Predictors for Trickling in WSN

INE5424 Operating Systems II

Douglas Martins
Lucas May Petry
Maike de Paula Santos

Department of Informatics and Statistics
Federal University of Santa Catarina

Florianópolis, June 28^{th}, 2017

Outline

Motivation & Objective

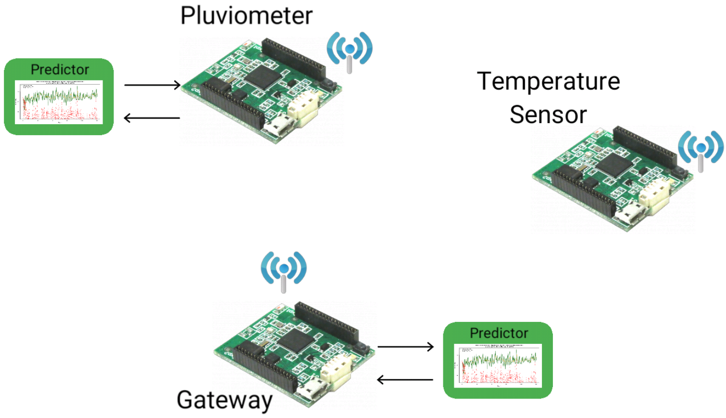
Motivation

- IoT environment with numerous devices
- Power consumption is an issue (devices' batteries)
- Communication between sensors consumes a lot of power

Objective

- Improvement of energy efficiency by reducing communication
- Development of **SmartData** predictors

Motivation & Objective



Powered by DrawExpress

Project Plan
Tasks

1. Study system restrictions and limitations of EPOSMote III and elaborate implementation strategies, such as design patterns ✓
2. Code and test the chosen prediction algorithms in C++ for a general purpose operating system ✓
3. Code and test the linear regression with gradient descent predictor on EPOS ✓
4. Code and test the Elman neural network on EPOS[1]
5. Deploy the final product using the UFSC IoT Gateway[1]

[1] See project changes

Changes & Issues

Changes

- Elman Neural Network to Multilayer Perceptron Network
- Deploy the final product using the UFSC IoT Gateway to test with data from UFSC IoT

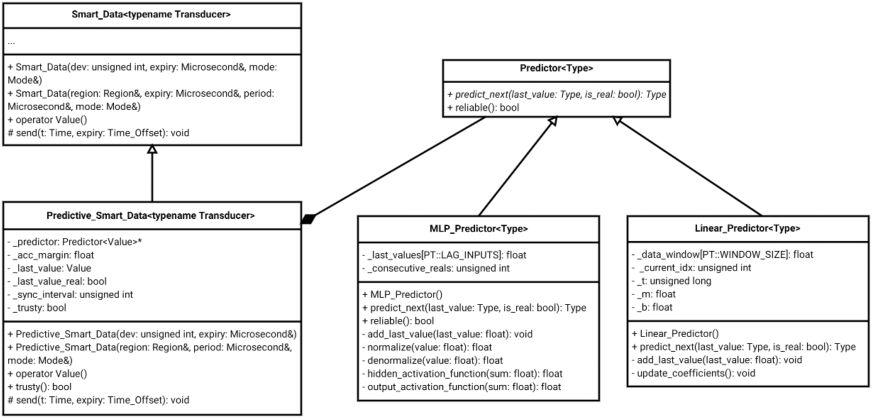
Issues

- Synchronization of predictors
- Implementation of exponential function

Douglas Martins
Lucas May Petry
Maike de Paula Santos

Project Design

Class Diagram



Project Design
Design Patterns

Design Patterns

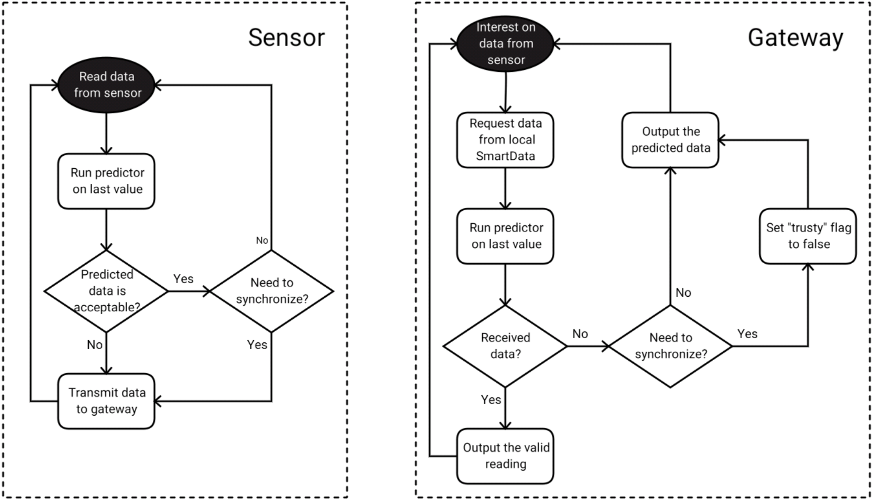
- Bridge
- Composition

Programming Techniques

- Generic programming
- Static mettaprogramming

Project Design

Sensor and Gateway Operation

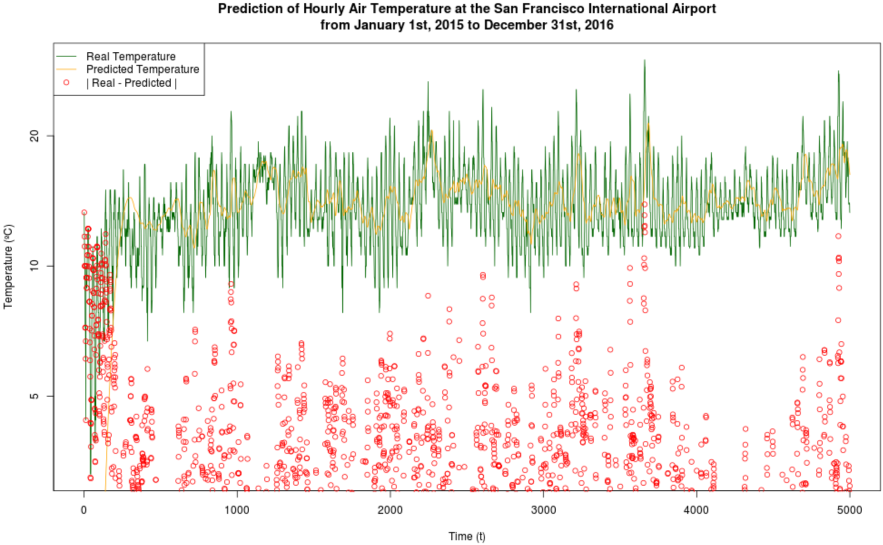


Linear Regression Predictor with Gradient Descent

- Performance tests and simulation in C++
- Static parameterization via Traits.h on EPOS
- Generic type prediction

Linear Regression Predictor

Performance Tests

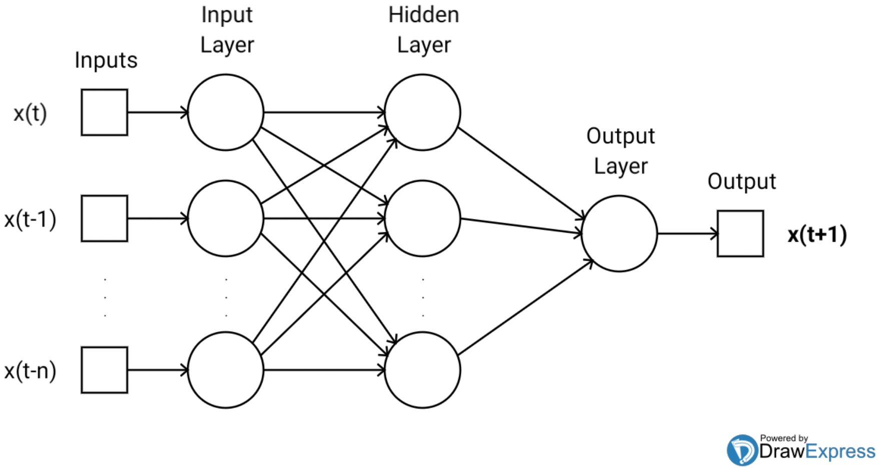


MLP Predictor

- Prediction of time series $x(t)$ based on n last values of $x(t)$.
- Requires historical data of the series to be predicted.
- Neural network training on a third-party software[2].

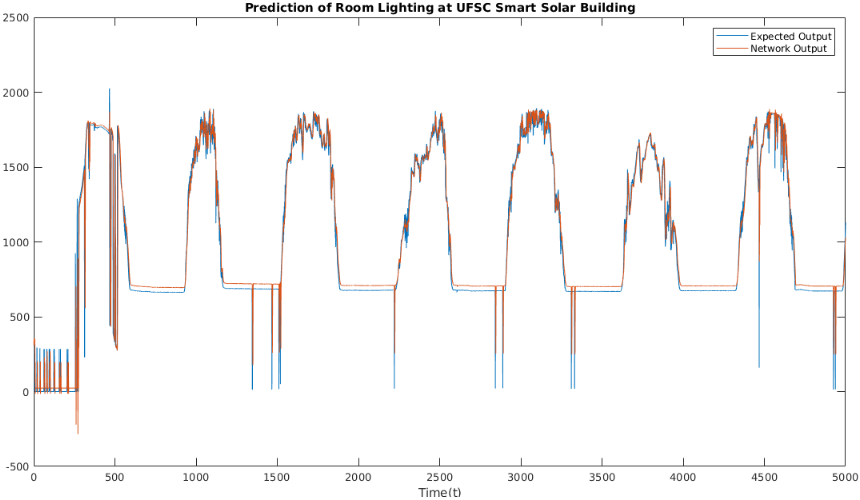
[2] MATLAB - MathWorks

Multilayer Perceptron Network



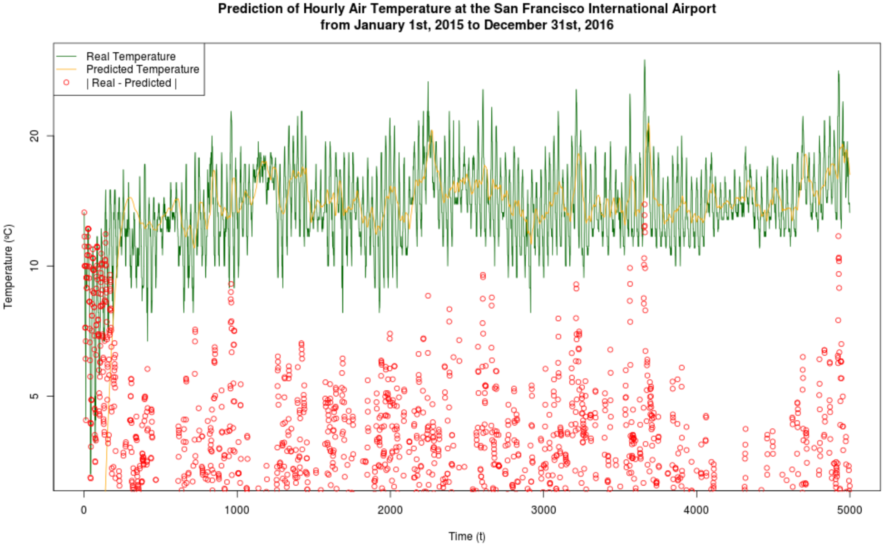
MLP Predictor

Performance Tests



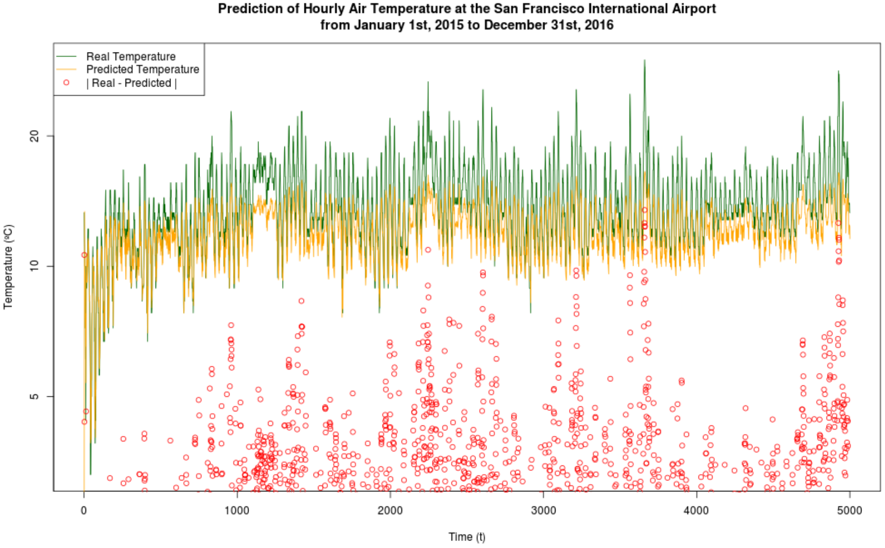
Linear Regression vs. MLP Predictor

Linear Regression Predictor



Linear Regression vs. MLP Predictor

MLP Predictor



Linear Regression vs. MLP Predictor

Overall Comparison

Feature	Linear Predictor	MLP Predictor
Type	Dynamic	Static
Online learning	✓	
Needs series historical data		✓
Resilient to desynchronization		✓
No preprocessing needed	✓	

Conclusion & Future Work

- Project planning requirements fulfilled
- Achieved satisfying results
- Future Work
  - Develop other types of predictors
  - Expand neural network implementation

Project Demo

Project demonstration

Predictive_Smart_Data

Traits

```
template<> template <typename S> struct Traits<
    Predictive_Smart_Data<S>>: public Traits<
    Smart_Data<S>>
{
    enum {LINEAR, MLP};

    static const bool debugged = true;
    static const unsigned int ACC_MARGIN = 8;
    static const unsigned int PREDICTOR = LINEAR;
    static const unsigned int SYNC_INTERVAL = 2;
};
```

Predictive_Smart_Data

Predictor Selection

```
private:
 typedef Traits<Predictive_Smart_Data<Transducer>> PT;
 typedef typename IF<(PT::PREDICTOR == PT::LINEAR),
                 Linear_Predictor<Value>,
                 MLP_Predictor<Value>>::Result P_Type;
```

Predictive_Smart_Data

send()

```
void send(const Time t, Time_Offset expiry) {
    Value predicted = _predictor->predict_next(
        _last_value, _last_value_real);
    Value real = Smart_Data<Transducer>::_value;

    bool acceptable_margins, check_sync = ...

    if(acceptable_margins && check_sync) {
        _last_value = predicted;
        _last_value_real = false;
        _sync_interval--;
    } else {
        _last_value = real;
        _last_value_real = true;
        Smart_Data<Transducer>::send(t, expiry);
        _sync_interval = PT::SYNC_INTERVAL;
    }
}
```

Predictive_Smart_Data

operator Value()

```
operator Value() {
  Value predicted;
  bool device_remote, sensor_side = ...
  if(device_remote)
    predicted = _predictor->predict_next(
      _last_value, _last_value_real);
  if(Smart_Data<Transducer>::expired()) {
    if(sensor_side) {
      Transducer::sense(
        Smart_Data<Transducer>::_device, this);
      Smart_Data<Transducer>::_time = TSTP::now();
    } else {
      _last_value = predicted;
      _last_value_real = false;
      if(PT::SYNC_INTERVAL && !_sync_interval)
        _trusty = false;
      if(_sync_interval > 0)
        _sync_interval--;
    }
  }
...
```

Predictive_Smart_Data

operator Value()

```
...
  else {
    _last_value = Smart_Data<Transducer>::_value;
    _last_value_real = true;
    if(device_remote)
      _sync_interval = PT::SYNC_INTERVAL;
    }

  if(!_trusty)
    _trusty = _predictor->reliable();
  return _last_value;
}
```

Predictor

Predictor Interface

```
template<typename Type>
class Predictor
{
public:
    virtual Type predict_next(
        Type last_value, bool is_real = false) = 0;
    virtual bool reliable() { return false; };
};

template<typename Type>
class Linear_Predictor: public Predictor<Type>
{
  ...
};

template<typename Type>
class MLP_Predictor: public Predictor<Type>
{
  ...
};
```

Linear_Predictor

Traits

```
template <typename S> struct Traits<Linear_Predictor<S
    >>: public Traits<void>
{
    static const bool debugged = true;
    static const unsigned int WINDOW_SIZE = 30;
    static const float LRATE;
    static const unsigned short GD_ITERATIONS = 200;
    static const unsigned short M = 0;
    static const unsigned short B = 10;
};
template <typename S> const float Traits<
    Linear_Predictor<S>>::LRATE = 0.000000001f;
```

MLP_Predictor

Traits

```
template <typename S> struct Traits<MLP_Predictor<S>>:
     public Traits<void>
{
    static const bool debugged = true;
    static const unsigned int HIDDEN_UNITS = 5;
    static const unsigned int LAG_INPUTS = 3;
    static const float HIDDEN_WEIGHTS[HIDDEN_UNITS*
        LAG_INPUTS];
    static const float HIDDEN_BIASES[HIDDEN_UNITS];
    static const float OUTPUT_WEIGHTS[HIDDEN_UNITS];
    static const float OUTPUT_BIAS;
    static const bool NORMALIZATION = true;
    static const float NORMALIZATION_MIN;
    static const float NORMALIZATION_MAX;
};
```