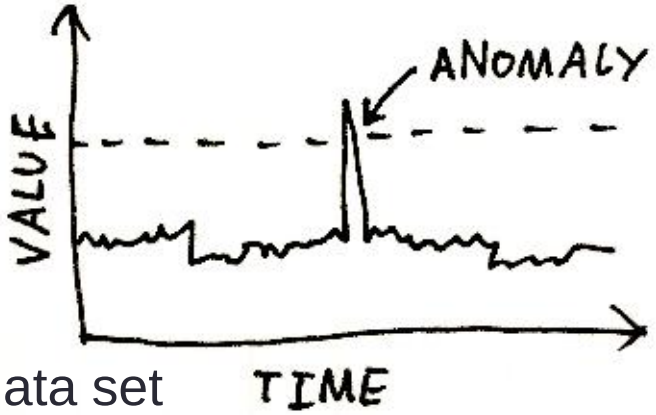
Distributed Anomaly Detection using Autoencoder Neural Networks in WSN for IoT

Tony T. Luo, Institute for Infocomm Research, A*STAR, Singapore - https://tonylt.github.io
Sai G. Nagarajan, Singapore University of Technology and Design

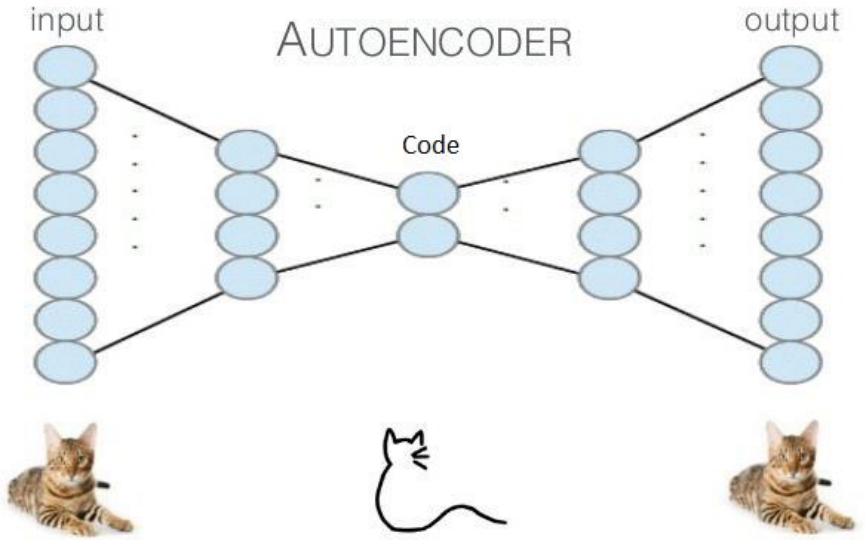
IEEE ICC 2018

Introduction



- Anomalies (a.k.a. outliers):
  - Data that do not conform to the patterns exhibited by the majority of data set
    - e.g. equipment faults, sudden environmental changes, security attacks
- Conventional approach to anomaly detection:
  - Mainly handled by “Backend”
  - **Disadvantages**: inefficient use of resources (bandwidth & energy); delay
- Other prior work:
  - Threshold-based detection with Bayesian assumptions [2]
  - Classification using kNN or SVM [3,6]
  - Distributed detection based on local messaging [4,5]
  - **Disadvantages**: computationally expensive, large communication overhead

Our approach



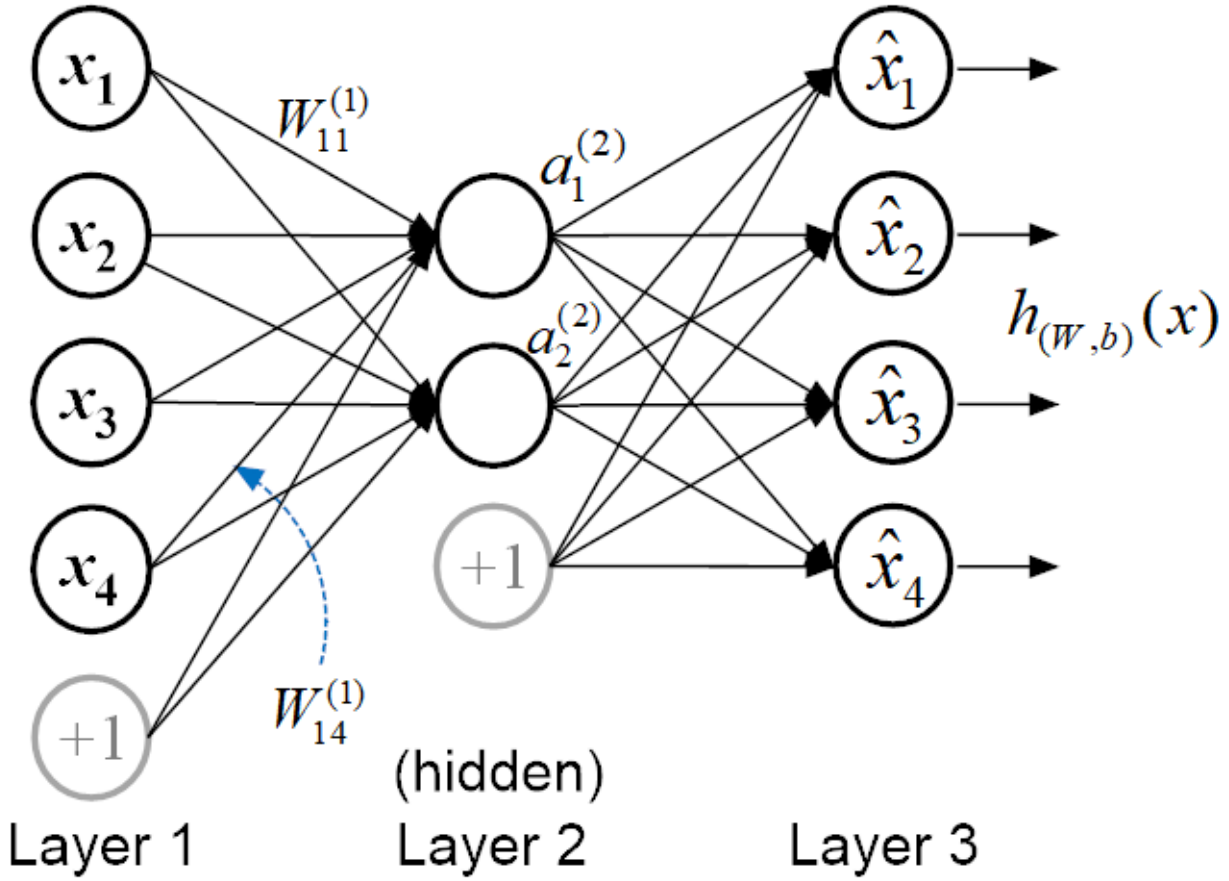
- Objective: push the task to the “**edge**”
- Challenges: sensors are resource-scarce
- Introducing autoencoder neutral networks
  - A deep learning model traditionally used in image recognition and spacecraft telemetry data analysis
  - But DL is generally not suitable for WSN!
  - We build a **three-layer autoencoder** neutral network with only one hidden layer, leveraging the power of autoencoder in reconstructing inputs
  - We design a **two-part algorithm**, residing on sensors and IoT cloud, respectively:
    - Sensors perform **distributed anomaly detection**, without communicating with each other
    - IoT cloud handles the computation-intensive **learning**
    - Only very infrequent communication between sensors and cloud is required

Contributions

1. First introduces autoencoder neutral networks into WSN to solve the problem of anomaly detection
2. Fully distributed
3. Minimal communication and edge computation load
4. Solves the common challenge of lacking anomaly training data

Preliminaries: autoencoder

- A special type of neural networks
  - Objective is to **reconstruct inputs** instead of predicting a target variable
- Structure:
  - **Input layer**: e.g., a time series of sensor readings
  - **Output layer**: a “clone” of the inputs
  - **Hidden layers**: “encode” the essential information of inputs



Preliminaries: autoencoder (cont'd)

- **Activation function**: each represented by a neuron, usually a sigmoid function

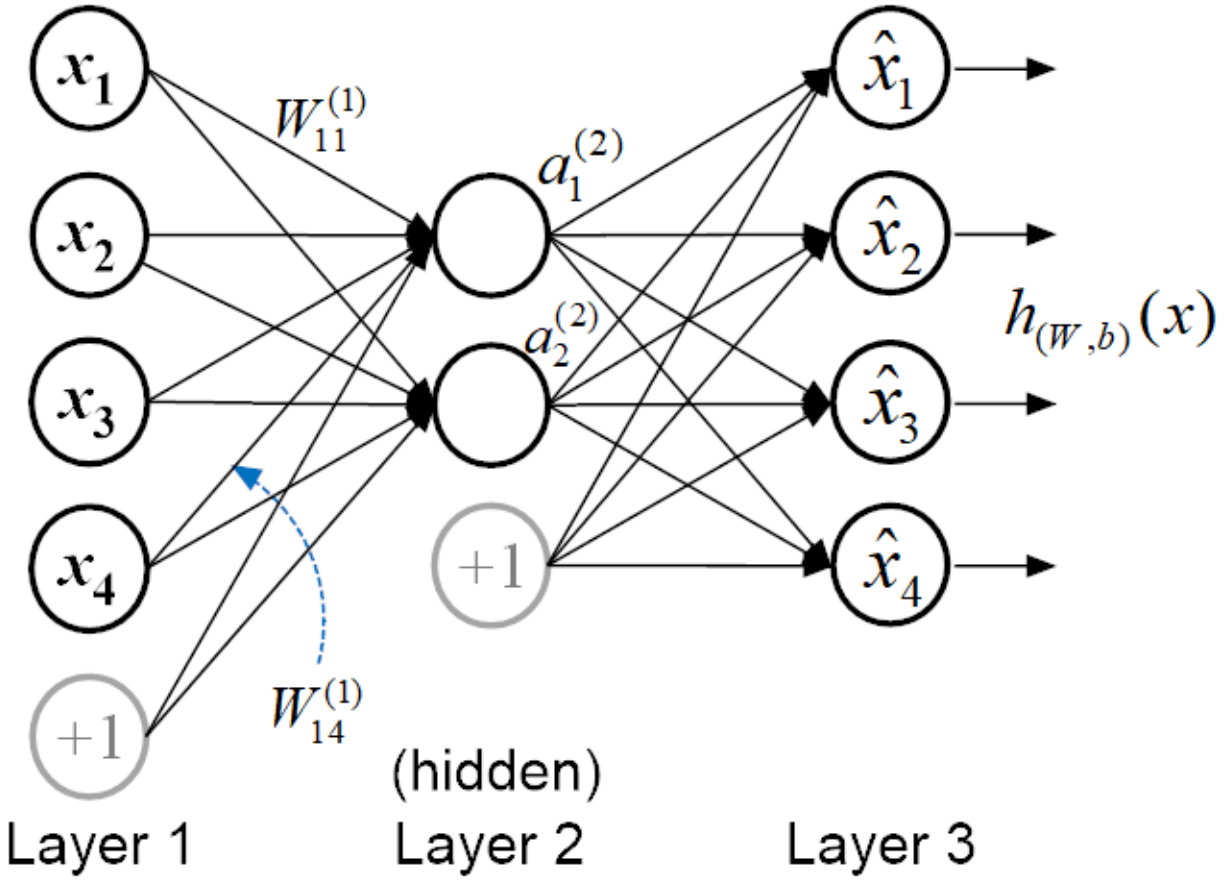
$$f(z) = \frac{1}{1 - e^{-z}}$$

- **Hyperparameters**:
  - ***W***: weights
  - ***b***: bias (the "+1" node)
- **Output** at each neuron:

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l-1)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l-1)})$$

- **Objective**: minimize cost function

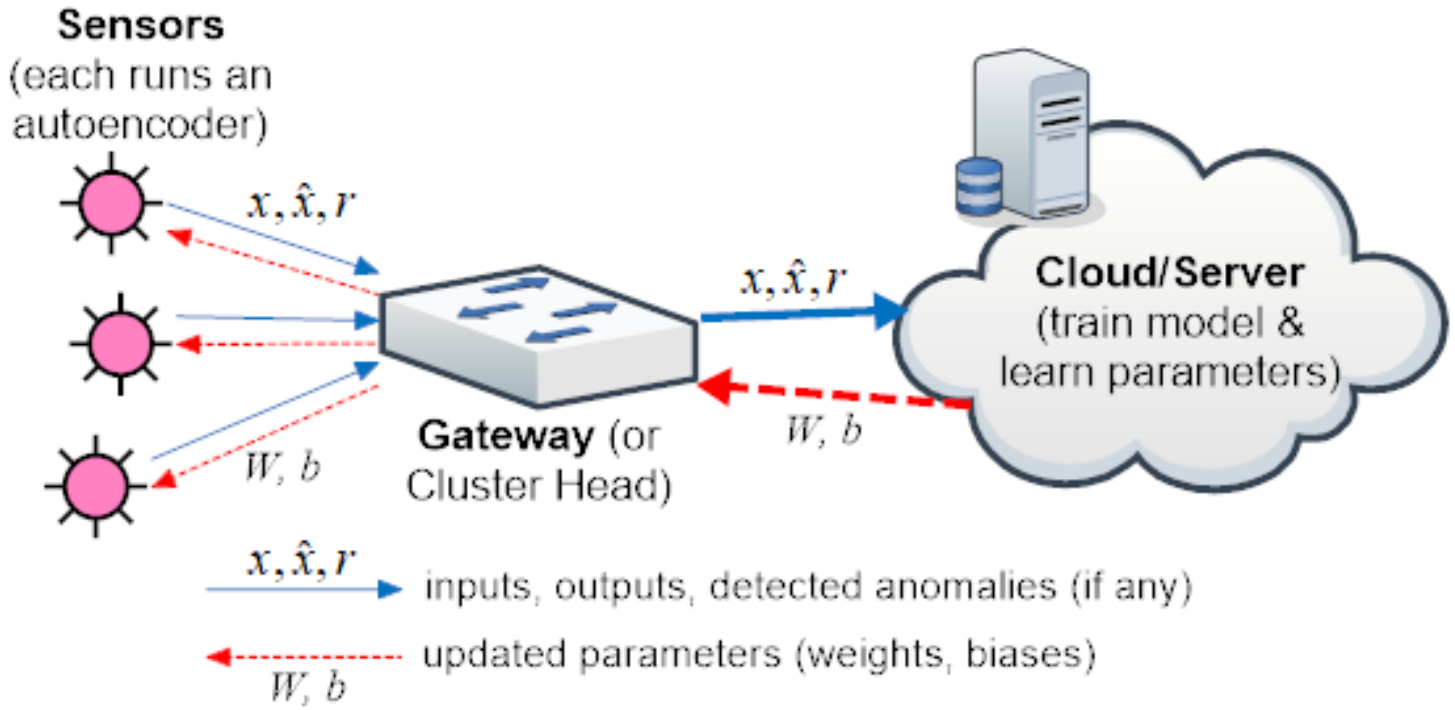
$$J(W,b) := \frac{1}{T}\sum_{i=1}^{T}\left(\frac{1}{2}||h_{W,b}(\mathbf{x}[i]) - \mathbf{x}[i]||^2\right) + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{j=1}^{n^{(l)}}\sum_{i=1}^{n^{(l+1)}}\left(W_{ij}^{(l)}\right)^2$$



i.e., Reconstruction error + Regularization term (to avoid overfitting)

System architecture

- **Sensors**
  - Each runs an autoencoder to detect anomalies
  - Sends inputs and outputs (in fact difference) of autoencoder to IoT cloud in low frequency
- **Cloud**
  - Trains autoencoder model using the data provided by all sensors
  - Sends updated model parameters (W, b) back to all the sensors



Anomaly detection

- Each sensor calculates reconstruction error (**residual**):

$$r_m(s,d) = x_m(s,d) - \hat{x}_m(s,d)$$

- Cloud calculates mean and variance over all sensors:

$$\mu_m = \frac{1}{DS}\sum_{d=1}^{D}\sum_{s=1}^{S} r_m(s,d)$$

$$\sigma_m^2 = \frac{1}{DS}\sum_{d=1}^{D}\sum_{s=1}^{S} (r_m(s,d) - \mu_m)^2$$

D: # of days
S: # of sensors

- Each sensor detects anomaly by calculating

$$\alpha_m(s,d) = \begin{cases} 0, & \text{if } |r_m(s,d) - \mu_m| \leq p\sigma_m \\ 1, & \text{otherwise} \end{cases}$$

  - *p*: assuming residuals are Gaussian, p=2 corresponds to 5% are anomalies and 3 corresponds to 2.5%

Two-part algorithm

- Sensor: **DADA-S**

Algorithm 1: DADA-S: Distributed Anomaly Detection using Autoencoders (Sensor's algorithm)

1 **for** $d \leftarrow 1$ **to** ∞ **do**
2 Obtain sensor readings $\mathbf{x} = \{x_1, x_2, ..., x_M\}$;
3 Feed $\mathbf{x}$ into autoencoder to obtain output $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_M)$;
4 Calculate residual vector $\mathbf{r} = \mathbf{x} - \hat{\mathbf{x}}$;
5 Detect anomaly according to (4) and obtain $\boldsymbol{\alpha}(s, d)$;
6 Send $\mathbf{x}, \hat{\mathbf{x}}, \mathbf{r}(s, d), \boldsymbol{\alpha}(s, d)$ to cloud (via gateway);
7 Update $\mathbf{W}, \mathbf{b}, \boldsymbol{\mu}, \boldsymbol{\sigma}$ received from cloud;
8 **end**

Computational complexity: $O(M^2)$
TPDS'13: $O(2^{M-1})$

- Cloud: **DADA-C**

Algorithm 2: DADA-C: Distributed Anomaly Detection using Autoencoders (Cloud's algorithm)

1 **for** $d \leftarrow 1$ **to** ∞ **do**
2 Receive $\mathbf{x}, \hat{\mathbf{x}}, \mathbf{r}(s, d), \boldsymbol{\alpha}(s, d)$ from all the sensors $s = 1, 2, ..., S$;
3 Store $\mathbf{x}, \hat{\mathbf{x}}$ in the training data set and react to $\boldsymbol{\alpha}(s, d)$ if needed;
4 **if** d *mod* $D_u = 0$ **then**
// D_u denotes training frequency (in no. of days) chosen by cloud
5 Retrain autoencoder with the updated training data set; // data will be pre-shuffled to avoid learning biases toward the latest data
6 Recalculate $\boldsymbol{\mu}, \boldsymbol{\sigma}$ using $\mathbf{r}(s, d), \boldsymbol{\alpha}(s, d)$ according to (3);
7 Send updated $\mathbf{W}, \mathbf{b}, \boldsymbol{\mu}, \boldsymbol{\sigma}$ to all the sensors;
8 **end**
9 **end**

Performance evaluation

- An indoor WSN testbed consisting of 8 sensors that measure temperature and humidity
- Data collected over 4 months (Sep – Dec 2016)
- Synthetic anomalies generated using two common models:
  - **Spike**: $x'(t) = x(t) + v\delta(t)$
  - **Burst**:

$$x'(t) = \begin{cases} x(t) + v, & t_{start} \leq t \leq t_{end} \\ x(t), & \text{otherwise.} \end{cases}$$

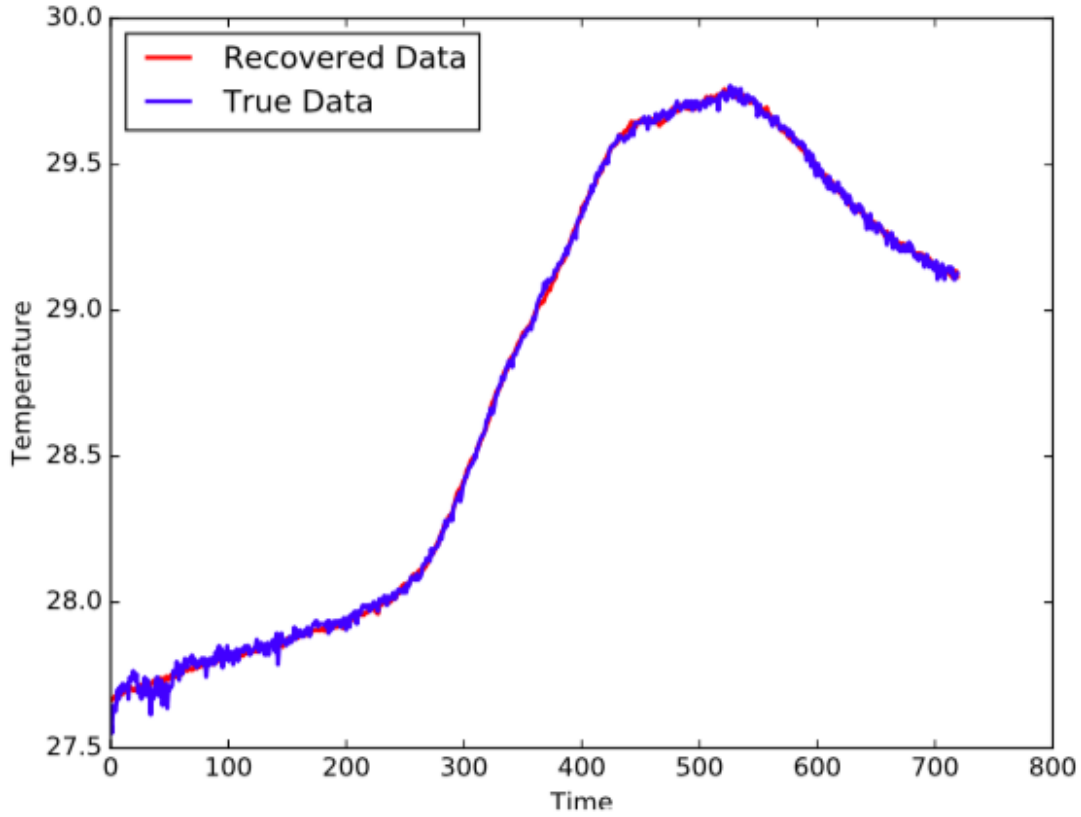
- # of neurons: 720 (I/O layer), 504 (hidden layer; optimized using k-fold cross validation)

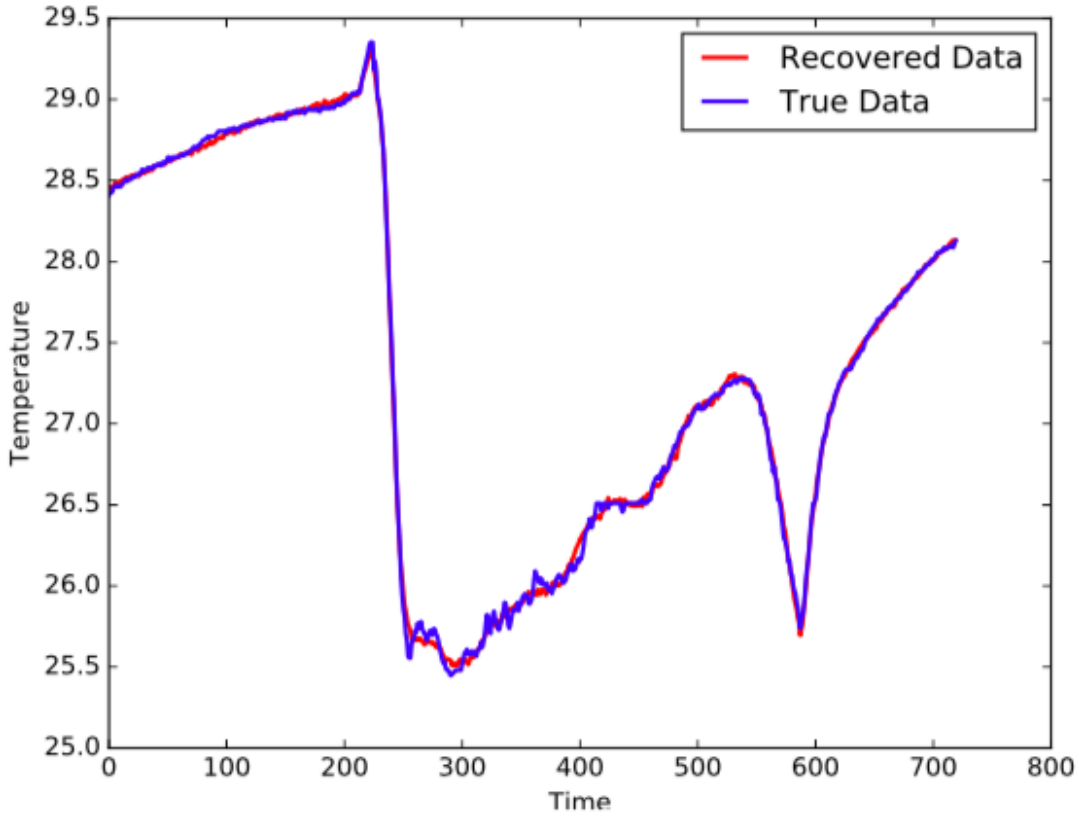
(a) Sensor at a corridor. (b) Sensor at cubicle 1

(c) Sensor at cubicle 2. (d) Sensor beside a window.

Reconstruction performance

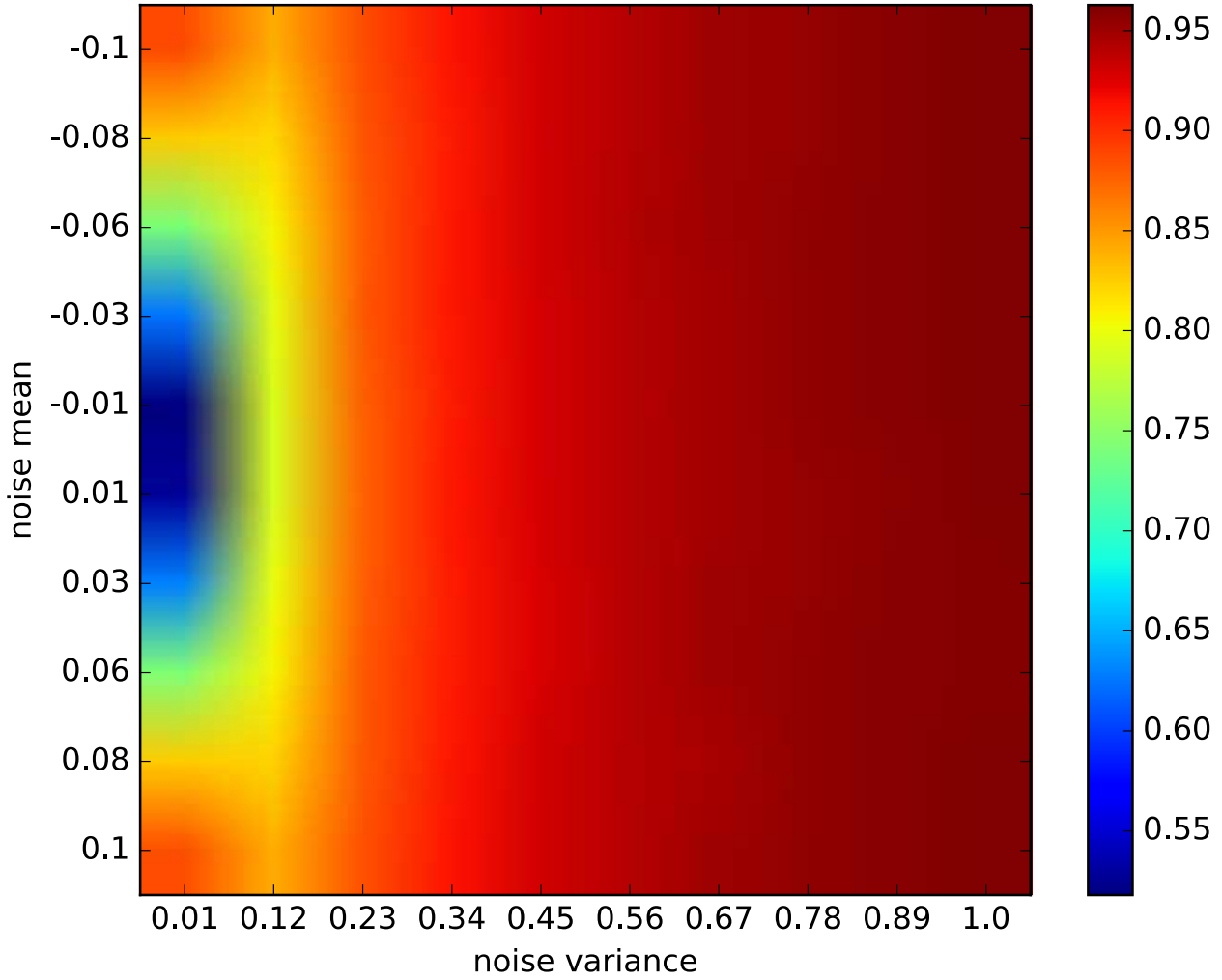
- When no anomaly is present
- Recovered data (output) almost coincides with true data (input) – model is validated





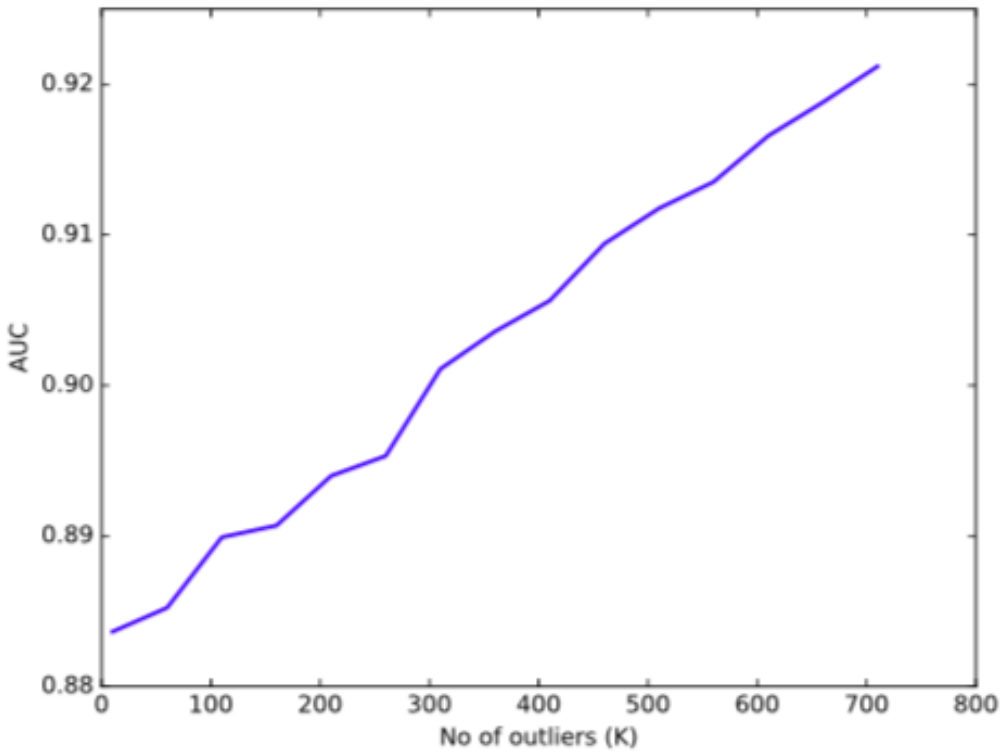
Varying anomaly magnitude

- Varying magnitude according to normal distribution $N(\mu, \sigma^2)$
- Plot AUC w.r.t. both μ and σ^2
- AUC > 0.8 in most cases, indicating a good classifier
- Lower AUC (0.5--0.8) appears when both μ and σ^2 are very small, which are insignificant deviations from the normal

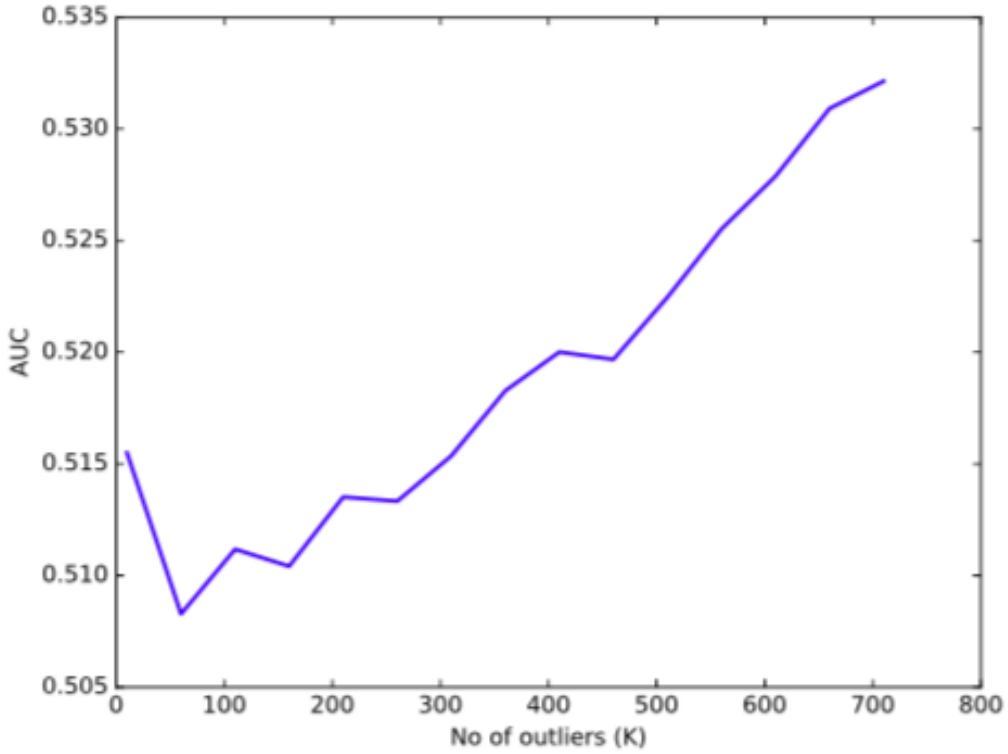


Varying anomaly frequency

- Continues to perform well even when the # of anomalies is large



(a) $\mu_v = 0.1$, $\sigma_v^2 = 0.01$.

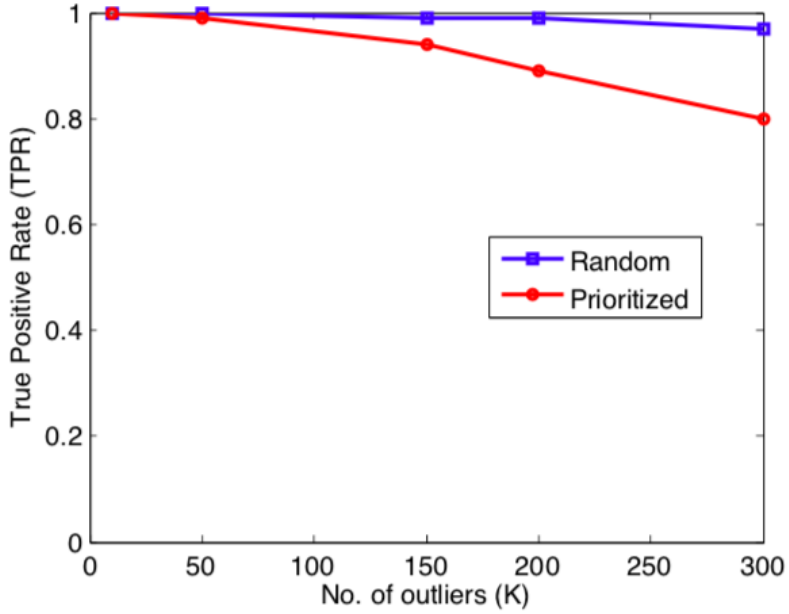


(b) $\mu_v = 0.01$, $\sigma_v^2 = 0.01$.

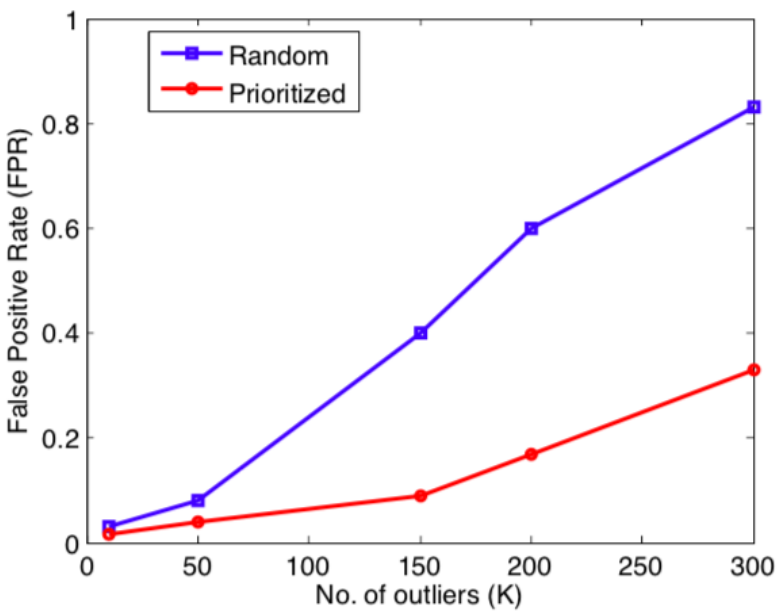
Adaptive to non-stationary environment

- Use two different configurations of training data:
  - **Random**: new observations are randomized with the entire historic data
  - **Prioritized**: most recent 14 days' data mixed with another randomly chosen 14 days' data

- TPR: **Random** performs better, because training data is less affected by changes
- FPR: **Prioritized** performs better, because autoencoder learns more from fresh inputs that contains more changes, thus recognizing some previous anomalies are no longer anomalies



(a) True positive rate (TPR).



(b) False positive rate (FPR).

Conclusion

- First introduces autoencoder neutral networks into WSN to solve the anomaly detection problem
- Fully distributed
- Minimal communication (zero among sensors) and minimal edge computation load (polynomial complexity)
- Solves the common challenge of lacking anomaly training data (by virtue of unsupervised learning)
- High accuracy and low false alarm (characterized by AUC)
- Adaptive to new changes in non-stationary environments

- Connect via my research homepage:
  - https://tonylt.github.io