbf{y} = \left[\hat{\mu}_{\text{run\_size}}\ \hat{\mu}_{\text{gap\_length}}\right]$ from $\mathbf{x}$ by averaging run burst sizes and gap lengths and use $\mathbf{y}$ as a feature with label l_i for flow i to build empirical distributions $p(\mathbf{y}|\mathbf{x},l_i)$. Note training Naïve Bayes yields classifiers, so we do not need to train any SVMs for Naïve Bayes.

We use K = 10 GMMs. Like Bayes, we use the same static $\mathbf{y}$ from $\mathbf{x}$ as a feature to train GMMs via the EM algorithm. However, unlike Bayes, GMMs do not yield classifiers. We train SVMs with the membership probabilities from the trained K Gaussians evaluated on $\mathbf{y}$ given l_i.

The proposed deep learning methods produce a max-pooled sparse representation $\mathbf{z}^{(2)}$ at the top of layer 2 for given $\mathbf{x}$. We use labeled $\mathbf{z}^{(2)}$'s to train SVMs.

Results. Figure 10 presents classification recall and false alarm rate of each inference scheme for Scenario 1. Overall, deep learning (DL) yields consistently higher recall at lower false alarm. When using a small observation window length to sample $\mathbf{x}$, the recall gap between the enhanced and baseline deep learning methods is noticeably larger than the case of using a large observation window. This is because dense sparse coding and overlapping max pooling in the enhanced deep learning scheme substantially reduce the probability of missing a feature. A possible explanation for poor ARMAX performance could be that CSMA introduces significant non-linear distortions. GMMs are on par with our baseline deep learning. If optimized to their limits, GMMs seem to be a reasonable alternative to deep learning for classification. This is not surprising since GMMs are really a form of K-SVD. (Note also that our effort to fine-tune GMMs was only fair as our focus in this paper was on deep learning.)

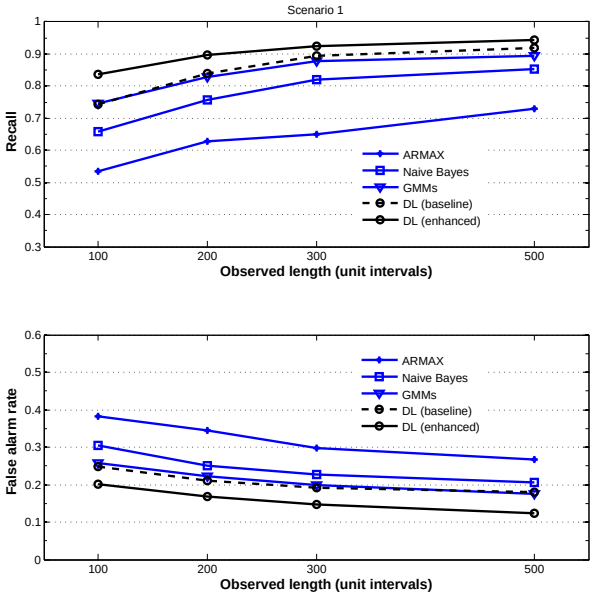

Figure 10: Classification recall and false alarm rate for Scenario 1

After classification, we predict the original mean burst size and gap length of a flow based on lookup of the pre-computed values from the training dataset (see explanation in §6.1.2). The estimation errors in Table 3 are the best case, obtained with the maximum observation window size of 5 seconds that we have tried.

6.3 Scenario 2: Wi-Fi Nodes in OPNET

Figure 11 illustrates Scenario 2 featuring seven Wi-Fi nodes simulated in OPNET. This scenario is important for several reasons. First, we can configure more realistic application profiles for simulated nodes. We can also scale the simulation. Lastly, we can validate our schemes with OPNET's built-in Wi-Fi protocols, particularly the IEEE 802.11g, which should be more complete than our MATLAB simulator. Scenario 2 preserves nodes A, B, C and their activities the same as Scenario 1. There are five additional Wi-Fi nodes that communicate in typical Internet styles as summarized in Table 4.

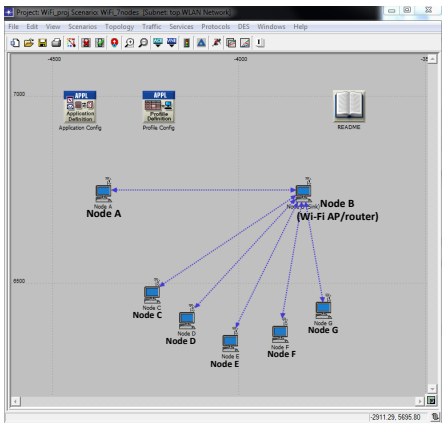

Figure 11: Scenario 2

Table 4: Configuration summary of Scenario 2

Node	Role	Main networking activity
A	Flow source	Transmits $\mathbf{f}_i$
B	Receiver	Intercepts flows as Wi-Fi router/AP
C	Flow source	Transmits $\mathbf{f}_j \ \forall \ j \neq i$
D	Flow source	Multimedia streaming over RTP/UDP/IP
E	Flow dest.	HTTP with page size $\sim$ U[10,400] B
F	Flow dest.	ftp file transfer with size 50000 B
G	Flow dest.	DB access with inter-arrival $\sim$ Exp(3) sec

We test the same inference schemes and keep the training methodology of Scenario 1. In Figure 12, we plot classification recall and false alarm rate of each scheme evaluated under Scenario 2. With more nodes and increased traffic, the overall classification performance is worse. Again, we can clearly see the benefits of dense sparse coding and overlapping pooling for enhanced deep learning that consistently outperforms the other schemes over various observation window sizes. For a small observation window in particular, recall for enhanced deep learning is substantially higher while achieving the lowest false alarm rate. Table 3 shows the estimation errors (in parenthesis) to predict original properties of the flows, the mean burst size and mean gap length, after classification.

7 Conclusion

We have addressed the problem of inferring the original properties of a flow sampled from a received Wi-Fi traffic mix. This inverse problem is challenging because CSMA significantly changes the origin pattern of the flow while scheduling with other flows in competition. Machine learning can provide tools to harness complexity and nonlinearity, but it requires to apply adept domain knowledge or application-specific insights to configure the overall learning pipeline and fine-tune all model parameters to their meticulous detail.

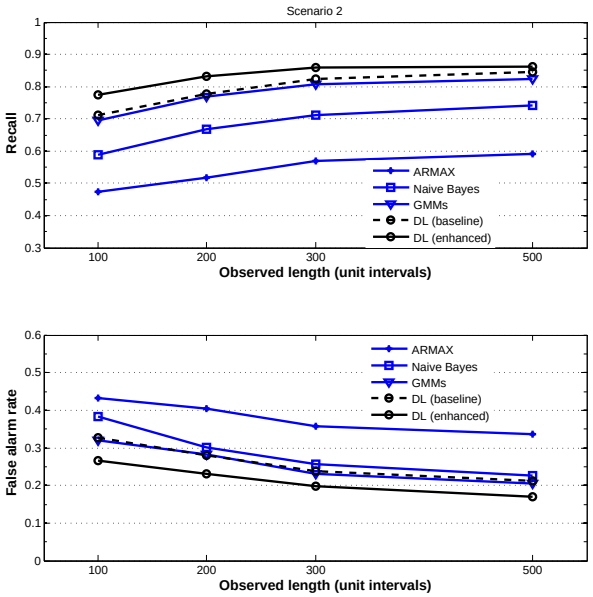

Figure 12: Classification recall and false alarm rate for Scenario 2

Learned from our initial, unsuccessful attempt to straightforwardly integrate sparse coding and dictionary learning into SVM classification, we have set up multiple layers of sparse feature extraction and max pooling that enable deep learning from received flow patterns. Multi-layer sparse coding allows us to learn local, atomic features such as run and gap sizes of a flow accurately at the lowest layer and global features such as periodicity in traffic patterns at higher layers. Max pooling summarizes often too many extracted features while providing translation invariance. We have explained how the proposed approaches incorporate these ideas and validated their superior performance.

In summary, contributions of this paper include a novel formulation of an inverse problem to recover origin flow patterns in Wi-Fi, identification of the key attributes for successful machine learning approaches to solve such inverse problems (*i.e.*, the use of sparse representation of features, multi-layer inference, and pooling), and demonstration of the sound working of these methods in recovering original flow properties. We have chosen not to discuss possible applications of this paper in security and network anomaly detection. We plan to address such applications in our future work.

Acknowledgment

We thank anonymous reviewers of ICAC for their valuable comments in revising the final version of this paper. This material is based on research sponsored in part by Intel Corporation.

References

[1] IEEE 802.11™ Wireless Local Area Networks. `http://www.ieee802.org/11/`.

[2] MATLAB Wireless Network Simulator. `http://wireless-matlab.sourceforge.net`.

[3] Software-defined Networking: The New Norm for Networks. Open Networking Foundation (White Paper), 2012.

[4] AHARON, M., ELAD, M., AND BRUCKSTEIN, A. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Trans. on Sig. Proc. 54*, 11 (2006).

[5] BASU, S., MUKHERJEE, A., AND KLIVANSKY, S. Time Series Models for Internet Traffic. In *INFOCOM* (1996).

[6] BOSER, B. E., GUYON, I. M., AND VAPNIK, V. N. A Training Algorithm for Optimal Margin Classifiers. In *COLT* (1992).

[7] BOUREAU, Y.-L., PONCE, J., AND LECUN, Y. A Theoretical Analysis of Feature Pooling in Visual Recognition. In *ICML* (2010).

[8] CIOFFI, J., AND KAILATH, T. Fast, Recursive-least-squares Transversal Filters for Adaptive Filtering. *IEEE Trans. on Acoustics, Speech and Signal Processing 32*, 2 (1984), 304–337.

[9] CLAFFY, K. C., POLYZOS, G. C., AND BRAUN, H.-W. Application of Sampling Methodologies to Network Traffic Characterization. In *ACM SIGCOMM* (1993).

[10] COATES, A., AND NG, A. Selecting Receptive Fields in Deep Networks. In *NIPS*. 2011.

[11] DEMPSTER, A. P., LAIRD, N. M., AND RUBIN, D. B. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of Royal Statistical Society, Series B 39*, 1 (1977).

[12] DUFFIELD, N., LUND, C., AND THORUP, M. Estimating Flow Distributions from Sampled Flow Statistics. In *ACM SIGCOMM* (2003).

[13] EFRON, B., HASTIE, T., JOHNSTONE, I., AND TIBSHIRANI, R. Least Angle Regression. *Annals of Statistics 32* (2004), 407–499.

[14] HEISELE, B., HO, P., AND POGGIO, T. Face Recognition with Support Vector Machines: Global versus Component-based Approach. In *ICCV* (2001).

[15] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS* (2012), vol. 1, p. 4.

[16] KUNG, H. T., LIN, C.-K., LIN, T.-H., TARSA, S. J., VLAH, D., HAGUE, D., MUCCIO, M., POLAND, B., AND SUTER, B. Location-dependent Runs-and-gaps Model for Predicting TCP Performance over UAV Wireless Channel. In *MILCOM* (2010).

[17] LAWRENCE, S., GILES, C. L., TSOI, A. C., AND BACK, A. D. Face Recognition: A Convolutional Neural Network Approach. *IEEE Trans. on Neural Networks 8*, 1 (1997), 98–113.

[18] LIN, T., LIU, S., AND ZHA, H. Incoherent Dictionary Learning for Sparse Representation. In *ICPR* (2012).

[19] LJUNG, L., Ed. *System Identification (2nd ed.): Theory for the User*. Prentice Hall, 1999.

[20] OLSHAUSEN, B., AND FIELD, D. Emergence of Simple-cell Receptive Field Properties by Learning a Sparse Code for Natural Images. *Nature 381*, 6583 (1996), 607–609.

[21] PATI, Y. C. *et al.* Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition. In *Asilomar Conference on Signals, Systems and Computers* (1993).

[22] RAINA, R., BATTLE, A., LEE, H., PACKER, B., AND NG, A. Y. Self-taught Learning: Transfer Learning from Unlabeled Data. In *ICML* (2007).

[23] RAMIREZ, I., SPRECHMANN, P., AND SAPIRO, G. Classification and Clustering via Dictionary Learning with Structured Incoherence and Shared Features. In *IEEE CVPR* (2010).

[24] RIESENHUBER, M., AND POGGIO, T. Hierarchical Models of Object Recognition in Cortex. *Nature 2*, 11 (1999), 1019–1025.

[25] SNYMAN, J. A. *An Introduction to Basic Optimization Theory: Classical and New Gradient-based Algorithms*. Springer, 2005.

[26] TIBSHIRANI, R. Regression Shrinkage and Selection via the Lasso. *Journal of Royal Statistical Society, Series B 58* (1994).

[27] WRIGHT, J., YANG, A., GANESH, A., SASTRY, S., AND M., Y. Robust Face Recognition via Sparse Representation. *IEEE Trans. on Pattern Analysis and Machine Intelligence 31*, 2 (2009), 210–227.

[28] ZHANG, Q., AND LI, B. Discriminative K-SVD for Dictionary Learning in Face Recognition. In *IEEE CVPR* (2010).