Deep Kalman Filters and Applications

Michał Piotrowski

September 5, 2023

Praca licencjacka napisana pod kierunkiem dr. hab.
Piotra Wnuk-Lipińskiego

1 Introduction

In this work we explore the Deep Kalman Filter algorithm. Deep Kalman Filter algorithm is a modification of the Kalman Filter, which is used to estimate unknown variables from a series of measurements observed over time, Kalman Filter is a linear dynamical system, this constrains its expressivity. Deep Kalman Filter develops on that by utilizing deep neural networks to be able to learn arbitrarily complex dynamics. We check the performance of Deep Kalman Filter on various datasets and propose a small change to the algorithm that improves performance in certain situations.

2 Kalman Filter

We start by describing and deriving the Kalman Filter [1]. Kalman Filter is an algorithm that is used to create a linear estimate of unknown variables from a series of measurements observed over time. We will describe the fundamental building blocks of the Kalman Filter below.

2.1 Mean squared error

Kalman Filter is a mean squared error minimizer, so we will write a bit about mean squared error.
Various signals can be described as:

$$y_t = g_t x_t + w_t \tag{1}$$

y_t is an observed signal dependant on time, g_t is a gain, x_t is a signal, w_t is an additive noise.
The objective is an estimation of x_t. The difference between estimator $\hat{x_t}$ and an estimated value x_t is called an error:

$$f(e_t) = f(x_t - \hat{x}_t) \tag{2}$$

The concrete shape of the error function $f(e_t)$ depends on the application but it should be positive and monotonically increasing. The example is a squared error function:

$$f(e_t) = (x_t - \hat{x}_t) \tag{3}$$

We want to predict a lot of data in time, so in our applications it makes more sense to use the expected value of the error function, from this we get the mean squared error (MSE):

Definition 2.1. Mean squared error (MSE) of the estimator $\hat{x}_t$ is:

$$\mathrm{MSE}(\hat{x}_t) = \mathbb{E}(x_t - \hat{x}_t)^2 \tag{4}$$

2.2 Maximum likelihood estimation

The above derivation of mean squared error is not rigorous, so we will derive it using maximum likelihood statistics. Let the goal of the Kalman Filter be finding $\hat{x}$ which maximizes the probability or likelihood of y.

$$\max[P(y|\hat{x})] \tag{5}$$

By assuming that the additive noise is Gaussian distributed with a standard deviation of σ_k we get

$$P(y_t|\hat{x}_t) = U_t \exp\left(-\frac{(y_t - g_t\hat{x}_t)^2}{2\sigma_t^2}\right) \tag{6}$$

where U_t is a normalization constant.
The maximum likelihood function is

$$P(y|\hat{x}) = \prod_t U_t \exp. \tag{7}$$

After applying logarithm we get

$$\log P(y|\hat{x}) = -\frac{1}{2}\sum_t \left(\frac{(y_t - g_t\hat{x}_t)^2}{\sigma_t^2}\right) + \text{stała} \tag{8}$$

2.3 Derivation of Kalman Filter

We will mathematically derive Kalman Filter here. Let's say that we want to get to know the value of the variable of a process of the form

$$x_{t+1} = Ax_t + w_t \tag{9}$$

where x_t is the state vector of the process at time t, $x_t \in \mathbb{R}^n$, A is the state transition matrix of the process from the state aat time t to the state at time $t+1$, it is assumed to be stationary over time and $A \in \mathbb{R}^{x \times m}$, w_t is a white

noise process with known covariance, $w_t \in \mathbb{R}^n$.
Observation on this variable can be modelled as

$$z_t = Hx_t + v_t \tag{10}$$

where z_t is a measurement of x at time t, H is a noiseless matrix that connects the state vector and the measurement vector, is stationary over time, $H \in \mathbb{R}^{m\times n}$, v_t is the measurement error, assumed to be a white noise process with known covariance, has zero cross-correlation with the process noise, $v_t \in \mathbb{R}^m$.
For minimization of MSE to give an optimal Kalman filter we have to model errors of the model using normal distributions. The covaraiances of both noise models are stationary in time and given by

$$U = E[w_t w_t^T] \tag{11}$$

$$P = E[v_t v_t^T] \tag{12}$$

The mean squared error is

$$E[e_t e_t^T] = R_t \tag{13}$$

where R_t is the error covariance matrix at time t, $R_t \in \mathbb{R}^{n\times n}$
The above can be expanded to give

$$R_t = E[e_t e_t^T] = E\left[(x_t - \hat{x}_t)(x_t - \hat{x}_t)^T\right] \tag{14}$$

Let $\hat{x}'_t$ be the prior estimate of $\hat{x}_t$. Then we can write an update equation

$$\hat{x} = \hat{x}'_t + K_t(z_t - H\hat{x}'_t) \tag{15}$$

where K_t is the Kalman gain.
By substituting (10) we get

$$\hat{x}_t = \hat{x}'_t + K_t(Hx_t + v_t - H\hat{x}'_t) \tag{16}$$

By substituting (16) into (14) we get

$$\begin{aligned} R_t = E[&[(I - K_t H)(x_t - \hat{x}'_t) - K_t v_t] \\ &[(I - K_t H)(x_t - \hat{x}'_t) - K_t v_t]^T\Big] \end{aligned} \tag{17}$$

Let's note that $x_t - \hat{x}'_t$ is the error of the prior estimate. Because this is uncorrelated with the measurement noise we can rewrite expectation as:

$$\begin{aligned} R_t = &(I - K_t H)E\left[(x_t - \hat{x}'_t)(x_t - \hat{x}'_t)^T\right](I - K_t H) \\ &+ K_t[v_t v_t^T]K_t^T \end{aligned} \tag{18}$$

Let's substitute equations (12) and (14) to get:

$$R_t = (I - K_t H)R'_t(I - K_t H)^T + K_t P K_t^T \tag{19}$$

where R'_t is the prior estimate of R_t.
Equation (19) is the error covariance update equation. The diagonal of the covariance matrix contains the mean squared error as shown:

$$R_{tt} = \begin{bmatrix} E\left[e_{t-1}e_{t-1}^T\right] & E\left[e_t e_{e_{t-1}}^T\right] & E\left[e_{t+1}e_{t-1}^T\right] \\ E\left[e_{t-1}e_t^T\right] & E\left[e_t e_t^T\right] & E\left[e_{t+1}e_t^T\right] \\ E\left[e_{t-1}e_{t+1}^k\right] & E\left[e_t e_{t+1}^k\right] & E\left[e_{t+1}e_{t+1}^T\right] \end{bmatrix} \tag{20}$$

Trace of the error covariance matrix is the sum of the mean squared errors. So we can minimize the mean squared error by minimizing the trace of R_t, which will minimize the trace of R_{tt}.
We can find the condition of this minimum by first differentiating the trace of R_t with respect to K_t and setting the result to zero. Let's expand (19) to get:

$$R_t = R'_t - R'_t H^T K_t^T - K_t H R'_t + K_t H R'_t H^T K_t^T + K_t P K_t^T \tag{21}$$

Let's observe that the trace of a matrix is equal to the trace of its transpose, so we can write it in a following way:

$$tr(R_t) = tr(R'_t) - 2tr(K_t H R'_t) + tr(K_t(H R'_t H^T + P)K_t^T) \tag{22}$$

where tr(A) is a trace of a matrix A.
Let's differentiate with respect to K_t to get:

$$\frac{dtr(R_t)}{dK_t} = -2(H R'_t)^T + 2K_t(H R'_t H^T + P) \tag{23}$$

Let's set this to zero and re-arrange to get:

$$(H R'_t)^T = K_t(H R'_t H^T + P) \tag{24}$$

Let's solve for K_t to get:

$$K_t = R'_t H^T (H R'_t H^T + P)^{-1} \tag{25}$$

Equation (25) is the Kalman gain equation.
Let's substitute equation (25) into (21) to get:

$$\begin{aligned} R_t &= R'_t - R'_t H^T (H R'_t H^T + P)^{-1} H R'_t \\ &= R'_t - K_t H R'_t \\ &= (I - K_t H) R'_t \end{aligned} \tag{26}$$

Equation (26) is the update equation for the error covariance matrix with optimal gain. The three equations (15), (25), (26) develop an estimate of the variable x_t. State projection is achieved using:

$$\hat{x}'_{t+1} = A\hat{x}_t \tag{27}$$

To complete the recursion it is necessary to find an equation which projects the error covariance matrix into the next time interval, $t+1$. This is achieved by first forming an expression for the prior error:

$$\begin{aligned} e'_{t+1} &= x_{t+1} - \hat{x}'_{t+1} \\ &= (Ax_t + w_t) - A\hat{x}_t \\ &= Ae_t + w_t \end{aligned} \tag{28}$$

Let's extend equation (14) to time $t+1$:

$$R'_{t+1} = E[e'_{t+1}{e^T}'_{t+1}] = E\left[(Ae_t + w_t)(Ae_t + w_t)^T\right] \tag{29}$$

Let's note that e_t and w_t have zero cross-correlation because the noise w_t actually accumulates between t and $t+1$ wheres the error e_t is the error up until time t. Therefore:

$$\begin{aligned} R'_{t+1} &= E[e'_{t+1}{e^T}'_{t+1}] \\ &= E[Ae_t(Ae_t)^T] + E[w_t w_t^T] \\ &= AR_tA^T + U \end{aligned} \tag{30}$$

This completes the derivation.

Description	Equation
Kalman Gain	$K_t = R'_i H^T \left(HR'_\kappa H^T + P\right)^{-1}$
Update Estimate	$\hat{x}_t = \hat{x}'_t + K_t\left(z_t - H\hat{x}'_t\right)$
Update Covariance	$R_t = \left(I - K_t H\right) R'_t$
Project into $t+1$	$\hat{x}'_{t+1} = A\hat{x}_t$ $R_{t+1} = AR_tA^T + U$

Figure 1: Kalman Filter Recursive Algorithm

3 Deep Kalman Filter

Here we will describe Deep Kalman Filter [3]. Deep Kalman Filter tries to address the expressivity issue of a regular Kalman Filter. Kalman Filter is a linear dynamical system, whereas Deep Kalman Filter is non-linear. It uses deep neural networks to accomplish that.

The goal is to fit a generative model to a sequence of observations. Let's assume that the observations come from a latent state which evolves over time, observations are a noisy, non-linear function of this latent state.

Denote the sequence of observations $\overrightarrow{z} = (z_1, ..., z_T)$, with corresponding latent states $\overrightarrow{x} = (x_1, ..., x_T)$. We assume that $z_t \in \mathbb{R}^d$ and $x_t \in \mathbb{R}^s$. The generative model for the Deep Kalman filter is then given by:

$$\begin{aligned} x_1 &\sim \mathcal{N}(\mu_0; \Sigma_0) \\ x_t &\sim \mathcal{N}(G_\alpha(x_{t-1}, \Delta_t), S_\beta(x_{t-1}, \Delta_t)) \\ z_t &\sim \Pi(F_\kappa(x_t)) \end{aligned} \tag{31}$$

This means we assume that the distribution of the latent states is Normal, with a mean and covariance which are nonlinear functions of the previous latent state, and the time difference Δ_t between time $t-1$ and time t. The observations z_t are distributed according to a distribution Π (e.g. a Bernoulli distribution if the data is binary) whose parameters are a function of the corresponding latent state x_t. Specifically, the functions G_α, S_β, F_κ are assumed to be parametrized by deep neural networks. We set $\mu_0 = 0, \Sigma_0 = I_d$, so we have $\theta = \alpha, \beta, \kappa$ as parameters of the generative model. We use a diagonal covariance matrix $S_\beta(\cdot)$ and employ a log-parametrization, ensuring that the covariance matrix is positive-definite.

3.1 Stochastic Backpropagation

We make use of variational autoencoders [2] [5] to optimize a variational lower bound on the model log-likelihood. The novelty is the use of so called recognition network, a neural network which approximates the intractable posterior.
Let $p(z,x) = p_0(x)p_\theta(z|x)$ be a generative model for the set of observations z, where $p_0(x)$ is the prior on x and $p_\theta(z|x)$ is a generative model parametrized by θ. In a model such a s the one we posit, the posterior distribution $p_\theta(x|z)$ isn't usually tractable. Through so called variational principle, we posit an approximate posterior distribution $q_\phi(x|z)$ also called a recognition model. We get the following lower bound on the marginal likelihood:

$$\begin{aligned}\log p_\theta(z) &= \log \int_x \frac{q_\phi(x|z)}{q_\phi(x|z)} p_\theta(z|x)p_0(x)dx \geq \int_x q_{x|z} \log \frac{p_\theta(z|x)p_0(x)}{q_\phi(x|z)}dx \\ &= \mathbb{E}_{q_\phi(x|z)}[\log p_\theta(z|x)] - KL(q_\phi(x|z)||p_0(x)) = \mathcal{L}(z,(\theta,\phi))\end{aligned} \tag{32}$$

where inequality is by Jensen's inequality. Variational autoencoders aim to maximize the lower bound using a parametric model q_ϕ conditioned on the input, usually using a neural net to parametrize q_ϕ, such that ϕ are the parameters of the neural net.

3.2 Learning Deep Kalman Filter

Our goal is to fit parameters θ which maximize the conditional likelihood of the data given the external actions, which means we want

$$\max_\theta \log p_\theta(x_1, ..., x_T | u_1, ..., u_{T-1})$$

. Using the variaional principle, we apply the lower bound on the log-likelihood of the observations $\overrightarrow{x}$. We extend the lower bound to the temporal setting where we use the following factorization of the prior:

$$q_\phi(\overrightarrow{x}|\overrightarrow{z}) = \prod_{t=1}^{T} q(x_t|x_{t-1}, z_t, ..., z_T) \tag{33}$$

Below the algorithm for learning Deep Kalman Filters is described:

Algorithm 1 Learning Deep Kalman Filter

while $notConverged()$ **do**
  $\overrightarrow{z} \leftarrow sampleMiniBatch()$
  Perform inference and estimate likelihood:
  $\hat{x} \sim q_\phi(\overrightarrow{x}|\overrightarrow{z})$
  $\hat{z} \sim p_\theta(\overrightarrow{z}|\hat{x})$
  Compute $\nabla_\theta \mathcal{L}$ and $\nabla_\phi \mathcal{L}$
  Update θ, ϕ using ADAM
end while

4 Experimental part

We check the accuracy of predictions of the Deep Kalman Filter by using an implementation available online on a few datasets. The goal of these experiments was to measure the accuracy of the model given ground truth.

4.1 FitRec Dataset

For some of our experiments we use the FitRec Dataset [4]. It consists of user records from Endomondo, which is an applications that tracks various information when user workouts. For our experiments we use information about user latitude, longitude and altitude, all measured in degrees. We use the filtered version of the dataset called endomondoHR_proper.json.

4.2 Synthetic dataset

We use synthetic dataset that was created for the experimental part of this work, it consists of 4 samples. The first sample has 50 values of 1 and then 50 values of 100 done 10 times, the second sample has decreasing by one values from 100 to 1 done 10 times, the third sample has increasing values from 1 to 100 and then decreasing values from 100 to 1 done 5 times. The fourth sample has 5 lower halves of the circle with values from 100 to 0 done 5 times. In some experiments we add Gaussian noise to the synthetic dataset to make it noisy.

4.3 Wrocław's waterworks

We use a dataset that measures Wrocław's water consumption profile. Every sample describes the water consumption of a different day at a specific time, where the day is divided in 10 minutes intervals. The data is available under list of assignments 2 here: `https://ii.uni.wroc.pl/~lipinski/lectureADM2023.html`.

4.4 Implementation

The implementation we use is written in Python programming language and it uses the Pytorch framework. The implementation is available at the service GitHub under this address: `https://github.com/kokikwbt/deep-kalman-filter`. Specifically the model from demo_dkf_mv.pdf was used.

4.5 Results

Here we present results of our experiments. The hyperparameters of the models were chosen according to the following procedure: input_dim was set equal to the number of samples the model was trained on, z_dim, rnn_dim, trans_dim, emission_dim were set equal to five times the number of samples the model was trained on. Number of epochs was set to 200 for every experiment, annealing factor was set to 0.2 for every experiment. No hyperparameter search was done. Below every result we show average MSE, R-squared and a MAE.
In figure 1 we see results of trying to predict the longitude, latitude and altitude of the first workout of the FitRec Dataset. As we can see, the model tries to predict a shape that is the most similar to altitude, possibly because the range of values of altitude is much bigger than the longitude and latitude, so the model decides it's more important to predict such a shape.

	MSE	R-squared	MAE
Average	42.403614	-12964.300281	4.923867

In the figure 2 we see the results of trying to predict the longitude of the first three workouts of the FitRec Dataset. It is hard to gauge the quality of the prediction given a small range of longitude values.

	MSE	R-squared	MAE
Average	0.019957	-2.544409	0.108569

In the figure 3 we see the results of trying to predict the altitude of the first 5 workouts of the FitRec Dataset. The prediction is fairly good.

	MSE	R-squared	MAE
Average	32.101151	0.727443	4.416904

In the figure 4 we see the result of trying to predict synthetic data. The first one is composed of 50 values of 1 and then 50 values of 100 done 10 times. The second one is composed of decreasing by one values from 100 to 1 done 10 times. The third one is composed of increasing values from 1 to 100 and then decreasing values from 100 to 1 done 5 times. The fourth one is composed of 5 lower halves of the circle with values from 100 to 0 done 5 times. The prediction

is good.

	MSE	R-squared	MAE
Average	21.112291	0.980398	1.964865

In the figure 5 we see the results of trying to predict an Euclidean distance in degrees from the longitude and latitude of the first 5 workouts of the FitRec Dataset. It's hard to gauge the quality of results because of a small range of values of the data.

	MSE	R-squared	MAE
Average	37.476048	-51632.292738	4.059708

In the figure 6 we see the results of trying to predict longitude and latitude of the first workout from the FitRec Dataset normalized to be between 0 and 100. As we can see the prediction seems to be fairly good, the issue with model getting inspired by the data sample with the biggest range seem to be smaller here.

	MSE	R-squared	MAE
Average	10.44821	0.987938	2.417282

In the figure 7 we see the results of trying to predict latitude, longitude and altitude of the first workout of the FitRec Dataset normalized to be between 0 and 100. As we can see the prediction is much better than for the not normalized variant of this experiment.

	MSE	R-squared	MAE
Average	41.885201	0.932859	4.74778

In the figure 8 we see the results of trying to predict altitude of first 5 workouts of the FitRec Dataset normalized to be between 0 and 100. As we can see, the results are much better than for not normalized version of this experiment.

	MSE	R-squared	MAE
Average	44.636532	0.90728	4.923034

In the figure 10 we see the results of trying to predict the Haversine distance of the first 5 workouts of the FitRec Dataset from the spot with coordinates equal to 0. The prediction is not good, we hypothesize that it's because of the small range of the Haversine distance values.

	MSE	R-squared	MAE
Average	0.009493	-84081.406163	0.077461

In the figure 9 we see the results of trying to predict synthetic data with Gaussian noise. The model is able to do that.

	MSE	R-squared	MAE
Average	39.025108	0.94911	4.363547

In the figure 11 we see the results of trying to predict the Haversine distance of the first 5 workouts from the FitRec Dataset from the spot with coordinates equal to 0 normalized between 0 and 100. We get a very good prediction, consistent with the hypothesis that normalization between 0 and 100 improves model performance.

	MSE	R-squared	MAE
Average	15.982272	0.983007	2.419631

In the figure 12 we see the results of trying to predict Wrocław's waterworks water use levels for the first 7 samples of the dataset. The prediction is fine.

	MSE	R-squared	MAE
Average	12.763991	0.86544	2.763783

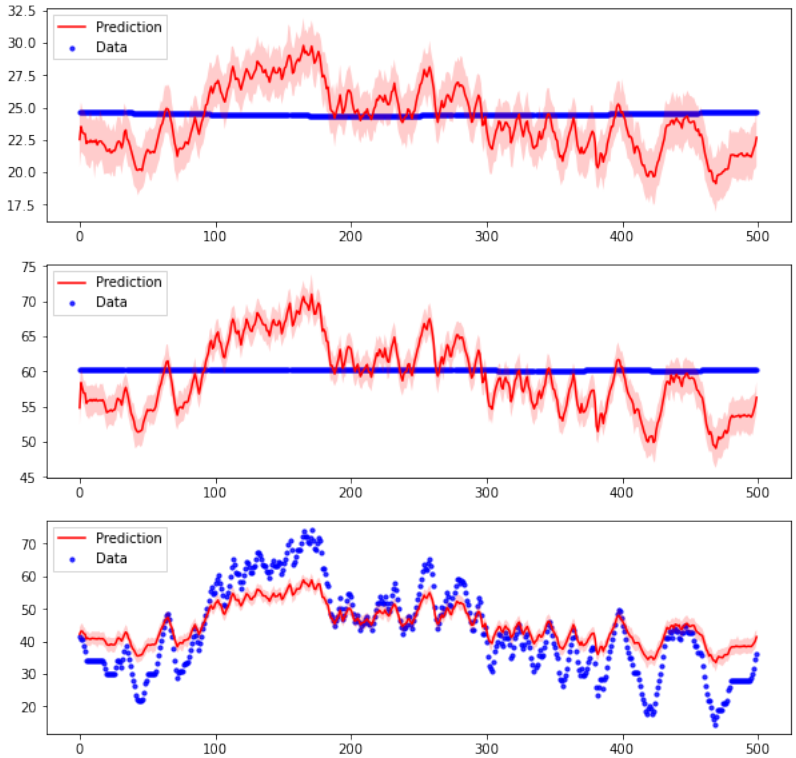

Figure 1: In figure 1 we see results of trying to predict the longitude, latitude and altitude of the first workout of the FitRec Dataset. As we can see, the model tries to predict a shape that is the most similar to altitude, possibly because the range of values of altitude is much bigger than the longitude and latitude, so the model decides it's more important to predict such a shape.

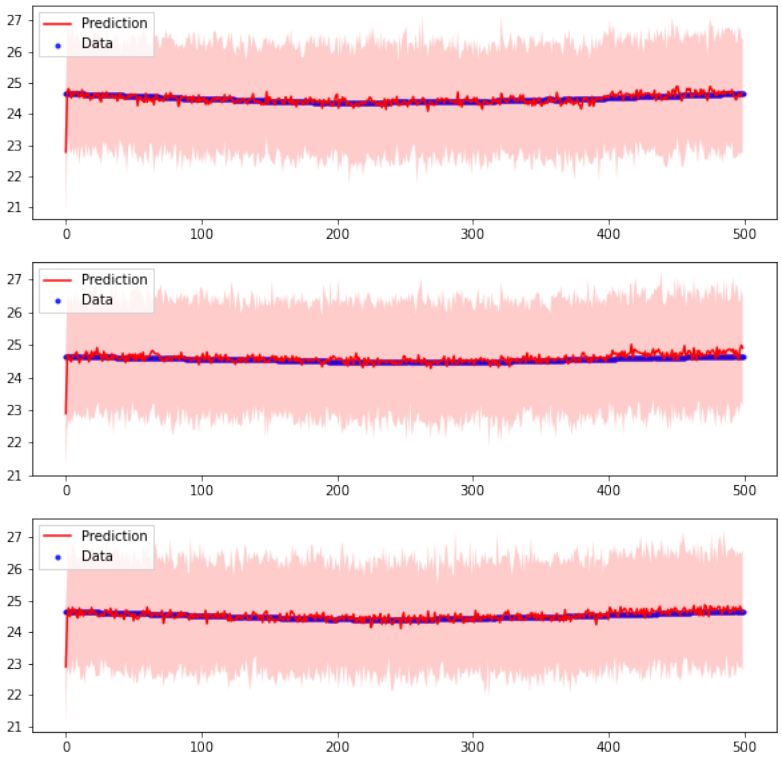

Figure 2: In the figure 2 we see the results of trying to predict the longitude of the first three workouts of the FitRec Dataset. It is hard to gauge the quality of the prediction given a small range of longitude values.

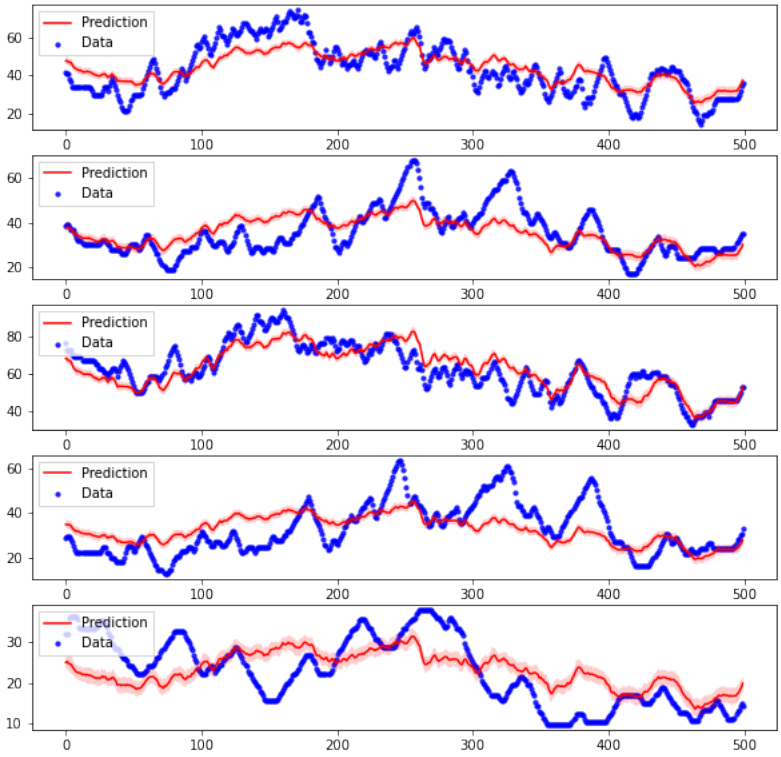

Figure 3: In the figure 3 we see the results of trying to predict the altitude of the first 5 workouts of the FitRec Dataset. The prediction is fairly good.

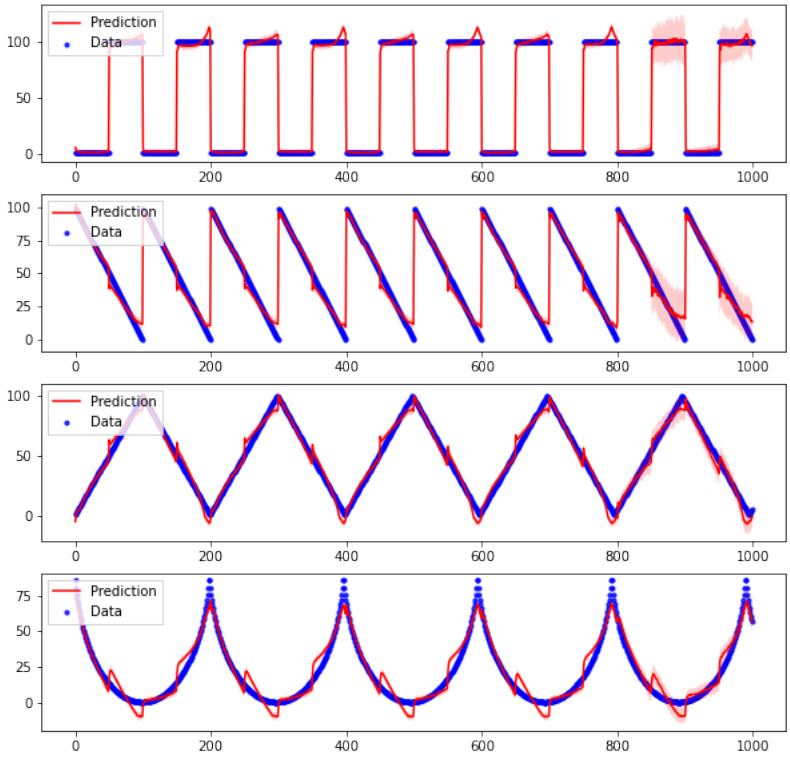

Figure 4: In the figure 4 we see the result of trying to predict synthetic data. The first one is composed of 50 values of 1 and then 50 values of 100 done 10 times. The second one is composed of decreasing by one values from 100 to 1 done 10 times. The third one is composed of increasing values from 1 to 100 and then decreasing values from 100 to 1 done 5 times. The fourth one is composed of 5 lower halves of the circle with values from 100 to 0 done 5 times. The prediction is good.

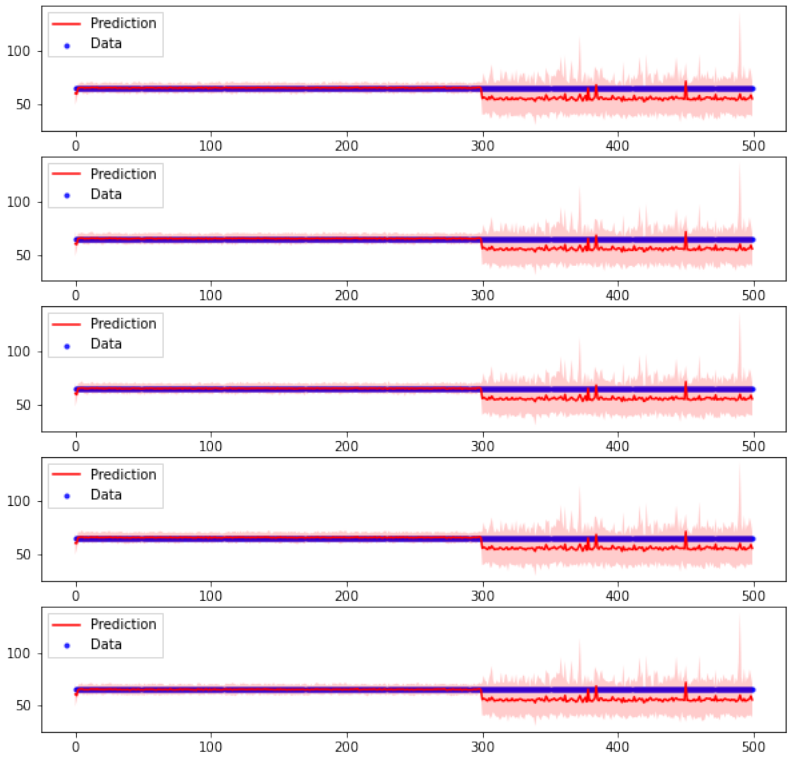

Figure 5: In the figure 5 we see the results of trying to predict an Euclidean distance in degrees from the longitude and latitude of the first 5 workouts of the FitRec Dataset. It's hard to gauge the quality of results because of a small range of values of the data.

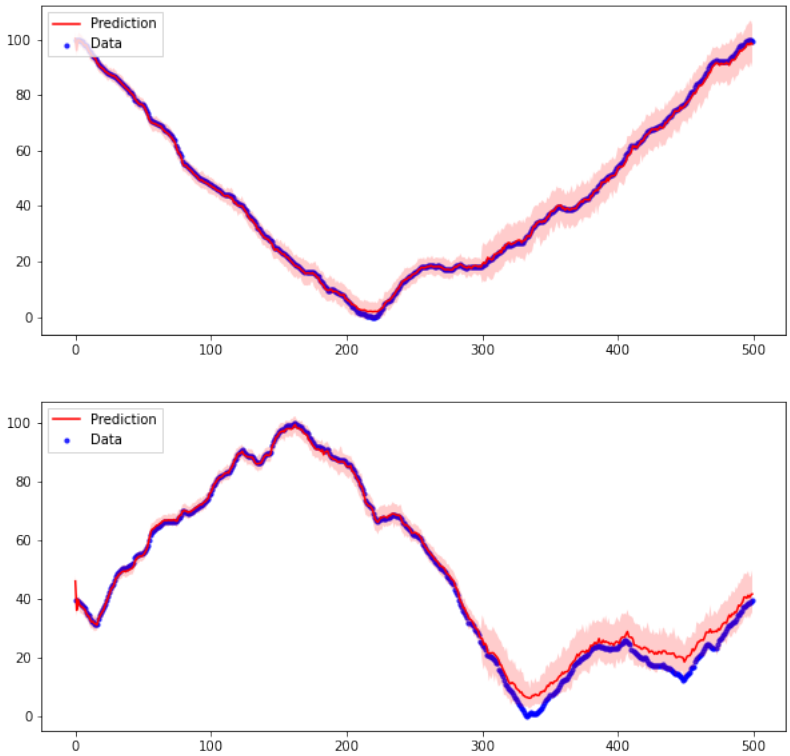

Figure 6: In the figure 6 we see the results of trying to predict longitude and latitude of the first workout from the FitRec Dataset normalized to be between 0 and 100. As we can see the prediction seems to be fairly good, the issue with model getting inspired by the data sample with the biggest range seem to be smaller here.

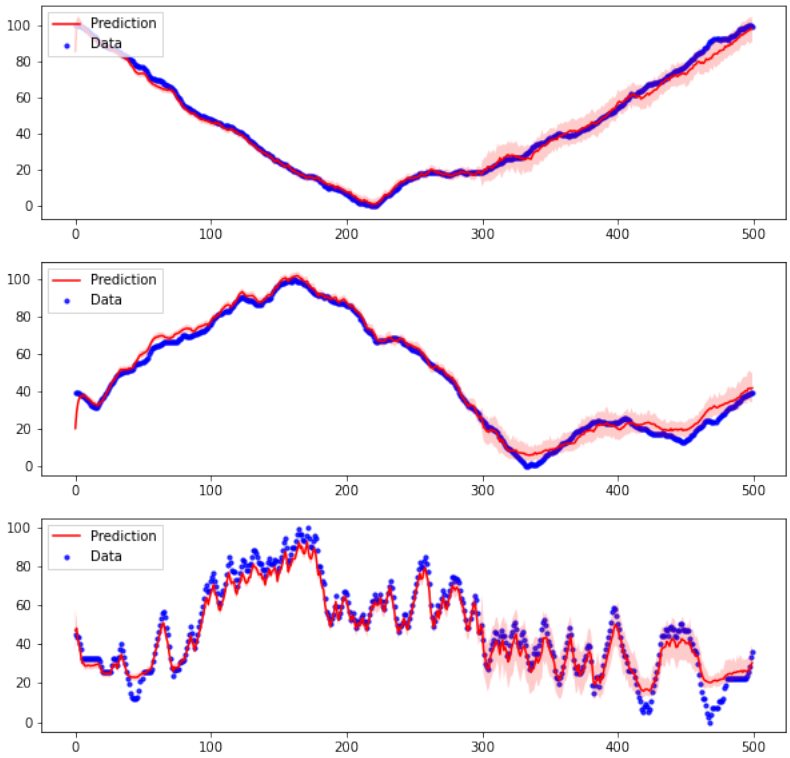

Figure 7: In the figure 7 we see the results of trying to predict latitude, longitude and altitude of the first workout of the FitRec Dataset normalized to be between 0 and 100. As we can see the prediction is much better than for the not normalized variant of this experiment.

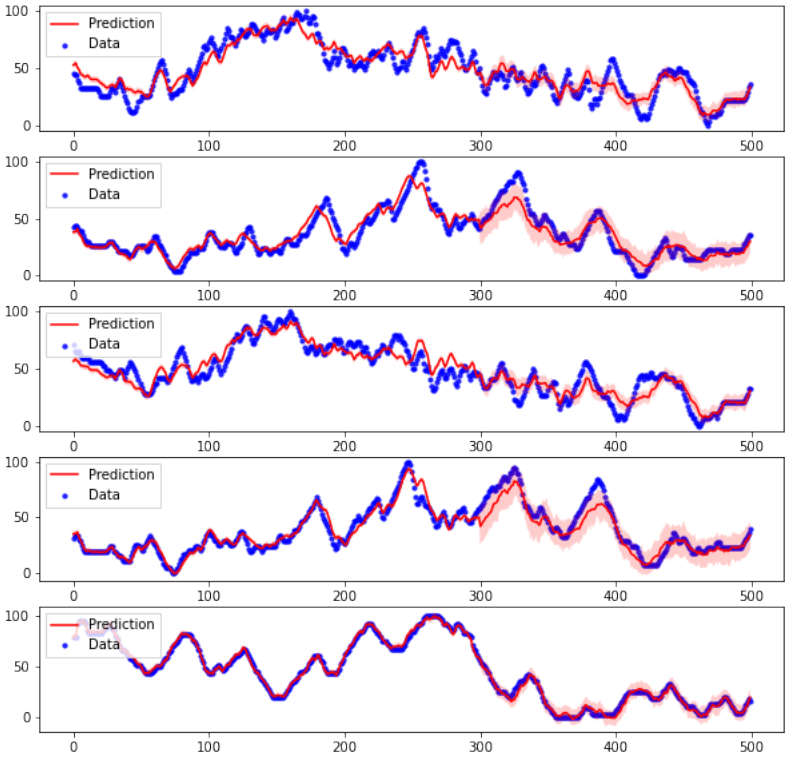

Figure 8: In the figure 8 we see the results of trying to predict altitude of first 5 workouts of the FitRec Dataset normalized to be between 0 and 100. As we can see, the results are much better than for not normalized version of this experiment.

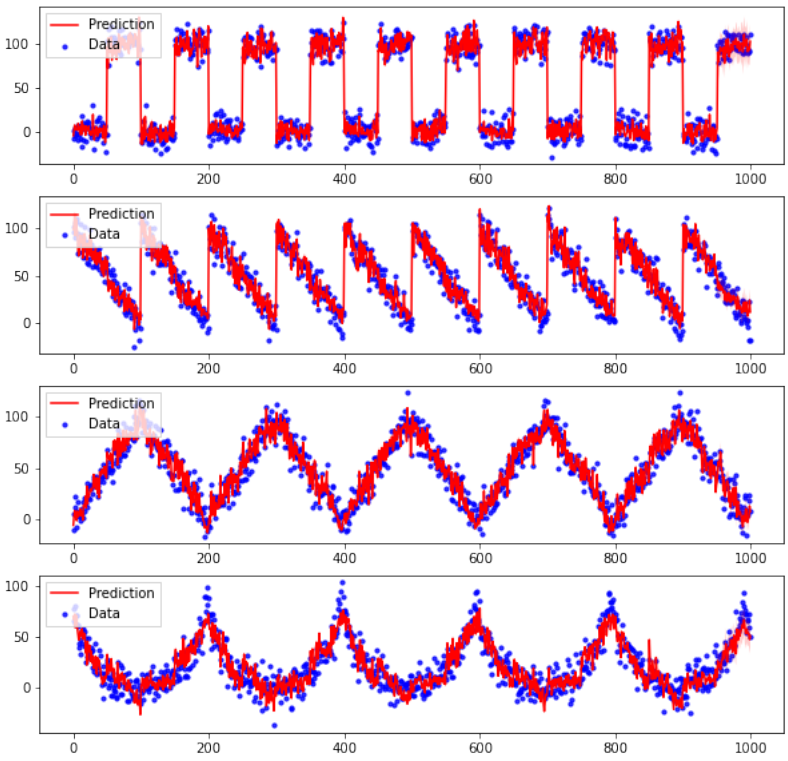

Figure 9: In the figure 9 we see the results of trying to predict synthetic data with Gaussian noise. The model is able to do that.

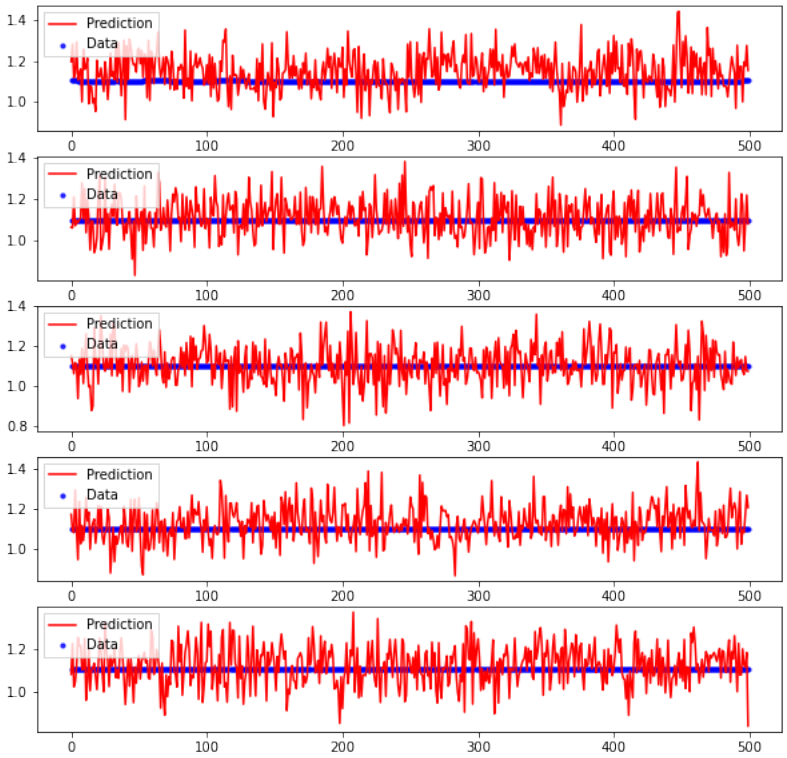

Figure 10: In the figure 10 we see the results of trying to predict the Haversine distance of the first 5 workouts of the FitRec Dataset from the spot with coordinates equal to 0. The prediction is not good, we hypothesize that it's because of the small range of the Haversine distance values.

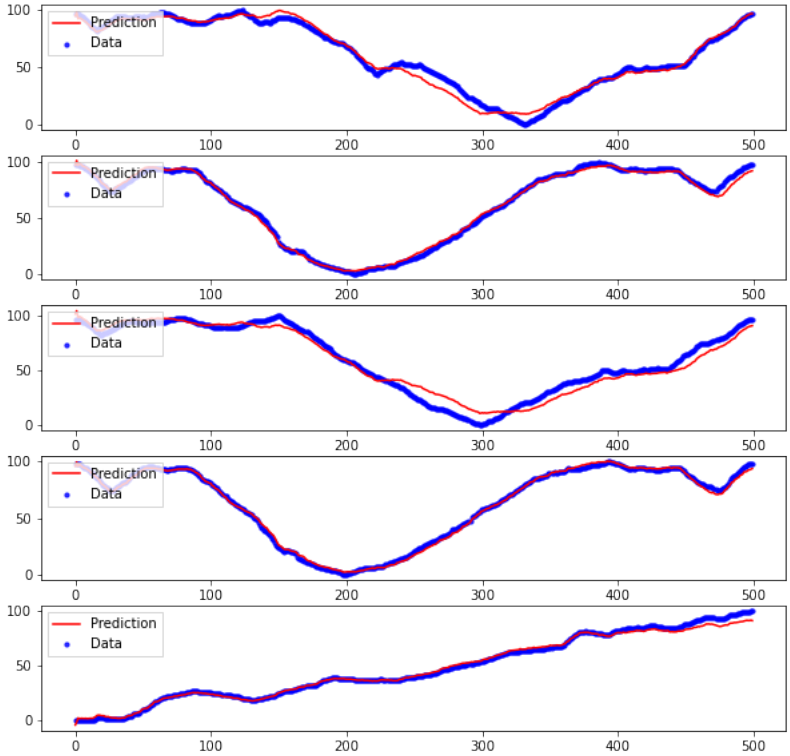

Figure 11: In the figure 11 we see the results of trying to predict the Haversine distance of the first 5 workouts from the FitRec Dataset from the spot with coordinates equal to 0 normalized between 0 and 100. We get a very good prediction, consistent with the hypothesis that normalization between 0 and 100 improves model performance.

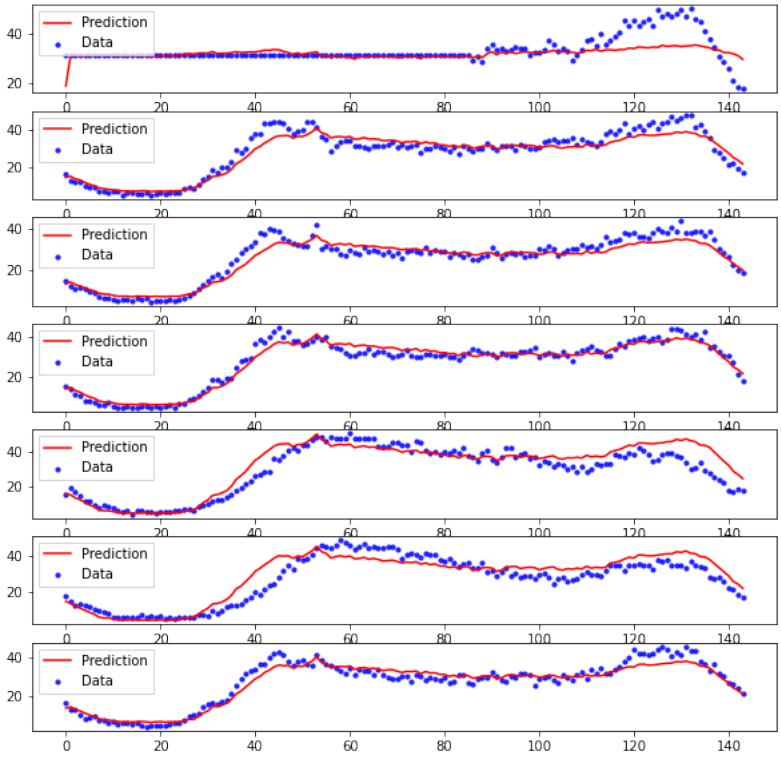

Figure 12: In the figure 12 we see the results of trying to predict Wrocław's waterworks water use levels for the first 7 samples of the dataset. The prediction is fine.

5 Sample-Normalized Deep Kalman Filter

We see that the model has problems when the range of values of samples differs significantly and when the range of values of samples is very small. For that we propose to normalize every sample to have values between 0 and 100 during training. During prediction, we denormalize the result to the original range of values. We call this modification Sample-Normalized Deep Kalman Filter. According to our experiments this increased performance of the method for the situations described above.
In the figure 13 we see the results of trying to predict the longitude for first 3 workouts of the FitRec Dataset using Sample-Normalized Deep Kalman Filter. The prediction is better than for the Deep Kalman Filter.

	MSE	R-squared	MAE
Average	0.000033	0.994926	0.004271

In the figure 14 we see the results of trying to predict the altitude for the first 5 workouts of the FitRec Dataset using Sample-Normalized Deep Kalman Filter. The prediction is better than for the Deep Kalman Filter.

	MSE	R-squared	MAE
Average	14.341215	0.909761	2.674136

In the figure 15 we see the results of trying to predict the Haversine distance for first 5 workouts of the FitRec Dataset using Sample-Normalized Deep Kalman Filter. The prediction is better than for the Deep Kalman Filter.

	MSE	R-squared	MAE
Average	5.082199e-09	0.970426	0.000042

In the figure 16 we see the results of trying to predict the longitude, latitude and altitude of the first workout of the FitRec Dataset using Sample-Normalized Deep Kalman Filter. The prediction is better than for the Deep Kalman Filter.

	MSE	R-squared	MAE
Average	0.292223	0.997089	0.204559

In the figure 17 we see the results of trying to predict Wrocław's waterworks consumption profiles for the first 7 samples of the dataset using Sample-Normalized Deep Kalman Filter. The prediction is not better than for the Deep Kalman Filter.

	MSE	R-squared	MAE
Average	24.164213	0.377955	3.103277

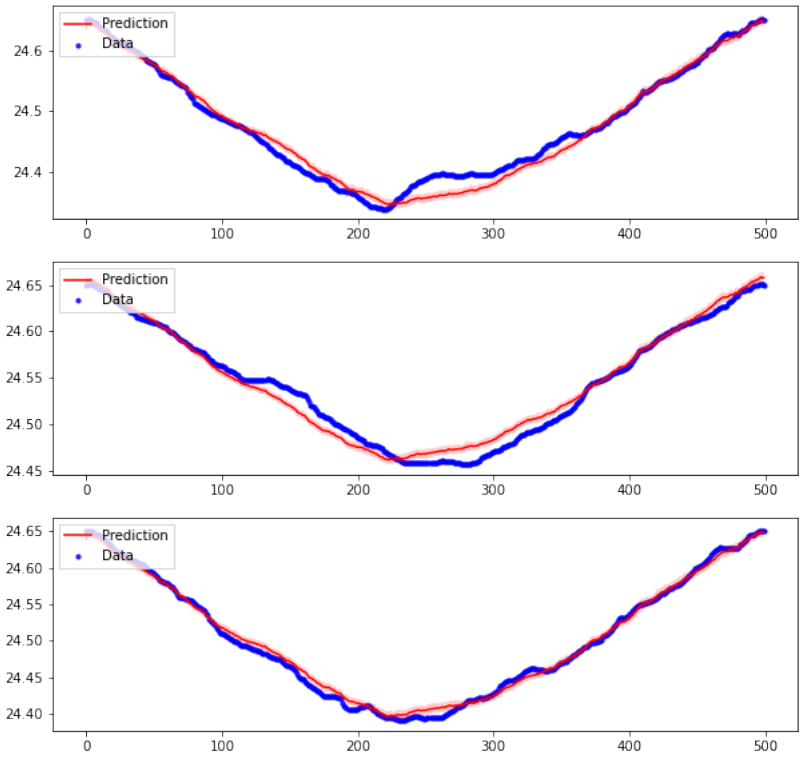

Figure 13: In the figure 13 we see the results of trying to predict the longitude for first 3 workouts of the FitRec Dataset using Sample-Normalized Deep Kalman Filter. The prediction is better than for the Deep Kalman Filter.

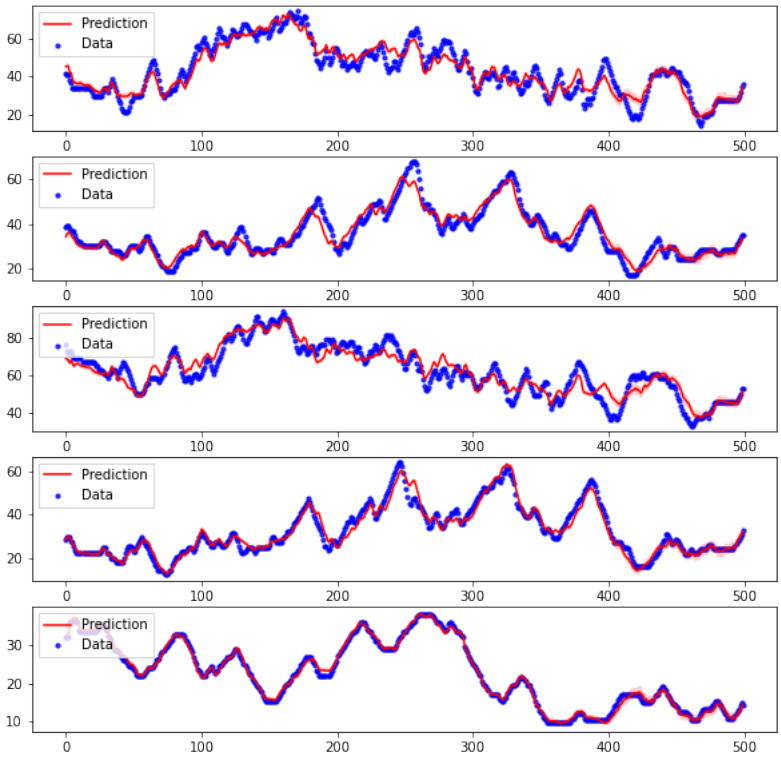

Figure 14: In the figure 14 we see the results of trying to predict the altitude for the first 5 workouts of the FitRec Dataset using Sample-Normalized Deep Kalman Filter. The prediction is better than for the Deep Kalman Filter.

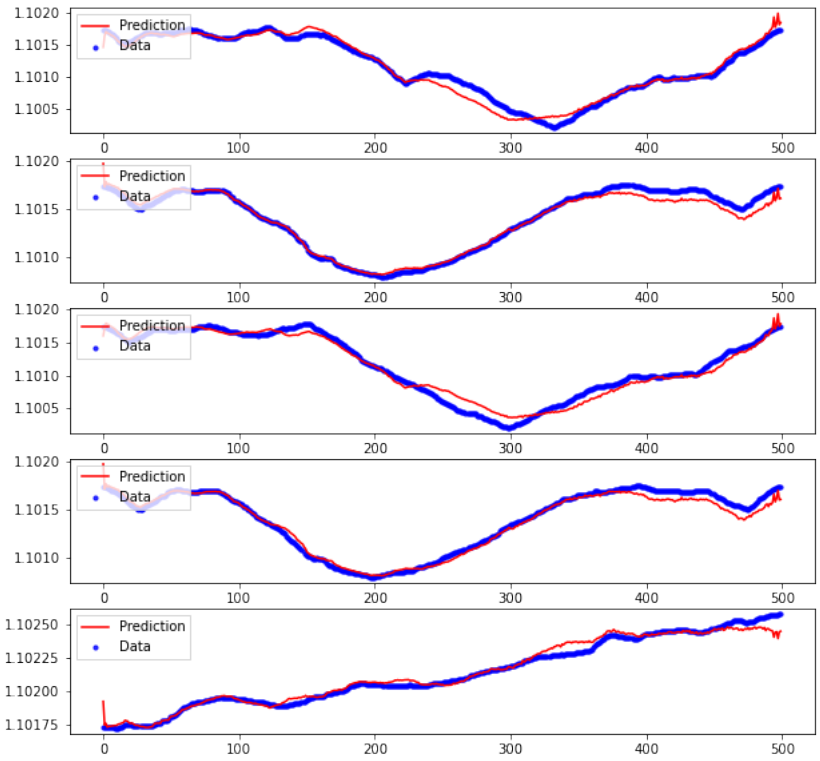

Figure 15: In the figure 15 we see the results of trying to predict the Haversine distance for first 5 workouts of the FitRec Dataset using Sample-Normalized Deep Kalman Filter. The prediction is better than for the Deep Kalman Filter.

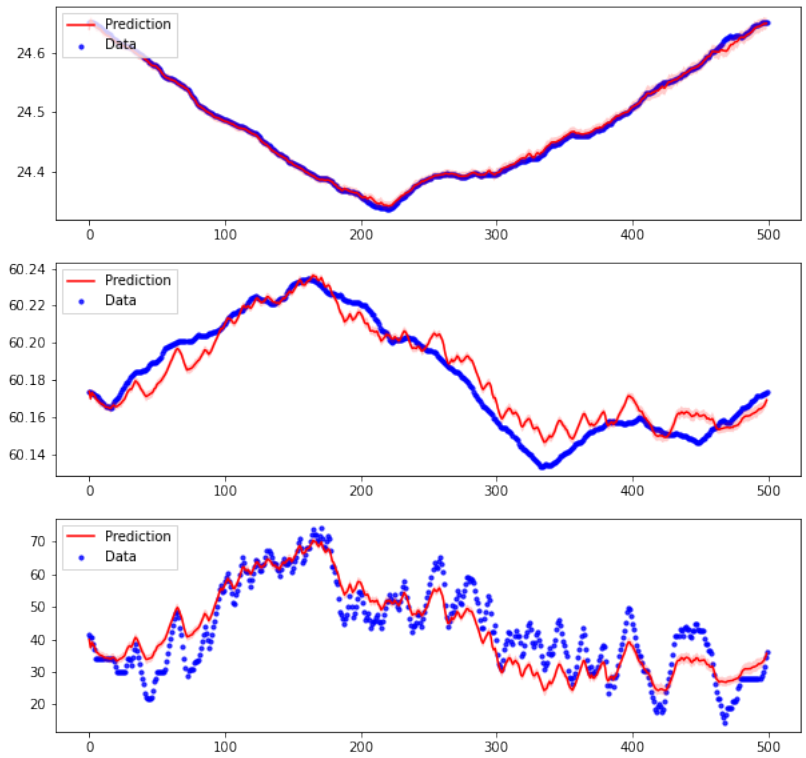

Figure 16: In the figure 16 we see the results of trying to predict the longitude, latitude and altitude of the first workout of the FitRec Dataset using Sample-Normalized Deep Kalman Filter. The prediction is better than for the Deep Kalman Filter.

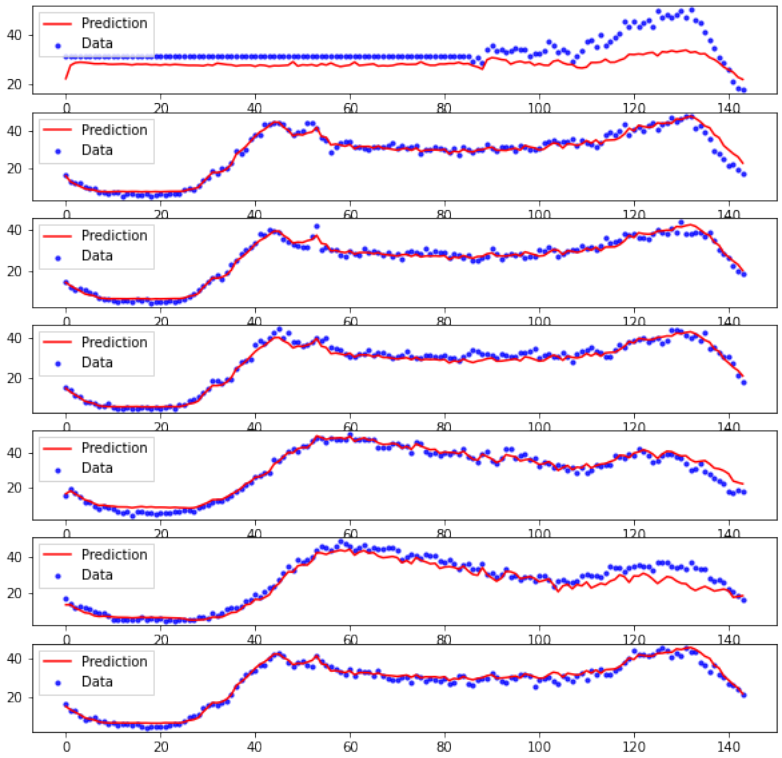

Figure 17: In the figure 17 we see the results of trying to predict Wrocław's waterworks consumption profiles for the first 7 samples of the dataset using Sample-Normalized Deep Kalman Filter. The prediction is not better than for the Deep Kalman Filter.

6 Conclusions

We described and derived the Kalman Filter algorithm, we described its modification called Deep Kalman Fiter, we checked its performance on a few datasets and we proposed the modification that we called Sample-Normalized Deep Kalaman Filter, which according to our experiments increased the accuracy of prediction of the model in certain situations.

A Source code

The code was run in a jupyter notebook environment. All experiments were done in a jupyter notebook added as an attachment called "Source_Code" and available online under this address: `https://github.com/PBPZA/Deep_Kalman_Filter_Thesis`. The Section "Oryginalny kod" consists of an original implementation avaiable online. The section "Moja praca" consists of experiments done on a Fit Rec dataset. The section "Przygotowanie danych syntetycznych" consists of preparation of a synthetic dataset and experiment on in. The section "Przesuniecie i normalizacja danych" consists of experiments done on normalized data, on Haversine of the data, on a translation of the data and on Wrocław's waterworks dataset. In the section "Sample-Normalized Deep Kalman Filter" we modify the Deep Kalman Filter to create Sample-Normalized Deep Kalman Filter and we check the accuracy of predictions of the Sample-Normalized Deep Kalman Filter experimentally.

References

[1] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[2] Diederik P Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv:1312.6114* (2013).

[3] Rahul G. Krishnan, Uri Shalit, and David Sontag. "Deep Kalman Filters". In: *arXiv:1511.05121v2* (2015).

[4] Jianmo Ni, Larry Muhlstein, and Julian McAuley. "Modeling heart rate and activity data for personalized fitness recommendation". In: *Proc. of the 2019 World Wide Web Conference (WWW'19)* (2019).

[5] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *arXiv:1401.4082* (2014).