An Open-World Time-Series Sensing Framework for Embedded Edge Devices

ABDULRAHMAN BUKHARI, SEYEDMEHDI HOSSEINIMOTLAGH AND HYOSEUNG KIM
UNIVERSITY OF CALIFORNIA, RIVERSIDE
{ABUKH001, SHOSS007, HYOSEUNG}@UCR.EDU
Motivational Example

Current sensing frameworks lack:

Incremental learning

Classification and learning locally
What is Incremental Learning
Supervised vs Open-world

**Incremental learning**: continuously learning new classes from a stream of data

**COST**: the classifier will forget old classes ⇒ **Catastrophic Forgetting**

This is a **supervised** approach if the new classes are **labelled**

Unlabeled data from unknown classes ⇒ **Open-world problem**

Recognize unknown samples and cluster them into new classes

Classification and Clustering the Unknowns

Classical classifier cannot recognize unknown samples

We can do that:
• Adding a threshold to the classifier output
• Use an open-world classifier
  • OpenMax¹
  • Extreme Value Machine (EVM)²

To Cluster unknown samples
• Unsupervised clustering algorithms ➔ FINCH³ Algorithm

Time-Series Sensing Data

Synthetic Sensors

- A single sensor board can capture multiple environmental facets
- Can be deployed into different environments to recognize different sets of events

Limitations ➔ requires access to a server for training and classification

DeepSense

- A unified framework for time-series sensing data
- Achieve high inference performance for both classification and regression problems

Limitations ➔ network architecture changes based on # of sensors

BOTH

- Cannot incrementally learn new classes for a data-stream

**Incremental Learning**

**(Supervised)** Incremental Classifier and Representation Learning\(^1\)
- Sets of exemplars to represent previously learned classes
- The model is updated with \textit{(new samples + exemplars sets)}
- Fixed-Representation class incremental learning (FRCI)

**(Unsupervised)** Open-World Learning without Labels (OWL)\(^2\)
- Uses Extreme Value Machine (EVM) as a classifier
- Cluster unknown samples using Finch Alg.

They only applied the algorithms on computer vision applications
- Have not been extended to time-series sensing data
- Or evaluated on embedded edge devices

---

\(^1\) Rebuffi et al. iCaRL: Incremental Classifier and Representation Learning. (CVPR, 2017)

Contributions

OpenSense: an open-world sensing framework for time-series data for embedded edge devices

• We present a sensing framework that can run different incremental learning algorithms for both supervised and unsupervised time-series sensing data problems

• We propose an efficient DNN architecture called sDNN, which outperforms the state-of-art architecture in both inference performance and resource efficiency for timeseries activity classification

• We demonstrate the implementation of OpenSense on a resource-constrained edge device and its effectiveness in open-world incremental learning of time-series data.
Outline

• Introduction
• OpenSense Framework
  • Sensor board
  • Embedded edge device
• Evaluation
  • Inference and learning performance
  • Latency and energy consumption performance
• Conclusion
Overview of OpenSense
Sensor Board

- Collect raw data from sensors
- Preprocess them as time-series data
- Transmit the data periodically
- The period managed by the dynamic sensor scheduler
Embedded Edge Device

- Features extraction
- Classification
- Incremental learning
- Sensor period and model updating scheduling
Features Extractor

Further data preprocessing based on the sensor data:

• High-sampling rate ➞ Fast Fourier Transform
• Low-sampling rate ➞ Statistical information

The feature extractor is based on the DNN model

Features are extracted by taking the output data of the last layer before the output layer

We need a light-weight DNN model that extract reliable features at low computational cost
Our Proposed DNN

We call it a simple DNN (sDNN)
- Convolutional Layers \(\rightarrow\) spatial relationship
- LSTM Layers \(\rightarrow\) temporal relationship
- \(f\) is the filter size
- An output layer

The size of input data \(X\) is \(n \times m\)
- \(n\) is the length of preprocessed time-series data
- These data are stacked \(m\) times to make a tensor \(X\)

Input example:
- 1-second sensor data is divided into \((m=10)\) slices
- Each slice is 100ms
- Preprocessing a slice produce an \(n\)-length vector

Feature extractor \(\rightarrow\) Last LSTM layer
Requirements for an open-world classifier:

1. Accurately classify samples from known classes
2. Recognize and reject unknown samples

For supervised approach ➔ a classical classifier

The rejected samples will be collected in a queue for incremental learning
Incremental Learning Algorithm

A true incremental learner must meet 3 criteria:

1. Can be trained from a stream of data with new classes
2. The inference performance must stay competitive
3. Updating the model must meet the resource requirements of the system

We evaluated three algorithms on our framework:

• The naïve approach (NA)
• Fixed-Representation class incremental learning¹ (FRCI)
• OpenSense based on EVM²

For unsupervised learning we use Finch algorithm to cluster the unknown samples

Sensor Dynamic Scheduler

**Goal:** change the sensor data transmission period to:

- Reduce the energy consumption on the sensor board
- Increase the idle time \( \Rightarrow \) free time for other tasks (e.g., learning)

**Propose a Class-based Sensor Dynamic Scheduler:**
The sensor data update period \( T_{sp} \) to detect an event \( C \) must meet the following condition:

\[
n \times T_{sp} - T_e \leq CL
\]

- \( n \): \( #T_{sp} \) repeated until a new event occurs
- \( T_e \): the time when the current event \( C \) ends

Classification Latency \( (CL) \) constraint: the maximum time for the current class to change to a different class while the sensor is idling
Algorithm 1

We propose a searching algorithm to find the maximum feasible sensor update period ($T_{sp}$) that does not exceed CL

Such as:

\[ n \times T_{sp} - T_e \leq CL \]

1. Select minimum time among all time intervals for a class C as a base idle period $T_{sp}$

2. The base period is compared with all other time intervals such that the difference after $n$ cycles does not exceed $CL$

3. If does not meet the condition $\Rightarrow$ decrement $T_{sp}$ by 1

---

**Algorithm 1: Sensor Dynamic Scheduler**

**Input**: $T_e$: All time intervals for class C  
$CL$: User allowable classification latency  

**Output**: $T_{sp}$: Sensor idle period for class C

1. $T_e \leftarrow \text{AscendingSort}(T_e)$
2. $T_{sp} \leftarrow \text{FindMinimum}(T_e)$
3. $L1 \leftarrow T_{sp}$
4. $L2 \leftarrow \text{Length}(T_{sp})$
5. $i \leftarrow 0$; $j \leftarrow 0$
6. for $i \leq L1$ do
   7.   for $j \leq L2$ do
   8.     $n \leftarrow \text{Ceil}(T_e[j]/T_{sp})$
   9.     Thresholds $\leftarrow T_e[j] + CL$
   10.    if $n \times T_{sp} \geq \text{Threshold}$ then
   11.       $T_{sp} \leftarrow T_{sp} - 1$
   12.       break
   13.   end
   14.   $j++$
   15. end
   16. $i++$
7. end
18. if $T_{sp} > 1$ then
19.    return $T_{sp}$
20. else
21.    return fail
22. end
Model and Classifier Updater

The model will be updated when number of samples of a new class meet the minimum requirement

**BUT: resource-constraint edge devices cannot update the model with all new samples**

**We propose a model updating scheduler:**

to partially update the model during the $T_{sp}$ set by the dynamic scheduler

$T_{min}$: the minimum average time to train 1 sample

---

**Algorithm 2: Model Update Scheduler**

- **Input:**
  - $T_{sp}$: The sensor period for a given class
  - $N_u$: # of samples of the new discovered class
  - $S_u$: Samples from the new discovered class

- **Output:**
  - $N_{old}$: Updated number of samples

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$N_{old} \leftarrow 0$</td>
</tr>
<tr>
<td>2</td>
<td><strong>while</strong> $N_u \neq 0$ <strong>do</strong></td>
</tr>
<tr>
<td>3</td>
<td><strong>if</strong> $T_{sp} \geq T_{min}$ <strong>then</strong></td>
</tr>
<tr>
<td>4</td>
<td>$N_{ST} \leftarrow ComputeSamplesToTrain(T_{sp})$</td>
</tr>
<tr>
<td>5</td>
<td>UpdateTheModel($S_u[N_{old} : N_{ST}]$)</td>
</tr>
<tr>
<td>6</td>
<td><strong>if</strong> $N_u \geq N_{ST}$ <strong>then</strong></td>
</tr>
<tr>
<td>7</td>
<td>$N_{old} \leftarrow N_{ST}$</td>
</tr>
<tr>
<td>8</td>
<td><strong>else</strong></td>
</tr>
<tr>
<td>9</td>
<td><strong>return</strong> success</td>
</tr>
<tr>
<td>10</td>
<td><strong>end</strong></td>
</tr>
<tr>
<td>11</td>
<td><strong>else</strong></td>
</tr>
<tr>
<td>12</td>
<td><strong>return</strong> fail /* wait for next $T_{sp}$ */</td>
</tr>
<tr>
<td>13</td>
<td><strong>end</strong></td>
</tr>
<tr>
<td>14</td>
<td><strong>end</strong></td>
</tr>
</tbody>
</table>
Experiment Sets

Classification and learning performance

1. Compare DeepSense vs. Proposed sDNN vs. Proposed sDDN + EVM
2. Evaluate incremental learning algorithms in a supervised setting
3. Evaluate open-world learning algorithms in an unsupervised setting

Latency and energy consumption performance

4. Compare the execution time for different tasks from experiment #3
5. Run the open-world learning based on OWL-EVM on an embedded device and compare the execution time of different tasks
6. Evaluate the latency performance of the sensor dynamic scheduler
7. Evaluate the energy consumption of the sensor dynamic scheduler
8. The Model Updater Scheduler Performance
Evaluation

Evaluation platforms:

• Intel i7 with a dedicated NVIDIA GeForce GTX 1060 GPU [experiments: 1-4]
• Raspberry Pi 4 Model B with 2GB memory as an edge device [experiment: 5]
• TI CC2640R2 LAUNCHXL Board as a sensor board [experiment: 6-8]

Datasets:

• HHAR\textsuperscript{1}: the Heterogeneous Human activity recognition (~120k samples)
  [Biking, Sitting, Standing, 'Walking, Stair Up and Stair down]
• PAMAP2\textsuperscript{2}: Physical Activity Monitoring Data Set (~27k samples)
  [lying, sitting, standing, walking, running, cycling, Nordic walking, watching TV, computer work, car driving, ascending stairs, descending stairs, vacuum cleaning, ironing, folding laundry, house cleaning, playing soccer, rope jumping]

DeepSense vs. sDNN vs. sDNN + EVM

**Objective:** compare the inference performance

- **HHAR** → 6 classes
- **PAMAP2** → 18 classes

**Metrics:**
- Classification Accuracy
- F1-Macro score

> The testing set is the same on all variants

Reason: DeepSense overfits the training dataset due to unnecessarily complex model
DeepSense vs. sDNN vs. sDNN + EVM

**Objective:** compare the training efficiency of each architecture

<table>
<thead>
<tr>
<th>Dataset</th>
<th>HHAR</th>
<th>PAMAP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN Model</td>
<td>DeepSense</td>
<td>sDNN</td>
</tr>
<tr>
<td>#epochs to converge</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>avg. execution time/epoch</td>
<td>37s</td>
<td>16s</td>
</tr>
<tr>
<td>speed-up/epoch</td>
<td>2.3x</td>
<td></td>
</tr>
<tr>
<td>total training time</td>
<td>61m 40s</td>
<td>2m 31s</td>
</tr>
</tbody>
</table>
Supervised Incremental Learning

**Objective:** evaluate incremental learning algorithms:

1. The naïve approach (NA)
2. Fixed-Representation class incremental learning (FRCI)
3. OpenSense (Ours)

Initial training set

- **HHAR** → 2 classes  **PAMAP2** → 3 classes
- # new classes in each data-stream
  - **HHAR** → 2 classes  **PAMAP2** → 3 classes

**Metrics:**
- Classification Accuracy

The testing set add classes at each increment
**Objective:** evaluate the same incremental learning algorithms from the previous experiment in unsupervised setting

Initial training set

- **PAMAP2** $\rightarrow$ 9 classes
- #unknown classes in each data-stream
- **PAMAP2** $\rightarrow$ 3 classes

**Metrics: Open-World Metric**

$$OWM = \frac{N_{KK} \cdot Acc(X_{KK}) + N_{UU} \cdot B3(X_{UU})}{N_{KK} + N_{KU} + N_{UK} + N_{UU}}$$

**The testing set add classes at each increment**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NA</th>
<th>FRCI</th>
<th>OpenSense</th>
</tr>
</thead>
<tbody>
<tr>
<td>#discovered new classes</td>
<td>3/9 classes</td>
<td>5/9 classes</td>
<td>9/9 classes</td>
</tr>
</tbody>
</table>


Evaluation - Classification and learning performance
## Execution Time of Open-World Incremental Learning

**Objective:** Compute the average execution time of different tasks in the framework from the previous experiment

*All algorithms ran on the same training and testing datasets*

<table>
<thead>
<tr>
<th>Task</th>
<th>Inference</th>
<th>Incremental Learning</th>
<th>Total Session Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Feature Extraction</td>
<td>Classification</td>
<td>Queuing</td>
</tr>
<tr>
<td>NA</td>
<td>0.5s</td>
<td>12ms</td>
<td>16µs</td>
</tr>
<tr>
<td>FRCI</td>
<td>0.47s</td>
<td>35ms</td>
<td>26µs</td>
</tr>
<tr>
<td>OpenSense</td>
<td>0.49s</td>
<td>31ms</td>
<td>18µs</td>
</tr>
</tbody>
</table>
Sensor Dynamic Scheduler Latency Performance

- We ran the sensor board for 1000s and capture each event for a random duration
- Assign $T_{sp}$ for each class based on the history of each event
- Used different CL for each class

![Graph showing cumulative latency for different $T_{sp}$]

Cumulative Classification Latency (s)

- Minimum Time Interval Based Period
- Period Based on Alg.1
Conclusion

Proposed OpenSense Framework

• We evaluated different incremental learning algorithms on our framework

• OpenSense can successfully run on the resource-constraint edge device

• sDNN outperforms DeepSense on different datasets

• The proposed sensor dynamic scheduler and model updater scheduler make the framework efficiently runnable on resource-constraint edge devices

Future work

• Extend OpenSense to consider other resources on edge devices, e.g., accelerators
Thank you

Q & A
Algorithm 1

We propose a searching algorithm to find the maximum feasible sensor update period ($T_{sp}$) that does not exceed CL such as:

\[ n \times T_{sp} - T_e \leq CL \]

1. Select minimum time interval among all time intervals for a class C as a base idle period $T_{sp}$
2. The base period is compared with all other time intervals such that the difference after $n$ cycles does not exceed $CL$
3. If does not meet the condition $\Rightarrow$ decrement $T_{sp}$ by 1

In the worst case where no feasible $T_{sp}$ is found, the user may decide to set $CL$ to the minimum value of one, which ensures $T_{sp}$ to be at least two, i.e., $CL = 1$ and $T_{sp} = 2$
Example: Fixed $T_{sp}$

Example:
Time-intervals history for the following events:

A: [5, 9, 11, 16, 19] seconds
B: [3, 7, 15, 20, 23] seconds
C: [10, 18, 23, 31, 39] seconds

If the following sequence of events occurred and the user set $CL = 2$ seconds

Naive approach 1: Focus on satisfying CL at $T_{sp} = 2$ for all events
#Sensor transmission = 39 times
Total classification latency = 2 sec
Missing CL = 0 times
Example: Minimum Interval $T_{sp}$

Example:
Time-intervals history for the following events:

- A: $[5, 9, 11, 16, 19]$ seconds
- B: $[3, 7, 15, 20, 23]$ seconds
- C: $[10, 18, 23, 31, 39]$ seconds

If the following sequence of events occurred and the user set $CL = 2$ seconds

Naïve approach 2: Focus on maximizing idle time

For each event, $T_{sp} = \text{minimum interval of that event}$

$T_{spA} = 5s$, $T_{spB} = 3s$, $T_{spC} = 10s$

#Sensor transmission = 12 times
Total classification latency = 10 sec
Missing CL = 2 times
Example: $T_{sp}$ Based on Alg. 1

Example:

Time-intervals history for the following events:

A: [5, 9, 11, 16, 19] seconds
B: [3, 7, 15, 20, 23] seconds
C: [10, 18, 23, 31, 39] seconds

If the following sequence of events occurred and the user set $CL = 2$ seconds

Based on the proposed algorithm

$T_{sp}A = 3s$, $T_{sp}B = 3s$, $T_{sp}C = 4s$

#Sensor transmission = 22 times

Total classification latency = 5 sec

Missing CL = 0 times
Objective: Compute the average execution time of different tasks using different batch sizes
Sensor Dynamic Scheduler Energy Consumption Performance

- The number of BLE transmissions is compared to a fixed period of 1 seconds.
  - The transmitted BLE packets using Alg. 1 is approximately 6% of the total number of transmissions made by the fixed period approach.
  - 3% more of polling requests is an acceptable trade-off.
Model Updater Scheduler Performance

- We assume there are 200 samples of an unknown class
- The model updater is triggered when it meets the conditions in Alg.2
- the model updater is triggered 3 times to adapt the 200 samples into the model
- $T_{sp}$ is based on the Minimum Time Interval Based Period