Machine Learning for Probabilistic Robotics with Webots

Joan Gerard[1]
Promotor: Prof. Gianluca Bontempi [1]

[1]Université Libre de Bruxelles

February 6, 2020

Table of Contents

Training Neural Network

Particles Filter

Update Odometry Values

Table of Contents

Training Neural Network

Particles Filter

Update Odometry Values

What is new?

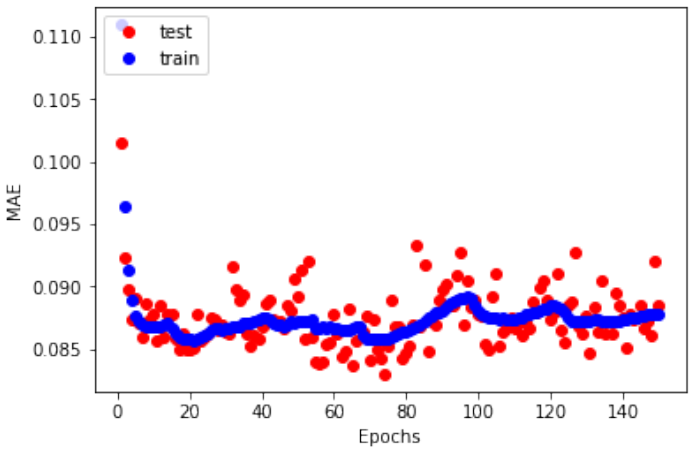
- Capture the real position of the robot + sensors measurements each 25 steps only
- In 10 minutes simulation (fast mode) 65342 samples captured
- All sensors has a lookup table with $\sigma = 0.2$ (± 2 cm)

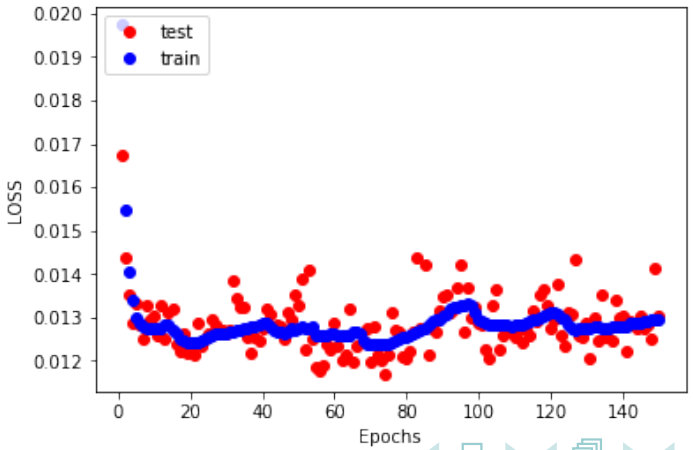
Preparing Data

- Normalize data
- Delete NA rows
- Shuffle data

Neural Network Configuration

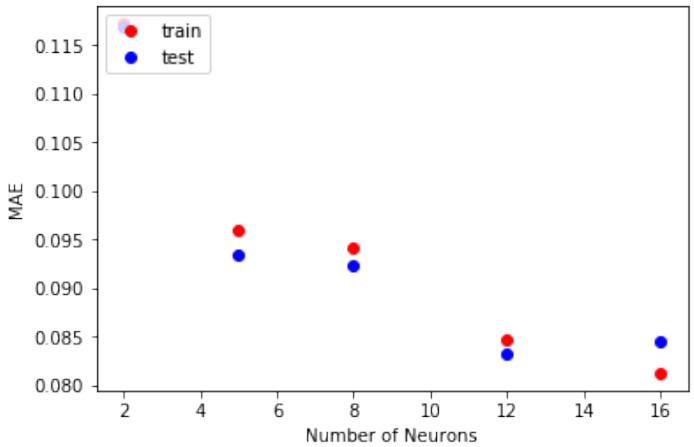
- One hidden layer with "relu" as optimizer
- 150 ephochs
- 5-fold
- 10 hidden neurons
- Loss: MSE
- Metric: MAE

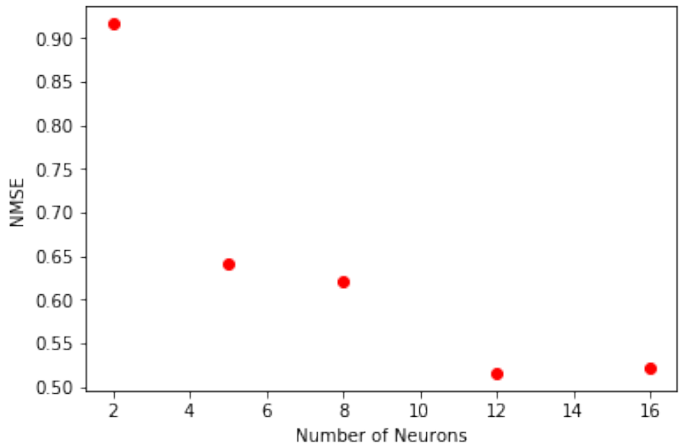




Tuning hidden neurons

- 3-fold
- 75 epochs
- Number of Neurons: 2, 5, 8, 12, 16





Final model

- NN trained with 75 epochs
- 12 neurons in hidden layer

Table of Contents

Training Neural Network

Particles Filter

Update Odometry Values

Configuration

- 50 particles used
- The initial position of all the particles are determined by the initial position of the robot.

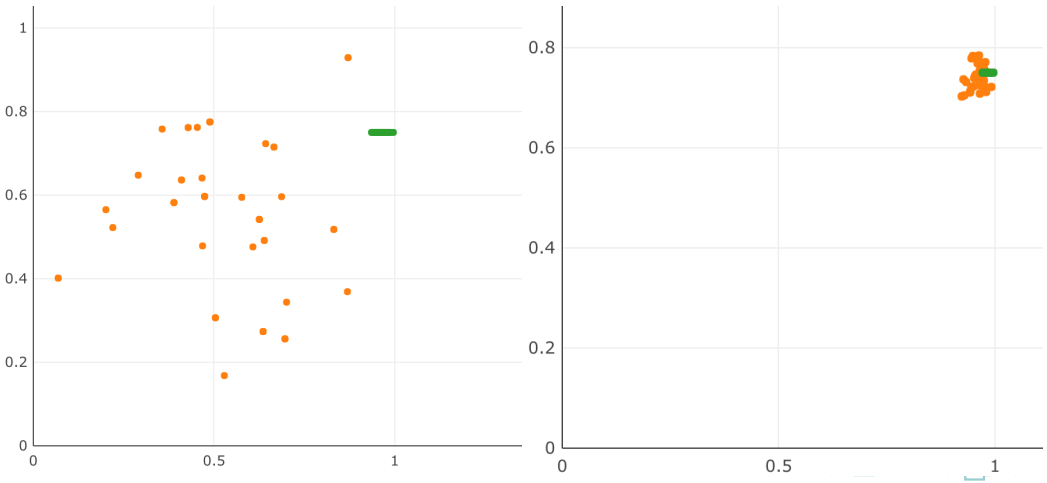
Pseudo code algorithm

input : robotState (odometry data), sensorsData
output: particles

```
1  deltaMove ← robotState - previousRobotState
2  particles.state = particles.state + deltaMove
3  addRandomMove(particles.state)
4  foreach particle ∈ particles do
5  |   particle.weight ← predictError(particle.state, sensorsData)
6  end
7  invertParticlesWeight(particles)
8  normalizeWeights(particles)
9  particles = resamplingBasedOnWeights(prob=particles.weight,
     replace=True)
10 previousRobotState ← robotState
11 return particles
```

Add Random Move

- Given the state of each particle (x, y, θ) it adds a random value following a normal distribution.
- For (x, y): $\mathcal{N}(0, 0.008)$
- For θ : $\mathcal{N}(0, 0.015)$
- Tested with other values as well. Left image corresponds to $\mathcal{N}(0, 0.08)$ for (x, y) and $\mathcal{N}(0, 1.5)$ for θ. Right image corresponds to the values mentioned above.



Predict Error

- Let y_i be the ith sensor measurement
- Let $\hat{y}_i$ be the prediction of the ith sensor measurement
- Normalize the inputs: (x, y, θ) using this as a reference
- Make a prediction using the neural network and the normalized inputs
- Denormalize the outputs
- Calculate error: $\sum_{i=1}^{i=8}(\hat{y}_i - y_i)^2$

Invert Particles Weight

- The prediction error algorithm will return a low value when the predicted sensor measurements will be close to the true sensor measurements. Thus the closer the particle to the real state, the lower its weight. We want exactly the opposite.
- To invert this we can apply:

```
1 maxWeight = max(particles.weight)
2 minWeight = min(particles.weight)
3 particles.weight = (maxWeight + minWeight) - particles.weight
```

Table of Contents

Training Neural Network

Particles Filter

Update Odometry Values

- The best particle will be the one whose weight is lower than the rest of the particles
- Each 25 steps the best particle is selected and the odometry data is updated with the selected particle's state.
- Results