Water Anomaly Detection Using Federated Machine Learning

Melker Wallén and Mauricio Böckin

***Abstract*—With the rapid increase of Internet of Things-devices (IoT), demand for new machine learning algorithms and models has risen. The focus of this project is implementing a federated learning (FL) algorithm to detect anomalies in measurements made by a water monitoring IoT-sensor. The FL algorithm trains across a collection of distributed IoT-devices, each using the local data acquired from the specific sensor. The local machine learning models are then uploaded to a mutual server and aggregated into a global model. The global model is sent back to the sensors and is used as a template when training starts again locally. In this project, we only have had access to one physical sensor. This has forced us to virtually simulate sensors. The simulation was done by splitting the data gathered by the only existing sensor. To deal with the long, sequential data gathered by the sensor, a long short-term memory (LSTM) network was used. This is a special type of artificial neural network (ANN) capable of learning long-term dependencies. After analyzing the obtained results it became clear that FL has the potential to produce good results, provided that more physical sensors are deployed.**

***Sammanfattning*—I samband med den snabba ökningen av Internet of Things-enheter (IoT) har efterfrågan på nya algoritmer och modeller för maskininlärning ökat. Detta projekt fokuserar på att implementera en federated learning (FL) algoritm för att detektera avvikelser i mätdata från en sensor som övervakar vattenkvaliteten. FL algoritmen tränar en samling decentraliserade IoT-enheter, var och en med hjälp av lokal data från sensorn i fråga. De lokala maskininlärningsmodellerna laddas upp till en gemensam server och sammanställs till en global modell. Den globala modellen skickas sedan tillbaka till sensorerna och används som mall när den lokala träningen börjar igen. I det här projektet hade vi endast tillgång till en fysisk sensor. Vi har därför varit tvungna att simulera sensorer. Detta gjordes genom att dela upp datamängden som samlats in från den fysiska sensorn. För att hantera den långa sekventiella data används ett long short-term memory (LSTM) nätverk. Detta är en speciell typ av artificiellt neuronnät (ANN) som är kapabelt att minnas mönster under en längre tid. Efter att ha analyserat resultaten blev det tydligt att FL har potentialen att producera goda resultat, givet att fler fysiska sensorer implementeras.**

***Index Terms*—Federated learning, neural network, anomaly detection, water monitoring, long short-term memory.**

Supervisors: José Mairton Barros Da Silva Júnior, Carlo Fischione

TRITA number: TRITA-EECS-EX-2021:179

I. Introduction

A. Background

Over the last few years, computing power of Internet of Things (IoT)-devices have increased drastically. Previously, the typical IoT-device was just a connection to pass data from one server to another, and all of the heavy computation was done server-side. Today, data processing is being pushed back to the devices. Performing the computations client-side is called *edge computing* [1].

Fig. 1: Visualization of the FL method [3].

With the rise of edge computing, the restrictions of doing computations server-side have become more apparent. For example, one flaw of server-side computing is that it can not be used without encryption, if the collected data is private or sensitive [2]. Locally, this is not an issue.

While the computations generally move closer to the edge devices, new clever algorithms which are able to utilize decentralized data are needed. One of the most influential areas of research regarding computations done locally in IoT-devices is machine learning. Recent research has generated plenty of techniques which tackle the problem in different ways, with one of the most popular being Federated Learning (FL) [4]. It handles the privacy problem by training a machine learning model locally on the collected data, and then only transmitting the fully trained model to a common server for all clients as shown in Figure 1. The local models are then combined into a global model, which is then downloaded to each client as a template for further training. Notice that no raw data is ever sent to the server.

This paper is a part of the iWater project, with partners such as Stockholm City, Ericsson, Stockholm University, SVOA, Telia and Linköping University. iWater is a freshwater monitoring project, with the goal to develop and install a cloud-based water quality monitoring system, using data from connected sensors. The iWater project aims at making water quality data public for all citizens as well as optimizing water quality testing for both drinking and bathing water [5].

Previously, water quality testing was primarily manual, with the analysis carried out in laboratories. This is not nearly as time- and cost-efficient as edge computing with deployed sensors. In EU, water quality is regulated by the Water Framework Directive [6], but it has recently been shown that many lakes and rivers in Sweden do not live up to the requirements presented [7]. The iWater project currently has one deployed sensor, handled by Ericsson and Stockholm

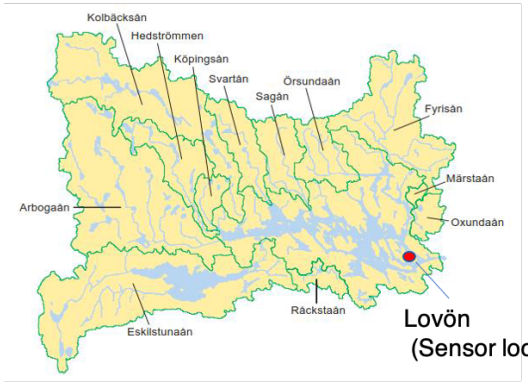

Fig. 2: Location of the water monitoring sensor.

TABLE I: Measurement types and their corresponding labels

Measurement type / unit	Label
Conductivity [μS/cm]	CN
Oxygen saturation [%]	OS
pH-value [1]	PH
Reduction potential [mV]	RX
Salinity [ppt]	SA
Temperature [°C]	TC

University, as well as a centralized machine learning algorithm that is used to detect anomalies in the water. The development of new Artificial Intelligence (AI) methods and the deployment of new sensors is expected to provide groundbreaking results [8].

B. Problem formulation

FL and IoT-sensors now enable us to create a multi-sensor analysis, with the hopes of creating a network of sensors that are communicating with each other. This could then be used to detect different types of water contamination almost instantly, at multiple locations simultaneously.

In this project, we will investigate whether FL can improve the results produced by the already developed centralized machine learning model in order to provide a basis for whether it would be reasonable, or not, to invest in additional physical sensors.

To fully understand the problem at hand, we will begin at the edge, with the sensor. It is manufactured by Libelium and located in the lake Mälaren. The exact location is shown in Figure 2. It measures the values presented in Table I.

The sensor samples and uploads new data to a server every 20 minutes. The FL method is based on training machine learning models locally on multiple clients. Since we only have had access to one sensor, we have been forced to simulate these clients by splitting the data from the one sensor and distributing it to multiple virtually simulated clients. We will go through this process more in depth in Section III.

The objective is to predict the value of the next measurement made by the sensor based on the earlier measurements using FL and the already developed machine learning model. The idea is that this will make it possible to detect anomalies in the fresh water.

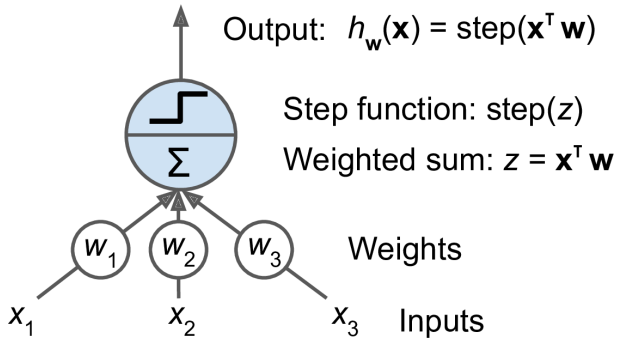

Fig. 3: A perceptron with three inputs and three weights [11].

II. Theory

In this section we briefly present the theory behind Machine Learning and ANNs, and more thoroughly introduce the theory of long short-term Memory (LSTM) networks and FL.

A. Machine Learning and ANN

The definition of a machine learning algorithm is that it is an algorithm able to learn from experience or as formulated by Mitchell [9]:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

In this project the task T is anomaly detection, which translates to finding data that does not fit the rest of the data set, and flagging these as anomalies. The performance measure, P, is the accuracy of these predictions. To measure the accuracy, the data is split into a *training set* and a *test set*. The test set is not used in training, so that we can evaluate how well our model generalizes to data that was not used during training.

ANNs are a class of machine learning algorithms that are inspired by biological neurons. The most basic unit of an ANN is called an *artificial neuron*. An artificial neuron is a function that takes one or more values as input, produces a weighted sum of these values and passes it through a function called an *activation function*, which in turn produces a final output value. Historically, the first artificial neuron was proposed by Rosenblatt [10] and is called a *perceptron*. The inputs and outputs of the perceptron are scalars and each connection is associated with a *weight*, the perceptron described in Figure 3 has three inputs and three weights.

Let $\mathbf{x} \in \mathrm{R}^n$ be the input and $\mathbf{w} \in \mathrm{R}^n$ be the weights. The perceptron computes a weighted sum, z, of its inputs according to

$$z = \mathbf{x}^\mathbf{T}\mathbf{w} = \sum_{i=1}^{n} w_i x_i, \tag{1}$$

and applies an activation function. The produced output can be expressed as

$$h_\mathbf{w}(\mathbf{x}) = step(\mathbf{x}^\mathbf{T}\mathbf{w}). \tag{2}$$

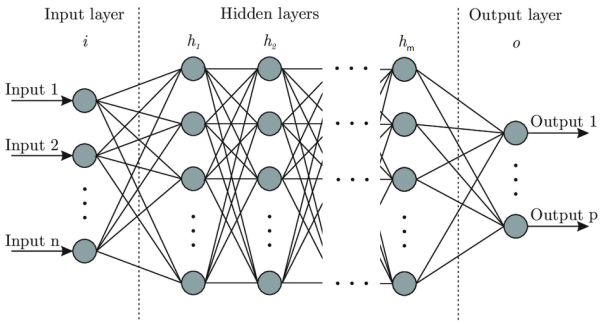

Fig. 4: ANN

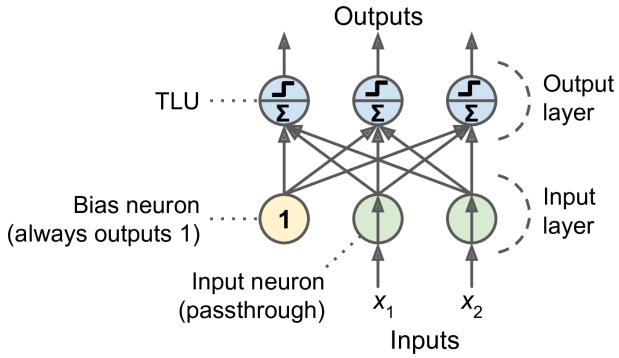

Fig. 5: Layer architecture [11]

The perceptron uses the *Heaviside step function*

$$step(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

as its activation function, which means that it is only able to approximate functions for linearly separable datasets. Modern ANNs use non-linear activation functions in order to be able to approximate functions for more complicated tasks [11]. There are many non-linear activation functions, two of the most frequently used being the Rectified Linear Unit function (ReLu) and the Sigmoid function, see Equation 4 and 5 respectively:

$$ReLu(x) = max(0, x) \tag{4}$$

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

ANNs are built up out of many layers of stacked artificial neurons as shown in Figure 4. It is also common to introduce *bias neurons*, which are neurons that always output a constant. Computing the output of a layer will now look like:

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = f(\mathbf{XW} + \mathbf{b}). \tag{6}$$

In Equation 6, f(x) is the activation function. **X** represents the matrix of inputs and has one row per input neuron. **W** denotes the matrix of weights, except for the bias neuron. It has one row per input neuron and one column per neuron in the layer. **b** is the bias vector which contains all connections between the bias neuron and the regular neurons, it has one bias term per neuron. There is always one bias term per neuron as shown in Figure 5 [11]. The name *bias* originates from the fact that the output is biased towards having the value b if there does not exist an input [12].

The objective of a neural network is to approximate a function ϕ. In short, a *feed forward neural network* defines a mapping $y = \phi(\mathbf{x}; \theta, b)$ and learns the values for the weights θ and biases b that minimizes the error of the approximation. The error is measured using a *loss function*, which is a function that computes the error between the output of the neural network and the given target value. Feed forward neural network are networks where the connections between the neurons do not form cycles, we will discuss other types of neural networks in the following subsection.

One example of a loss function, and the one that is used in the centralized machine learning model, is the cross-entropy loss function. It is especially useful when classifying labeled data. The output is given as a probability between 0 and 1. As the actual predicted probability diverges from the intended label, the loss function increases. In other words, it penalizes predictions which are both wrong and confident the hardest. More rigorously, the cross-entropy loss function L is defined as:

$$L = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}). \tag{7}$$

Data binning is a commonly used data-quantization technique where the original data values that fall into a given interval, a *bin*, are replaced by a value representative of that interval. We will elaborate on how this technique works and how it connects to our project in an upcoming section. In the equation above, M is the number of bins, y is an indicator, which is either 0 or 1 depending on if the bin label c is the right classification for the observed data o and p is the predicted probability.

The learning is done using the *Backpropagation algorithm* and *Stochastic gradient descent* (SGD) [12]. The gradient of the loss function is calculated using the Backpropagation algorithm. It does this by computing the partial derivatives of the loss function with respect to each weight and bias. SGD is a commonly used optimizer that uses the gradient of the loss function in order to determine how to update the weights and biases such that the error between the output and the target value is minimized. The so-called learning rate parameter η controls how much the weights and biases of the ANN changes in response to the loss function after an update. Choosing a value that is too large may result in instability or learning a set of sub-optimal weights and biases too quickly, while setting it too low might result in the training taking a very long time.

SGD is not the only commonly used optimization method, the *Adam optimizer* is another example of a frequently used optimizer. In short, it uses momentum and an adaptive learning rate to converge faster, it is explained in more detail by Kingma and Ba in the article *Adam: A method for stochastic optimization* [13].

To reduce the risk of over-fitting a neural network, L1-regularization is a commonly used method. It works as an

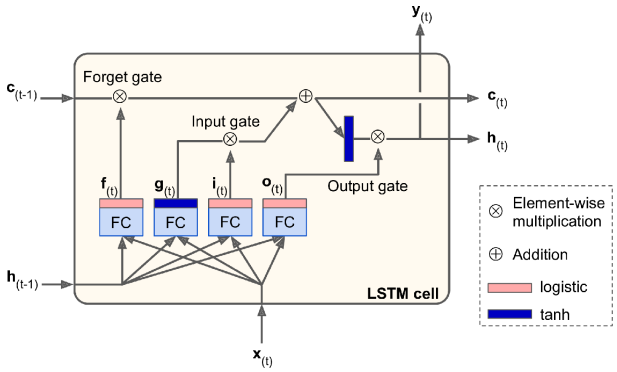

Fig. 6: Architecture of a LSTM cell [11].

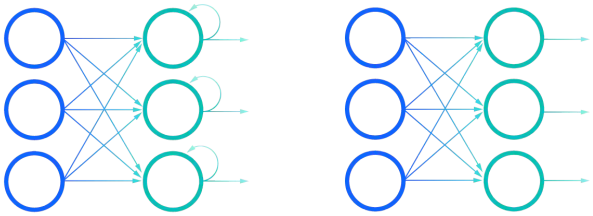

Fig. 7: Illustration of a RNN (left) and a feed forward neural network (right), respectively [17].

extra term in the loss function and reduces some weights to zero, producing sparse models [11].

We will now introduce the terms *batch-size* and *epoch*, both of which will be used frequently throughout this report. Data is typically passed through an ANN in so called batches, or smaller subsets of the entire dataset. The batch-size is the total number of training examples present in one batch. One epoch is done when an entire dataset is passed through an ANN exactly one time.

B. Long short-term memory

The most common and effective way to train long, sequential data is using a *recurrent neural network* (RNN) with LSTM, the difference between a feed forward neural network and a RNN is illustrated in Figure 7. LSTM has become the new state of the art for solving many previously difficult problems. This includes but is not limited to translation [14], speech modeling [15] and audio analysis [16].

The reason for LSTM-networks being effective is that the cells, which are the building blocks that make up the layers in an ANN, can keep track of their state over time, as well as having a *forget gate*. To show what this means we will follow the explanation formulated by Géron [11]. The cell is made up by three gates: an input gate, an output gate and a forget gate as, shown in Figure 6. Observing the figure whilst reading the explanation below is highly recommended. The long-term state $\mathbf{c_{(t-1)}}$ first goes through the forget gate, which makes it drop some memories. In the next stage it acquires new memories, chosen by the input gate. The resulting state vector $\mathbf{c_{(t)}}$ is then sent out of the network. However, after the addition of memories the long-term state vector is also copied, passed through the *tanh* function and filtered through the output gate. This whole procedure creates the short-term state $\mathbf{h_{(t)}}$, which is equal to the output $\mathbf{y_{(t)}}$. The input vector at time *t*, $\mathbf{x_{(t)}}$, and $\mathbf{h_{(t-1)}}$ are then fed to 4 different layers, the main layer outputs $\mathbf{g_{(t)}}$. The most important parts of this output is stored in the long-term state vector, and the rest is dropped.

The three other layers are all *gate controllers*. Their outputs are ranging from zero to one. A zero represents a closed gate and a one represents a gate being open. The *forget gate* decides which parts of the long-term state should be deleted, the *input gate* decides which parts of $\mathbf{g_{(t)}}$ should be added to the long-term state and the *output gate* decides which part of the long-term state that should be output in the present time step. We define the weights for a LSTM-layer as:

Input Weights: $\mathbf{W_{xi}}, \mathbf{W_{xf}}, \mathbf{W_{xo}}, \mathbf{W_{xg}}$.

Recurrent Weights: $\mathbf{W_{hi}}, \mathbf{W_{hf}}, \mathbf{W_{ho}}, \mathbf{W_{hg}}$.

Bias terms: $\mathbf{b_i}, \mathbf{b_f}, \mathbf{b_o}, \mathbf{b_g}$.

This gives us the following equation for how to compute the long-term state, the short-term state and the output at every time step:

$$
\begin{aligned}
\mathbf{i_{(t)}} &= \sigma(\mathbf{W_{xi}^T x_{(t)}} + \mathbf{W_{hi}^T h_{(t-1)}} + \mathbf{b_i}),\\
\mathbf{f_{(t)}} &= \sigma(\mathbf{W_{xf}^T x_{(t)}} + \mathbf{W_{hf}^T h_{(t-1)}} + \mathbf{b_f}),\\
\mathbf{o_{(t)}} &= \sigma(\mathbf{W_{xo}^T x_{(t)}} + \mathbf{W_{ho}^T h_{(t-1)}} + \mathbf{b_o}),\\
\mathbf{g_{(t)}} &= \tanh(\mathbf{W_{xg}^T x_{(t)}} + \mathbf{W_{hg}^T h_{(t-1)}} + \mathbf{b_g}),\\
\mathbf{c_{(t)}} &= \mathbf{f_{(t)}} \otimes \mathbf{c_{(t-1)}} + \mathbf{i_{(t)}} \otimes \mathbf{g_{(t)}},\\
\mathbf{y_{(t)}} &= \mathbf{o_{(t)}} \otimes \tanh(\mathbf{c_{(t)}}),
\end{aligned}
\tag{8}
$$

where σ(x) is the logistic function and $\otimes$ denotes element-wise multiplication.

This is the most commonly used version of a LSTM cell, but there are many other variants. The most common modification of the vanilla LSTM cell is the *gated recurrent unit* (GRU), which has a coupled input and forget gate.

C. Federated learning

There is a vast amount of data which is generated on IoT-devices around the world every day. Traditionally when training ANNs, or mathematical models in general, this data is collected by a centralized server, which also executes training.

FL proposes an approach where a set of clients learns locally and only sends an updated model to a global server. The global server learns a shared global model by aggregating the locally computed weights and biases. We will present a FL algorithm called FederatedAveraging (FedAvg) which was developed by Google researchers [4] and has shown promising results.

When developing any machine learning algorithm, we usually make the assumption that examples in each dataset are *independent* and *identically distributed*. If these assumptions

are met we call the data *IID*, but in a FL setting this is rarely the case. However, it is shown by McMahan et al. [4] that good results can be achieved even if the training data is non-IID.

Assume that there exists a global server and that there are K clients, each with a fixed local dataset. Consider the following neural network loss function

$$\min_{\omega \in R^d} f(\omega) \text{ where } f(\omega) := \frac{1}{N} \sum_{i=1}^{N} f_i(x_i, y_i, \omega), \tag{9}$$

where $f_i(\omega)$ is the loss of the prediction on example (x_i, y_i) given model parameters ω.

Let $\mathcal{P}_k$ refer to the dataset stored on client k and let n_k=$|\mathcal{P}_k|$. Equation 9 can now be written as

$$\begin{aligned} f(\omega) &= \textstyle\sum_{k=1}^{K} \frac{n_k}{n} F_k(\omega), \\ F_k(\omega) &= \frac{1}{n_k} \textstyle\sum_{i \in \mathcal{P}_k} f_i(x_i, y_i, \omega). \end{aligned} \tag{10}$$

Let g_k=$\nabla F_k(\omega_t)$ and let η be the learning rate. FedAvg initiates with the random selection of a fraction C of the clients, then the server communicates the current global state to each of the clients.

The selected clients proceed to perform their local training based on the global state and their local datasets. Their updates are then sent to the server, which in turn aggregates all the local updates and applies these to its global state.
The local model parameters are updated according to

$$\omega_{t+1}^k \leftarrow \omega_t^k - \eta g_k \; \forall k, \tag{11}$$

and then aggregated to the central server according to

$$\omega_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \omega_{t+1}^k. \tag{12}$$

The process is then repeated.

In other words, for each global step, each client trains locally using its local data and the server then takes a weighted average of the resulting local models. Note that when aggregating the local updates, the new updated weights are weighted to be proportional to the number of data samples for client k as n_k/n. This makes sure that clients with more data have a larger impact on the aggregated global model. The entire algorithm is described, in pseudo-code, in Figure 8.

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Fig. 8: Pseudo-code describing FedAvg [4].

Fig. 9: Illustration of centralized machine learning model with one LSTM-layer.

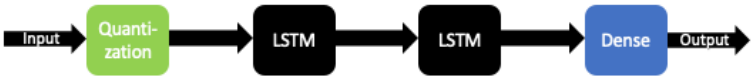

Fig. 10: Illustration of centralized machine learning model with two LSTM-layers.

III. Method

The centralized machine learning model will be described in Section III-A, the various software frameworks used in the project will be described in Section III-B, the process of simulating sensors will be explained in section III-C, the implementation of the FedAvg algorithm will be presented in section III-D and our choices regarding producing results will be presented in III-E.

A. Centralized machine learning model

The input, I of the centralized machine learning model is:

$$\begin{aligned} I &= TC + X, \\ X &\in \{SA, CN, RX, PH, OS\}. \end{aligned} \tag{13}$$

The temperature measurement will always be part of the input, the reason being that temperature highly correlates with all the other measurement types, hence we want our model to always take the temperature into account.

The output is the prediction of in what range the next measurement of X will be in. This means that we have to split the prediction range into bins. We are using a so-called quantile cut function in order to put all historical data into 6 equally large bins. We used 6 bins since this gave us a good balance between the ranges associated with the bins and the accuracy of the predictions.

Figure 9 and Figure 10 illustrate our model with one and two LSTM-layers, respectively.

B. Software frameworks

The centralized machine learning model had been implemented in *Python* using *Tensorflow 1.13*, which is a machine learning library developed by Google. It was therefore natural for us to also use these technologies in this project, all the code developed in this project is available on GitHub [18]. All data from the sensor has been available for us to use through *MongoDB*, which has allowed us to easily download the most up-to-date data for training.

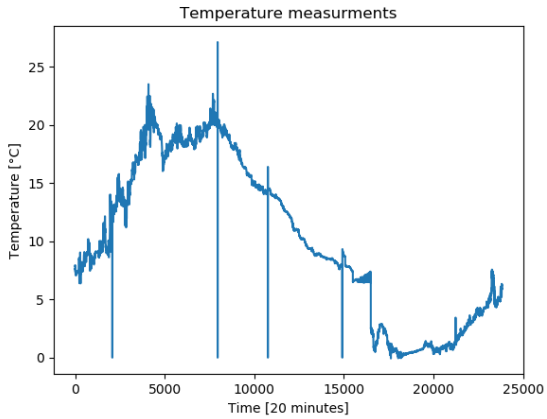

Fig. 11: Raw temperature data collected by the sensor during a time period of 365 days.

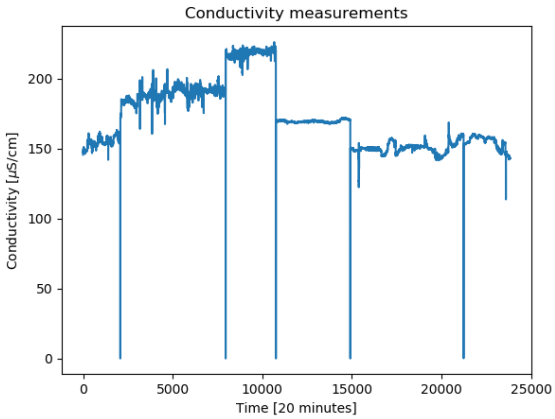

Fig. 14: Raw conductivity data collected by the sensor during a time period of 365 days.

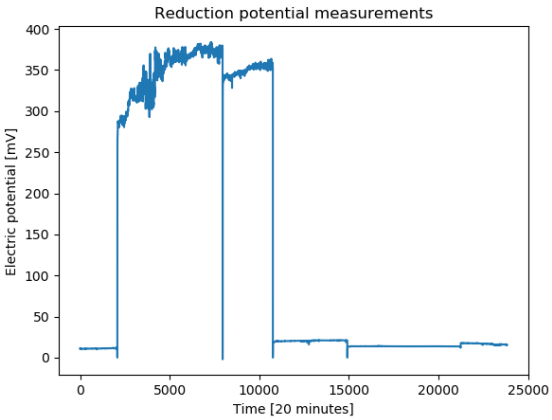

Fig. 12: Raw reduction potential data collected by the sensor during a time period of 365 days.

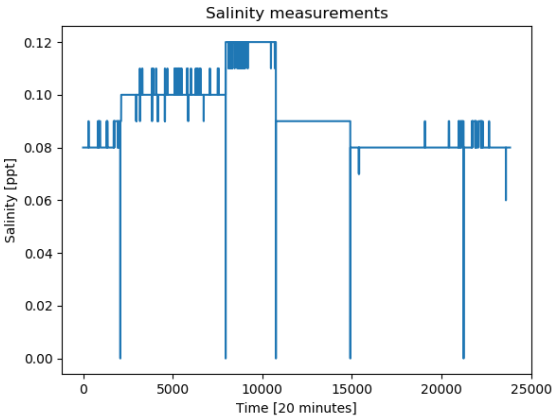

Fig. 15: Raw salinity data collected by the sensor during a time period of 365 days.

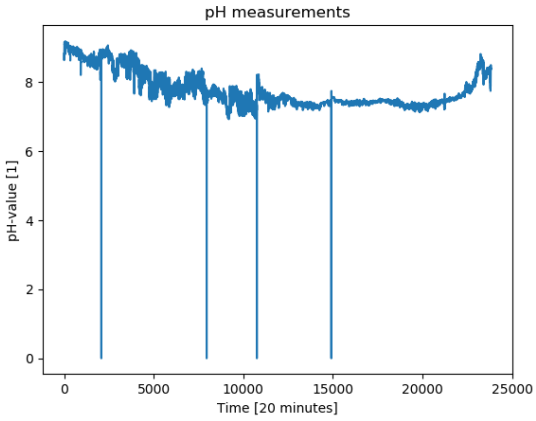

Fig. 13: Raw pH-value data collected by the sensor during a time period of 365 days.

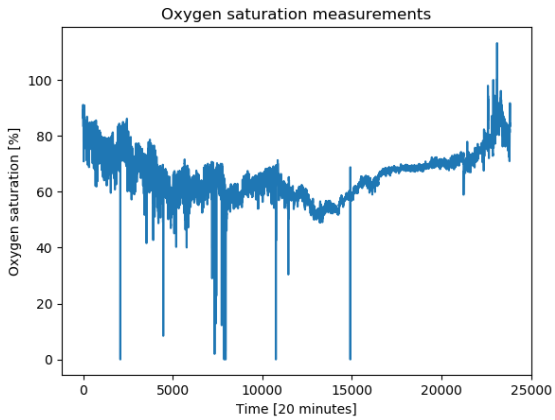

Fig. 16: Raw oxygen saturation data collected by the sensor during a time period of 365 days.

C. Data generation and simulation of sensors

FL requires multiple clients and since we only have had access to one sensor, we have had to virtually simulate clients. Clients were simulated by down-sampling from the data gathered by the only existing sensor, and then distributing it to K simulated clients.

TABLE II: Implementation parameters

Implementation parameters	Value
LSTM-cell-neurons of each LSTM-layer	50
Local batch size B	6
Loss function	Cross-entropy
Optimizer	Adam
Learning rate η	0.0005
L1-regularization-weight	0.0002
Number of clients K	5
Fraction of clients used in each round C	1

The physical sensor has a sampling period of 20 minutes, which means it makes a new measurement every 20 minutes, and uses a time window of 365 days. The data gathered by the sensor is distributed such that each simulated sensor gets a sampling period of 20·K minutes. If we for example simulate three sensors, every simulated sensor will get a new measurement every 60 minutes and will start gathering data at different times.

In Figures 11-16 we show the raw data measured by the sensor. The spikes in all figures occur as the sensor was malfunctioning at times. There has also been problems with algae being stuck on the sensor, which has caused issues with several measurements. More details on how this has affected our results will be brought up in the discussion.

D. Implementation parameters

The parameters used in our local machine learning models and in the FedAvg algorithm are shown in Table II.

When producing results we have been using 5 clients, which is around the realistic amount to be bought and deployed. The L1-regularization value of 0.0002 was used in the already developed LSTM-network, which worked well, and has therefore not been changed. The AdamOptimizer was used as optimizer. We settled for the learning rate $\eta = 0.0005$ since this value provided high accuracy without training taking too long. The centralized machine learning model had a batch-size of 30 training examples. Since we split up the data gathered by the sensor to 5 different clients, we set the local batch-size B to 6, as every client had access to one fifth of the data. The fraction of clients used in each global round C, was set to one. Lowering this fraction would reduce computing costs. However we did not want one more hyperparameter to tweak, and computing costs was not an issue. Both one and two LSTM-layers have been used.

E. Measuring accuracy and producing results

The global accuracy for the FL model was supposed to be measured as the total number of correct predictions divided by the total number of predictions made by each local model. In our case we decided to measure the global accuracy as the average local accuracy of all the clients. This is a good estimation of the global accuracy since all the clients have access to virtually the same amount of data, it only differs by maximum one training example. 10% of the data is used for testing and the rest for training.

TABLE III: Accuracy of the federated algorithm vs a centralized model for all measurement types

Measurement type	Global accuracy	Centralized accuracy
Salinity (SA)	0.98170	0.98816
Conductivity (CN)	0.92052	0.94123
Reduction potential (RX)	0.94079	0.97124
pH-value (PH)	0.76209	0.87489
Oxygen saturation (OS)	0.73823	0.87194

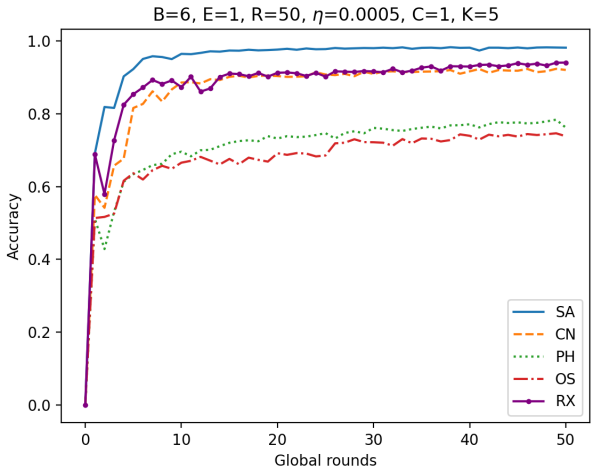

Fig. 17: Global accuracy, with one local epoch E and 50 global rounds R, of each measurement type plotted against number of global rounds R. The final values are shown in Table III.

TABLE IV: Global and centralized accuracy for input CN with different amount of total epochs, and the ratio between them

E/R/CE	Global accuracy	Centralized accuracy	CA ÷ GA
1/50/50	0.92052	0.94912	1.03107
5/50/250	0.93433	0.95332	1.02032
10/50/500	0.93562	0.95332	1.01892

When comparing the global and centralized accuracy, we decided to use the same number of total epochs in both cases. For example, if we use 250 epochs in the centralized training then $R \cdot E = 250$, where R denotes the number of global rounds and E denotes the number of local epochs, in the FL training.

We have also decided to use the measurements of CN in most of our experiments since the gathered data seemed relatively regular compared to other measurement types.

IV. Results

In Figure 17, the global accuracy is plotted against the number of global rounds for every available measurement type, when E=1 and R=50. The final global accuracy for each measurement type is provided in Table III as well as the final accuracy of a similar centralized model. The centralized model is 7.8% more accurate on average for the different measurement types than the federated model.

In Figure 18, the global accuracy is plotted against the number of global rounds when using one and two LSTM-layers and CN as measurement type, when E=1 and R=50.

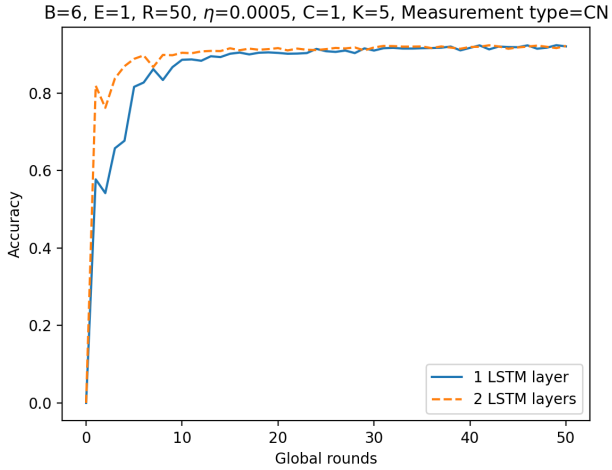

Fig. 18: Comparison of performance using one and two LSTM-layers.

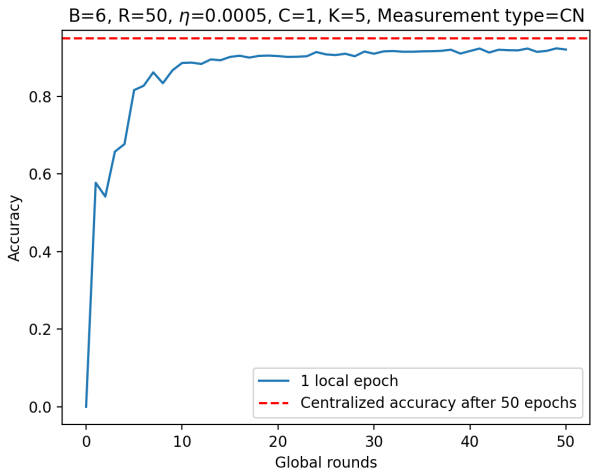

Fig. 19: Global accuracy with one local epoch and 50 global rounds plotted together with final accuracy of centralized model after 50 epochs. The final values are shown in the first row of Table IV.

In Figure 19-21 the global accuracy has been plotted against the number of global rounds for one, 5 and 10 local epochs E, respectively. The final global accuracy for each figure is provided in Table IV as well as the final accuracy of a similar centralized model for each case. In Table IV, CE denotes the number of epochs used in the centralized model. CA and GA are used as abbreviations for centralized accuracy and global accuracy, respectively.

CN is the measurement type used when producing the results presented in Figure 19-21 and Table IV.

V. Discussion

The results presented in Figure 17 show how different measurement types have reached significantly different accuracy. There are some obvious reasons to why this has happened and all of them are related to the raw data gathered by the sensor.

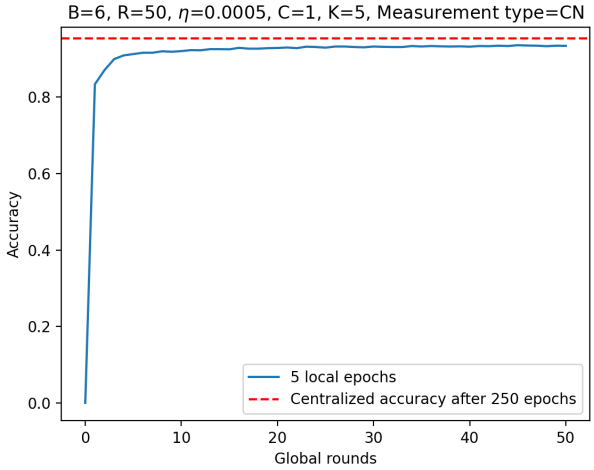

Fig. 20: Global accuracy with 5 local epochs and 50 global rounds plotted together with final accuracy of centralized model after 250 epochs. The final values are shown in the second row of Table IV.

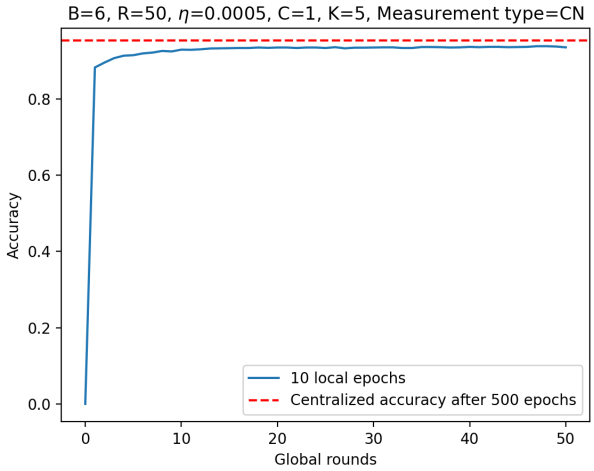

Fig. 21: Global accuracy with 10 local epochs and 50 global rounds plotted together with final accuracy of centralized model after 500 epochs. The final values are shown in the third row of Table IV.

If we look back to Figures 11-16 we realize that the measurements of OS and PH are the most volatile. This makes it harder for our machine learning model to predict them. The raw data also indicates that the measurement type most affected by sensor malfunctions and the algae issue discussed earlier, is RX. Paradoxically, this is one of the measurement types on which the FL model has attained the highest accuracy. This does however seem reasonable, since during the time the sensor has been malfunctioning, the fluctuation of the reduction potential data has decreased, which makes it more predictable.

The downwards spikes that can be observed in the raw data presented in Figures 11-16 originates from the sensor being completely shut down. Since this has not happened more than

five times over the course of the last year, which is the time window used for collecting data, we do not think that the impact of the sensor being shut down has had a significant effect on the prediction results.

The results presented in Table III show that the centralized model outperforms the federated model for all the measurement types. On average, the centralized model has a 7.8 % better accuracy than the federated model. However, this does not mean that the federated model produced in this project is useless. It is important to realize, as discussed in section III-C, that the amount of data available to train the simulated sensors has been reduced, which has led to the clients only getting access to one fifth of the data that they would have had access to if they were physical sensors. With this in mind, the results presented in Table III should be seen as promising, and as an incentive to conduct more research regarding deploying more physical sensors.

The results presented in Figure 18 indicates that faster convergence is achieved when using two rather than one LSTM-layer, however, the training time increases significantly as more weights and biases have to be tweaked.

The results presented in Figures 19-21 and Table IV seem to indicate that the global accuracy tends towards the centralized accuracy as the number of local epochs increase. If we would increase the number of local epochs and/or global rounds it could affect the global accuracy in three possible ways:

- It never reaches the centralized accuracy;
- It converges to the same accuracy as the centralized accuracy;
- It surpasses the centralized accuracy.

It would have been interesting to experiment with more local epochs and/or global rounds, however, this was too time consuming for the scope of this project.

VI. Conclusion

In this project we have implemented a FL algorithm with the intentions of detecting anomalies in measurements made by a water monitoring IoT-sensor.

Depending on the different measurement types, we reached an accuracy in the range of 73.8 % to 98.2 %. These results can be compared with a, previously developed, centralized machine learning model which achieved an accuracy in the range of 87.2 % to 98.8 %. On average, the centralized model had a 7.8 % better accuracy than the federated model. However, these results do not necessarily mean that the use of FL to improve anomaly-detection should be discarded, as simulating sensors have significantly reduced the training data for each simulated sensor. Problems with the deployed sensor, for example algae being stuck on it, have also had a negative impact on the results.

In summary, we believe that FL has the potential to produce interesting results, provided that more physical sensors are deployed.

VII. Future work

There are several aspects of our project that could be expanded upon, experimenting with different number of bins being one of them. It would also have been interesting to remove outliers from data in order to explore if better accuracy could be achieved. Another activity worthy of further investigation would have been to experiment with using more local epochs and/or global rounds in order to explore if the federated model converges to, or even outperforms, the accuracy of the centralized model.

Since most of the unsatisfying results were related to problems with the sensor, it would be reasonable to conduct further research with fully functioning equipment. This could be achieved either by more frequent cleaning and better maintenance of the sensor, or by investing in a better sensor.

Another interesting extension of the problem would be to compare a FL approach with a multi-client centralized approach. A multi-client centralized approach would imply uploading all the data from all the sensors to a main server, execute training on the server and then download the final global model to all clients for predictions. In order to conduct this comparison, additional physical sensors have to be deployed.

Acknowledgment

We would like to thank our supervisor José Mairton Barros da Silva Júnior for his invaluable support and guidance during the project.

References

[1] M. Gusev and S. Dustdar, "Going back to the roots—the evolution of edge computing, an iot perspective," *IEEE Internet Computing*, vol. 22, no. 2, pp. 5–15, 2018.

[2] H. Zheng, H. Hu, and Z. Han, "Preserving user privacy for machine learning: Local differential privacy or federated machine learning?" *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 5–14, 2020.

[3] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, May. 2020.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. Fort Lauderdale, FL, USA: PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html

[5] A. Andersson. (2021, Apr.) iwater: Digital overvakning av stadens vattenkvalitet. [Online]. Available: http://miljobarometern.stockholm.se/vatten/samarbeten-och-projekt/iwater-digital-overvakning-av-stadens-vattenkvalitet/

[6] "Directive 2000/60/ec of the european parliament and of the council of 23 october 2000 establishing a framework for community action in the field of water policy," *Official Journal*, Sep. 2014. [Online]. Available: http://data.europa.eu/eli/dir/2000/60/2014-11-20

[7] G. Destouni, I. Fischer, and C. Prieto, "Water quality and ecosystem management: Data-driven reality check of effects in streams and lakes," *Water Resources Research*, vol. 53, no. 8, pp. 6395–6406, 2017. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2016WR019954

[8] J. Salonsaari. (2021, Apr.) Projektsbeskrivningsmall iot for innovativ samhallsutveckling, utlysning hosten 2017- varen 2018. [Online]. Available: https://www.digitaldemostockholm.com/media/1037/iwater-vinnova-approved-1.pdf

[9] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[10] F. Rosenblatt, "The perceptron - a perceiving and recognizing automaton." Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[11] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. Sebastopol, California: O'Reilly Media, Inc., Sep. 2019, ch. 15.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.
[14] X. Huang, H. Tan, G. Lin, and Y. Tian, “A lstm-based bidirectional translation model for optimizing rare words and terminologies,” in *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, 2018, pp. 185–189.
[15] S. An, Z. Ling, and L. Dai, “Emotional statistical parametric speech synthesis using lstm-rnns,” in *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2017, pp. 1613–1616.
[16] L. Huang and C.-M. Pun, “Audio replay spoof attack detection using segment-based hybrid feature and densenet-lstm network,” in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 2567–2571.
[17] I. C. Education. (2021, Apr.) Recurrent neural networks. [Online]. Available: https://www.ibm.com/cloud/learn/recurrent-neural-networks
[18] M. Wallén and M. Böckin. (2021, May.). [Online]. Available: https://github.com/mauriciobvaldes/FedAvg_iWater