A Lightweight Concept Drift Detection Method for On-Device Learning on Resource-Limited Edge Devices

Takeya Yamada & Hiroki Matsutani (Keio University)
IoT: Applications

• ML application in real fields (e.g., anomaly detection)
  Factory, monitoring, robot, safety, security, surveillance, …
Edge AI: Equipment monitoring

- Monitoring of air-conditioning systems (e.g., fans)
  Using wireless sensor nodes that can train and predict
Edge AI: Equipment monitoring

- Wireless sensor nodes that can train and predict [1]
  Raspberry Pi Pico, sensors, magnet, battery, LoRa module
  On-device learning of neural networks

Edge AI: Classification

Cloud side

Task partitioning

Edge server

Edge side
Edge AI: Classification

Six-level rating for edge intelligence [1]


Cloud side

Edge server

Communication cost is low

Level 6
All on-device

Level 5
All in-edge

Level 4
Cloud-edge co-training

Level 3
On-device inference

Level 2
In-edge co-inference

Level 1
Cloud-edge co-inference

On-device learning
Train and predict at IoT devices

Prediction only
On-device learning: Motivation

• Challenges of edge AI: Addressing the gap between training data and deployed environment

Typical solution
✓ Generalization capability to absorb the gap
On-device learning: Motivation

- Challenges of edge AI: Addressing the gap between training data and deployed environment

Typical edge AI use case
1. Collect train data
2. Train at server
3. Predict at edge
   At different environments

Typical solution
- Generalization capability to absorb the gap
On-device learning: Motivation

• Challenges of edge AI: Addressing the gap between training data and deployed environment at low-cost

Typical solution
✓ Generalization capability to absorb the gap

Our approach [1]
✓ Small neural networks
✓ Train at deployed environment

On-device learning: Two modes

1. Train mode
2. Predict-only mode

Question: How and when is the mode changed?

On-device learning: Trigger to retrain

1. Manual retraining

Field-engineers can train edge AI whenever they want

Triggered by train button

2. Automatic retraining

Automatically trained when concept drift is detected

Triggered by concept drift

A lightweight concept drift detection for automatic retraining

On-device learning: Trigger to retrain

• Concept drift
  Phenomenon where statistical properties of target data change over time

On-device learning: Trigger to retrain

- **Concept drift**
  Phenomenon where statistical properties of target data change over time

Old data distribution before drift does not appear

Old data distribution is gradually replaced with new one

Data distribution is incrementally shifted from old one to new one

Old data distribution reoccurs after the data distribution has been changed

---

On-device learning: Prediction

- Prediction is done by \( K \) autoencoder instances, each of which is specialized to each class.

Input: \( n \)-dimensional data, Output: Loss \( l \) and class \( k \)

![Diagram showing prediction and training modes](image)

(a) Two modes

(b) Prediction with \( K \) instances

(c) Sequential training for instance \( k \)
On-device learning: Prediction

- Prediction is done by $K$ autoencoder instances, each of which is specialized to each class.

Input: $n$-dimensional data, Output: Loss $l$ and class $k$

Instance with the smallest loss value is “the closest” instance or class.
On-device learning: Sequential training

- "The closest instance" is updated with the input data.
- OS-ELM [1] is used as sequential training algorithm.
- Weight parameter $\beta$ is sequentially updated with input data $x$.

On-device learning: Sequential training

• “The closest instance” is updated with the input data

By repeating the sequential training of incoming data, each autoencoder is trained to be specialized to each class.

An input data is detected as anomaly if all the instances detect it as anomaly.
Conception drift detection algorithm

- Train time: Trained centroids sequentially updated

Centroids are sequentially updated every time incoming data is sequentially trained.
Concept drift detection algorithm

- **Train time**: Trained centroids sequentially updated
- **Predict time**: Recent centroids sequentially updated after an anomaly is detected

Centroids are sequentially updated every time prediction is done for incoming data.
Concept drift detection algorithm

- **Train time**: Trained centroids sequentially updated
- **Predict time**: Recent centroids sequentially updated

Drift is detected when moving distances exceed a threshold.

After a certain time-window is passed, the moving distances are evaluated.
Concept drift detection algorithm

- Train time: Trained centroids sequentially updated
- Predict time: Recent centroids sequentially updated

Drift is detected when moving distances exceed a threshold.

Timing chart of concept drift detection and retraining (Steps 1, 2, 3, 4, and 5)
Evaluations: Comparisons

- Proposed detector is compared w/ other approaches

Detect the drifts and trigger retraining of the discriminative model

<table>
<thead>
<tr>
<th>Detector</th>
<th>Discriminative model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed method</td>
<td>OS-ELM</td>
</tr>
<tr>
<td>Baseline</td>
<td>OS-ELM</td>
</tr>
<tr>
<td>Quant Tree [1]</td>
<td>OS-ELM</td>
</tr>
<tr>
<td>SPLL [2]</td>
<td>OS-ELM</td>
</tr>
<tr>
<td>ONLAD [3]</td>
<td>OS-ELM w/ forgetting method</td>
</tr>
</tbody>
</table>

Trainable neural network that has a single hidden layer is used as the discriminative model for anomaly detection

### Evaluations: Comparisons

<table>
<thead>
<tr>
<th>Detector</th>
<th>Discriminative model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed method</td>
<td>OS-ELM</td>
</tr>
<tr>
<td>None</td>
<td>OS-ELM</td>
</tr>
<tr>
<td>Quant Tree</td>
<td>OS-ELM</td>
</tr>
<tr>
<td>SPLL</td>
<td>OS-ELM w/ forgetting method</td>
</tr>
<tr>
<td>No detection (No retraining)</td>
<td>Batch algorithms</td>
</tr>
<tr>
<td>Sequential algorithm</td>
<td>Retrain model whenever new data comes while forgetting old data</td>
</tr>
</tbody>
</table>

---

Evaluations: Dataset

• Train & test samples of NSL-KDD dataset [1] are concatenated at 8333rd sample as a concept drift.

Accuracy drops after the concept drift.

No detection (No retraining)

Evaluations: Accuracy

- Train & test samples of NSL-KDD dataset [1] are concatenated at 8333rd sample as a concept drift.

Proposed, Quant Tree, and SPLL detect the concept drift → Accuracy recovers quickly.

Evaluations: Memory utilization

- Memory utilization for Cooling fan dataset [1]
  Frequency spectrum (1 - 512Hz)

- Our target platform
  Raspberry Pi Pico (264 kB SRAM)
  Accelerometer


Wireless sensor nodes for anomaly detection on vibration patterns
Evaluations: Memory utilization

- Memory utilization for Cooling fan dataset [1]
  Frequency spectrum (1 - 512Hz)

- Our target platform
  Raspberry Pi Pico (264 kB SRAM)

Sequential algorithm can significantly save memory utilization

<table>
<thead>
<tr>
<th></th>
<th>Batch size</th>
<th>Memory utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed method</td>
<td>1 (Sequential)</td>
<td>69 kB</td>
</tr>
<tr>
<td>Quant Tree</td>
<td>235</td>
<td>619 kB</td>
</tr>
<tr>
<td>SPLL</td>
<td>235</td>
<td>1933 kB</td>
</tr>
</tbody>
</table>

Summary

- A lightweight concept drift detection for on-device learning at tiny devices (e.g., Raspberry Pi Pico)

Concept drifts can be detected as well as existing batch-based methods while reducing memory utilization by the sequential algorithm.