SHIP AS A WAVE BUOY – ESTIMATING RELATIVE WAVE DIRECTION FROM IN-SERVICE SHIP MOTION MEASUREMENTS USING MACHINE LEARNING

Conference Paper · June 2019

CITATIONS
9

READS
707

2 authors, including:

Bulent Duz
Maritime Research Institute Netherlands
33 PUBLICATIONS **204** CITATIONS

SEE PROFILE

Proceedings of the ASME 2019 38th International Conference on Ocean, Offshore & Arctic Engineering
OMAE 2019
June 9-14, 2019, Glasgow, Scotland

OMAE2019-96201

SHIP AS A WAVE BUOY - ESTIMATING RELATIVE WAVE DIRECTION FROM IN-SERVICE SHIP MOTION MEASUREMENTS USING MACHINE LEARNING

Bart Mak*
Bülent Düz*
Maritime Research Institute Netherlands (MARIN)
P.O. Box 28, 6700AA, Wageningen
The Netherlands
Email: {b.mak, b.duz}@marin.nl

ABSTRACT

For operations at sea it is important to have a good estimate of the current local sea state. Often, sea state information comes from wave buoys or weather forecasts. Sometimes wave radars are used. These sources are not always available or reliable. Being able to reliably use ship motions to estimate sea state characteristics reduces the dependency on external and/or expensive sources.

In this paper, we present a method to estimate sea state characteristics from time series of 6-DOF ship motions using machine learning. The available data consists of ship motion and wave scanning radar measurements recorded for a period of two years on a frigate type vessel. The research focused on estimating the relative wave direction, since this is most difficult to estimate using traditional methods. Time series are well suited as input, since the phase differences between motion signals hold the information relevant for this case. This type of input data requires machine learning algorithms that can capture both the relation between the input channels and the time dependence. To this end, convolutional neural networks (CNN) and recurrent neural networks (RNN) are adopted in this study for multivariate time series regression.

The results show that the estimation of the relative wave direction is acceptable, assuming that the data set is large enough and covers enough sea states. Investigating the chronological properties of the data set, it turned out that this is not yet the case. The paper will include discussions on how to interpret the results and how to treat temporal data in a more general sense.

1 INTRODUCTION

Sea state information is important for operational decisions or advice on all activities at sea. Personal and environmental safety require correct interpretations of the actual conditions. Efficiency suffers from incorrect interpretations as well. In the longer run, having statistics on encountered conditions helps in designing structures and ships. This results in higher levels of safety and more realistic lifetime expectations.

Traditionally, sea state characteristics are measured using wave buoys or satelite data. These methods result in a rough estimation, limited by the distance from the buoys and the resolution of the satelite data. Occasionally the sea state is monitored using a ship mounted wave radar. Although this can give reliable results, using wave data fusion [1], it is not used very often because of the high costs.

An alternative is to use the motions of the vessel itself to infer sea state information. Methods have been developed by Iseki [2] and Tannuri [3], optimizing the sea state prediction and its associated motion spectrum with respect to the measured motion spectrum. The optimal sea state is found either by Bayesian optimization or a parameterized iterative method. Both approaches use estimated response functions as mapping from sea state to ship motions. This idea has been further optimized by Nielsen [4] to improve

*Equal contribution

results for sailing vessels and for more efficient sea state estimation [5].
In this study measured ship motion data is used to build a parameterized model for direct sea state estimation. No knowledge of the ship is used to build the model. Instead, a parameterized machine learning model is trained using time series data of the ship motions together with a measured sea state in a supervised learning setup. Being able to capture time series data in this way allows the use of local phase differences in the measured signals to be used in the sea state estimation. In this study it is shown that specific neural network structures can be used to do so and that the resulting models perform well in a wide variety of sea states.

2 MACHINE LEARNING

Suppose some unknown model generates a large amount of data from known inputs. In the case of Ship As a Wave Buoy this would be ship motions and measured sea state characteristics. Assuming the unknown model is smooth and continuous with respect to its inputs, similar input generates similar output. So, from the output, the input could be inferred. The more data is available, the more accurate this inverse mapping can be estimated.
Data driven solutions to find the inverse mapping can be split into parametric and nonparametric solutions. In nonparametric solutions, the inverse mapping is based directly on the available data. For example, the k-nearest neighbours method directly uses similarity in the observed data to find the k nearest samples, averaging the corresponding outputs for the final prediction. Parametric solutions, on the other hand, define a mapping consisting of functions and associated parameters. Using some iterative process, the parameters are optimized to produce the desired output[1]. In some solutions these parameters are referred to as weights, including neural networks.
Time series data has been used in many fields of science and engineering. As such a significant increase can be observed in the amount of collected time series data, for example by sensors [6]. Particularly, multivariate time series classification has been the focus in many practical applications such as healthcare and activity/object/action recognition [7–12]. Inspired by this popularity, many methods have been developed for time series classification both in distance-based and feature-based approaches [13]. However, multivariate time series regression has not received enough attention compared to classification. The use of CNNs in multivariate time series regression is discussed in [14]. Autoregressive CNNs were used in [15] for multivariate asynchronous time series regression with financial data. Using CNNs for conditional time series forecasting is discussed in [16] with multivariate financial data.

2.1 Neural networks

A neural network is a composite function with some specific properties. A classical neural network consists of a number of so-called layers, each consisting of a number of nodes that take a weighted sum of the outputs of the previous layer as input to an activation function. The output of a node is the output of its activation function. There is one layer that is mandatory, which is the output layer.
Activation functions have non-negative derivatives, such that an increase of its input results in the same or higher output. This way, the output of a node can be updated in a converging iterative way to better fit the desired output. This is achieved by changing the weights used in the summation and by changing the outputs of the previous layer. Since each node in a layer is connected to each node in the next layer, the change that is needed in a node is determined by the combined desire for change coming from all nodes in the previous layer. The amount of change on each layer is governed by the output error and the combination of summations and activations in the layers it preceeds. Details of this so-called backpropagation process are described in a nice way by Bernacki [17].

2.2 Designing a neural network for time series data

Traditionally, machine learning uses a fixed input scheme, such that each input has a specific meaning. Conceptually, this is most easy to understand, as each weight is trained with respect to its associated input. One way to translate time series samples into fixed meaning representations is by transforming them to frequency domain. In this study we will not do so, as this approach does not maintain phase differences for individual events. Instead, a neural network will be trained to encode the time domain data in a consistent way.
Neural network schemes exist that share weights amongst multiple inputs. One such scheme is the Convolutional Neural Network (CNN), which uses multiple sets of shared weights, called filters, to learn to respond to different patterns in the input, regardless of the position of the pattern within the input. Imagine an array of inputs in which the elements are related to their neighbours. This can be image data, in which case the elements have a spatial relation with their neighbours, or time series data, in which case the relation is temporal. Typically, filter sizes are smaller than the input dimension, so it covers only part of the total input. Therefore, filters will be repeatedly applied to different parts of the input, maintaining the relation between those different parts.
CNNs encode the input by gradually reducing the dimensions of the data. Filters are applied with a fixed step, resulting in dimensional reduction if the step size is greater than 1. On top of that, the dimensions of the data are often deliberately reduced

[1] When the objective is to find an inverse mapping, the terms input and output are inverted as well in the data driven solution.

by pooling, taking only a single statistic from a fixed number of neigbouring elements. This reduction in spatial dimensions is made up for by the number of filters that is applied, each resulting in a new representation of the input. This process can be repeated, using the output of a convolutional stage as input for the next. The combination of the resulting low dimensional convolutions can be thought of as an encoding of the original input. This encoding resembles the input in a traditional scheme, that has fixed meanings for each input.

As a final step, a few fully connected layers are applied to interpret the encoding. This whole process eliminates the need to have certain patterns at fixed locations, while they can still be used for classification or regression. This structure is sometimes referred to as encoder-classifier or encoder-regressor.

A network trained on image data will have a trained set of filters that respond to the different distinguishing features in the dataset. Similarly, using the motions of a ship exposed to various sea states, distinguishing features will be captured by the filters. For image data, recent developments have enabled CNNs to classify images at accuracy levels that supersede average human performance [18]. This is achieved by combining many convolutional layers to capture both local features and global combinations of theses features in a very rich way. The ship as a wave buoy problem is more focused though and requires regression instead of classification, leading to different design choices. These design choices are described in the next few sections.

2.3 Temporal Convolutions

The input to the neural network is noted as $X \in \Re^{6\times d\times 1}$, where 6 refers to the 6-DOF motions[2] and d is the length of the motion signals. Consider now that the first layer has n^1 filters $W^{1,g} \in \Re^{w^1\times h^1\times 1}$ for $g = 1,\ldots,n^1$, where all the superscripts of value 1 refer to the first layer apart from the number of channels. In order to capture local temporal information, the size of the filters $\left(w^1,h^1\right)$ is restricted to small values. Note that the weight matrix has only one channel since the number channels in the input X is also one. Each filter $W^{1,g}$ convolves the input and subsequently a bias term $b^{1,g}$ is added:

$$Z^{1,g}_{i,j} = \left(\mathrm{X} * W^{1,g}\right)_{i,j} = \sum_{w^1}\sum_{h^1} \mathrm{X}_{i+w^1,j+h^1} W^{1,g}_{w^1,h^1} + b^{1,g} \quad (1)$$

[2]No explicit selection of motion channels is made, as the training mechanism of the neural network will ignore any non-contributing channel automatically.

In Eq. 1, i and j refer to the position of the element in the tensor, $*$ denotes the convolution operator and $Z^{1,g} \in \Re^{a^1\times c^1\times 1}$ where

$$\begin{aligned} a^1 &= \frac{6+2p-w^1}{s}+1, \\ c^1 &= \frac{d+2p-h^1}{s}+1, \end{aligned} \quad (2)$$

in which p refers to *padding* and s refers to *stride*. The output feature map from the first layer is then calculated by passing Z^1 through a non-linearity $f(\cdot)$ to obtain $Y^1 = f\left(Z^1\right)$ where $Y^1 \in \Re^{a^1\times c^1\times n^1}$. In this work, hyperbolic tangent (tanh) function was used as the non-linearity.

In each subsequent layer $l = 2,\ldots,L$, each filter $W^{l,g} \in \Re^{w^l\times h^l\times n^{l-1}}$ for $g = 1,\ldots,n^l$ convolves the output feature map from the previous layer, $Y^{l-1} \in \Re^{a^{l-1}\times c^{l-1}\times n^{l-1}}$. After adding the bias term $b^{l,g}$, the following result is obtained to pass through the non-linearity

$$Z^{l,g}_{i,j} = \left(Y^{l-1} * W^{l,g}\right)_{i,j} = \sum_{w^l}\sum_{h^l}\sum_{k=1}^{n^{l-1}} Y^{l-1}_{i+w^l,j+h^l,k} W^{l,g}_{w^l,h^l,k} + b^{l,g} \quad (3)$$

where $Z^l \in \Re^{a^l\times c^l\times n^l}$, and a^l and c^l can be found in a similar fashion to Eq. 2. The output Z^l is then passed through the non-linearity resulting in $Y^l = f\left(Z^l\right)$.

2.4 Long Short-Term Memory RNNs

Recurrent Neural Networks (RNN) are sometimes used for time-varying inputs and outputs since they capture implicit compositional representations in the time domain. The connections between the layers of an RNN model a temporal dynamic behavior for time-varying data. However, traditional RNN models suffer from a significant issue known as the "vanishing gradient" effect: it becomes difficult to backpropagate error signals through a long-term temporal interval.

Long Short-Term Memory RNNs (LSTM) [19] circumvents this issue by employing gating functions in their designs, see Fig. 1. As such, they are able to remember information for long periods of time. LSTM unit keeps the existing memory $c_t \in \Re^N$ at time t with N hidden units. Note that we here adopt the notation in [20]. Given the inputs x_t, h_{t-1}, c_{t-1} at time t, the outputs h_t and c_t are

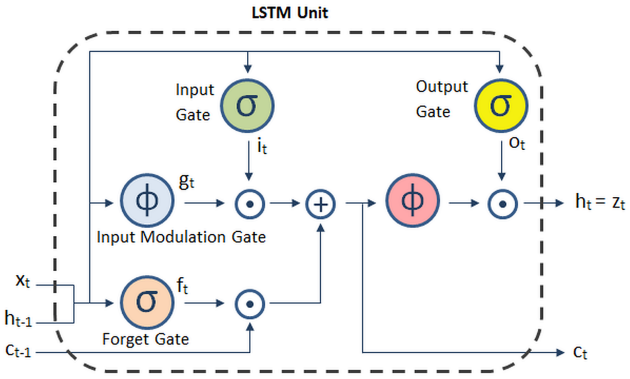


FIGURE 1: A diagram of a basic LSTM memory cell.

updated using the following equations

$$\begin{aligned} i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + b_i\right) \\ f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right) \\ o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + b_o\right) \\ g_t &= \sigma\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \tanh\left(c_t\right) \end{aligned} \tag{4}$$

where $\odot$ denotes element-wise multiplication of two vectors, and $\sigma(\cdot)$ denotes the logistic sigmoid function. The LSTM includes a hidden unit $h_t \in \Re^N$, input gate $i_t \in \Re^N$, forget gate $f_t \in \Re^N$, output gate $o_t \in \Re^N$, input modulation gate $g_t \in \Re^N$, and memory cell $c_t \in \Re^N$. The memory cell unit c_t is a sum of two terms: the previous memory cell unit c_{t-1} which is modulated by f_t, and g_t, a function of the current input and previous hidden state, modulated by the input gate i_t. Because i_t and f_t are sigmoidal, their values lie within the range $[0,1]$, and i_t and f_t can be thought of as knobs that the LSTM learns to selectively forget its previous memory or consider its current input. Likewise, the output gate o_t learns how much of the memory cell to transfer to the hidden state. These additional cells seem to enable the LSTM to learn complex and long-term temporal dynamics for a wide variety of sequence learning and prediction tasks.

2.5 Squeeze-and-Excitation Blocks

The squeeze-and-excitation block was proposed in [21] as a computational unit for any given transformation $F_{tr} : X \to U$, $X \in \Re^{W' \times H' \times C'}$, $U \in \Re^{W \times H \times C}$, where F_{tr} can be considered as a standard convolutional operator. Note that for the sake of consistency, we switch to the notation used in the original paper. The outputs of F_{tr} can be written as $U = [u_1, u_2, \ldots, u_C]$ where

$$u_c = v_c * X = \sum_{s=1}^{C'} v_c^s * x^s. \tag{5}$$

In Eq. 5, $*$ denotes convolution, $v_c = \left[v_c^1, v_c^2, \ldots, v_c^{C'}\right]$ and $X = \left[x^1, x^2, \ldots, x^{C'}\right]$ (as in the original work, bias terms are omitted to simplify the notation). v_c^s denotes a 2D spatial kernel and a single channel of v_c acting on the corresponding channel of X. v_c implicitly embeds the channel dependencies which are entangled with the spatial encoding learned by the filters. The main purpose of the squeeze-and-excitation blocks is to increase the sensitivity of the neural network to informative features and suppress less useful ones. This is achieved in two steps, *squeeze* and *excitation*. Figure 2 shows a single squeeze-and-excitation building block.

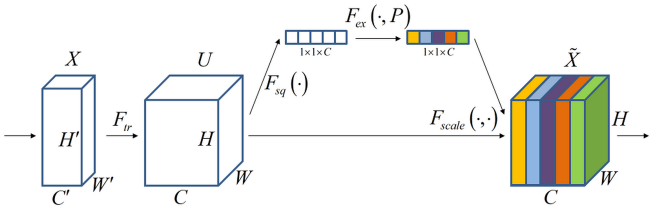


FIGURE 2: A Squeeze-and-Excitation block.

During the *squeeze* step, channel-wise statistics are generated by using global average pooling. The transformation output U is shrunk through spatial dimension $W \times H$ to calculate the channel-wise statistics, $z \in \Re^C$. The c-th element of z is calculated via

$$z_c = F_{sq}(u_c) = \frac{1}{W \times H} \sum_{i=1}^{W} \sum_{j=1}^{H} u_c(i,j). \tag{6}$$

During the *excitation* step the aim is to fully capture the channel-wise dependencies. This is achieved by employing a simple gating mechanism

$$s = F_{ex}(z, P) = a_2\left(g(z,P)\right) = a_2\left(P_2 a_1\left(P_1 z\right)\right). \tag{7}$$

The gating mechanism includes two fully connected layers around the non-linearity, a dimensionality-reduction layer with parameters $P_1 \in \Re^{\frac{C}{r} \times C}$ and the activation function a_1, and a dimensionality-increasing layer with parameters $P_2 \in \Re^{C \times \frac{C}{r}}$ and the activation function a_2. In [21], the ReLU function [22] was

used for a_1 and the sigmoid function was used for a_2. In this work, the tanh function was used for both activations. The reduction ratio r was set to 16 as in the original work. The final output of the block is calculated by rescaling the transformation output U as follows

$$\tilde{x}_c = F_{scale}(u_c, s_c) = s_c \cdot u_c \tag{8}$$

where $\tilde{X} = [\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_C]$ and $F_{scale}(u_c, s_c)$ refers to channel-wise multiplication between the feature map $u_c \in \Re^{W \times H}$ and the scalar s_c.

2.6 Architecture of the neural networks

Several neural networks were used in this study and their performances compared. Each neural network will be described below.

2.6.1 CNN-REG The convolutional neural network for regression (CNN-REG) was considered as a base network for comparison, see Fig. 3 for a diagram of the network. It comprises two convolution layers, each of which has the tanh activation function and is followed by a max pooling layer. After the output of the second max pooling layer is flattened, it is connected to a dense layer with the tanh activation followed by a dropout. The output of the dropout is sent to the output layer with tanh activation. Both convolution layers have 48 filters. The filter sizes in the first

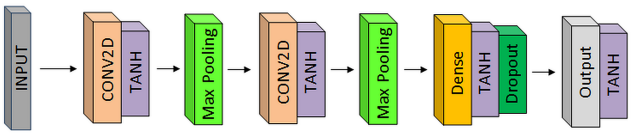


FIGURE 3: The architecture of the CNN-REG network.

two layers are 6×15 and 1×9, respectively, and the pooling size is 3. The dense layer has 30 nodes, and the dropout rate is 0.25.

2.6.2 Multivariate LSTM-CNN (MLSTM-CNN) MLSTM-CNN network was mainly adopted from [23]. It consists of a fully convolutional block and an LSTM block as shown in Fig. 4. The fully convolutional block contains three temporal convolutional blocks, used as a feature extractor. Each convolutional block contains a convolutional layer, and is succeeded by a batch normalization and tanh activation. In addition, the first two convolutional blocks conclude with a squeeze and excite block (see section 2.5). The added squeeze and excite block enhances the performance of the neural network. The final temporal convolutional block is followed by a global average pooling layer.

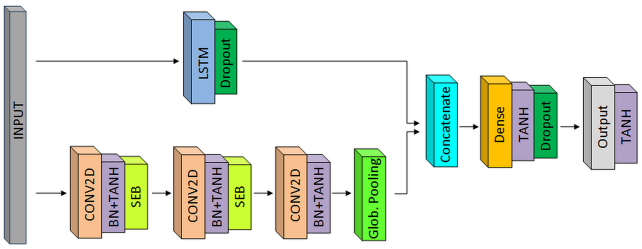


FIGURE 4: The architecture of the MLSTM-CNN network.

On the other hand, the multivariate time series input is passed through the LSTM block (see section 2.4) consisting of an LSTM layer followed by a dropout. The outputs from the fully convolutional block and the LSTM block are then concatenated and passed through a dense layer. The dense layer is followed by tanh activation and dropout. Finally the output layer produces the result of the neural network with tanh activation.
The depth of the LSTM layer is 8. The first convolution layer has 16 filters of size 6×11. The second and third convolution layers have 32 filters and sizes of 1×6 and 1×3, respectively. The dense layer has 8 nodes and the dropout rate is 0.1.

2.6.3 Sliding Puzzle Network As we have seen in section 2.2, CNNs can be used for datasets that have many distinct features. The benefit is that it allows very good predictions for very large datasets, but it tends to overfit when applied to smaller datasets, like our ship motion dataset. Even though regularization mitigates this problem to some extent, it cannot be fully removed and at the same time the expressive power of the network is reduced. Since the ship motion data is quite different from image data, specific solutions may be available to solve this problem.
The final encoding in the CNNs encodes the progression of local features that are encoded in the first convolutional layer. Suppose the local features are all we need, then the final encoding acts like a sample signature which can be remembered by the network. In order to prevent that, we have constructed a network that only does this initial encoding, without any information on the order of events. The idea is to encode single patches with a length equal to the filter size and step size. This leads to separate non-overlapping filter results. Expanding on this, the input to the network can be constructed by concatenation of single patches chosen at random from the original input sample. These patches are allowed to overlap and can be placed in any order, since the network does not assume any relation between consecutive patches. For each training epoch, the input samples in the training set can be regenerated, which is the final step to prevent sample signature recognition. This approach resembles a sliding puzzle with a picture to be solved. Even though it may be shuffled, it is still clear what the picture is about. The precise location

in which the tiles are seen does not matter.
One problem still needs to be solved. As explained in section 2.2, a position independent encoding is required for the neural network to work. Since we now have a number of activations per filter, which are still position dependent with respect to the original sample, we have to take one extra step. For each filter we take the minimum, maximum and mean of the activations of the different patches, removing the positional dependency. The result is an encoding purely based on the initial convolutions, which are applied to random patches of the original input sample. This architecture is shown in Fig. 5.

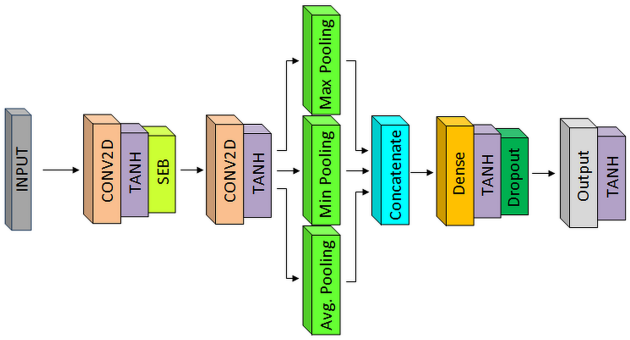


FIGURE 5: The sliding puzzle network. Note that two convolutional layers are used instead of one. The first convolutional layer only works in the time direction, the second combines the 6 channels. The receptive field of this combination is the same as the receptive field of a single convolution with combined dimensions. The number of trainable weights is less though, which leads to better generalization. Also, a squeeze-and-excite block is used for better stability (see section 2.5).

The network uses 64 filters in the temporal direction, with size 1×25, followed by 128 filters that combine the 6 channels, with size 6×1. The dense layer has 30 nodes and no dropout is used. Even though training of this network may give decent results, we can take further advantage of the random nature of the input to the network. If we construct a number of inputs from one sample, we can use the average prediction as final result, reducing any noise in the network output that is not related to the characteristics of the sample itself. Also, the standard deviation of the predictions can be used as a measure for the uncertainty of the prediction[3].

[3]This is the uncertainty in the predictions the network will give for the particular sample and not the uncertainty for any prediction the network would make. Also, this number is not related to the actual error the network makes in its predictions.

3 TREATMENT OF THE IN-SERVICE MEASUREMENT DATA

The in-service measurement data was collected for a period of two years on board a frigate-type naval vessel. The 6-DOF motions at the center of gravity of the ship were measured using accelerometers, and the wave characteristics were measured via a wave scanning radar mounted on the ship. The data was saved as 30 minutes long pieces. The sampling rate of the ship motions was 20 Hz. The sampling rate of the relative wave direction and ship speed was 1 Hz. Hs and Tp were recorded as two-minutes averages. The in-service measurement data required preprocessing and was unbalanced. Furthermore, the ground truth, i.e. wave characteristics, contained contributions from wind and swell seas and possibly other sources such as currents.
Data preparation and cleaning were performed in four steps:

1. In cases where the measurement device recorded an error code
   It was observed that measurement devices occasionally recorded an error code for one or more of signals of interest. In this case, that piece of data was completely ignored.
2. Outlier detection
   Outliers were detected in the relative wave direction values using the moving mean and moving median approaches, and were replaced by the moving mean values. Figure 6 illustrates an example for outlier detection where the blue colored data points were identified as outliers.

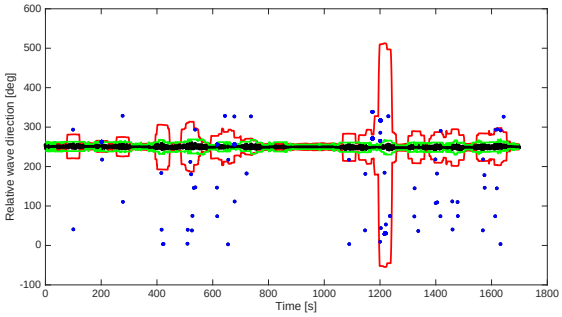


FIGURE 6: Outlier detection using the moving mean and moving median approaches. Both black and blue colored dots represent the measured values. Blue colored data points were identified as outliers.

3. Correction of large shifts ($\approx 360°$) in values of relative wave direction for the following or close to following seas
   For the following or close to following seas, large shifts of approximately 360° were sometimes observed in the relative wave direction data, because the direction was defined on the 0 deg-360 deg interval. These shifts were corrected in order

to have smooth behavior in data. Figure 7 shows an example.

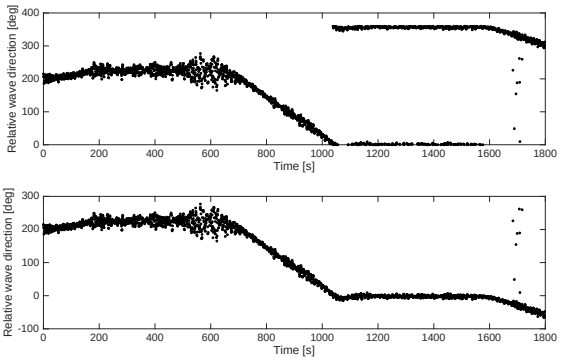


FIGURE 7: Correction of large shifts ($\approx 360^\circ$) in values of relative wave direction. Top plot shows the original data points where large shifts can be observe between approximately $t = 1000s$ and $t = 1800s$. Bottom plot shows the corrected data points.

4. Extracting chunks of data where the behavior of ground truth, ship speed and heading was acceptable according to a selected set of criteria
   During this step, the final extraction of chunks of data was carried out. The aim was to extract chunks of data that were sufficiently long for which the moving mean and maximum variation of Hs, Tp, Dp and Vs were within acceptable limits. Figure 8 demonstrates an example for this step. The top plot shows three chunks of data that were initially extracted from Dp. The first chunk covers the range between $t = 0s$ and $t = 140s$, the second between $t = 420s$ and $t = 1070s$, and the third between $t = 1180s$ and $t = 1700s$. After the initial extraction, the behavior of ship speed, Hs and Tp were analyzed during the three chunks. The middle plot shows that ship speed varied considerably during the first chunk which therefore was ignored. The second chunk was shortened and further split into two chunks with two different mean values, and the third chunk was shortened since during the beginning ship speed varied considerably. The bottom plot shows that the moving mean of Hs and Tp was almost constant during the entire time trace. As a result, from the time trace shown in Fig. 8, the three chunks in the middle plot were extracted after the entire data treatment process for this time trace.

The input to the neural networks was a multivariate time series $X \in \Re^{6\times d\times 1}$ where d was chosen to be 200, and the resulting signal duration was 2.5 minutes, resulting in an effective time step of 0.75s. In total, 20120 samples were extracted from the in-

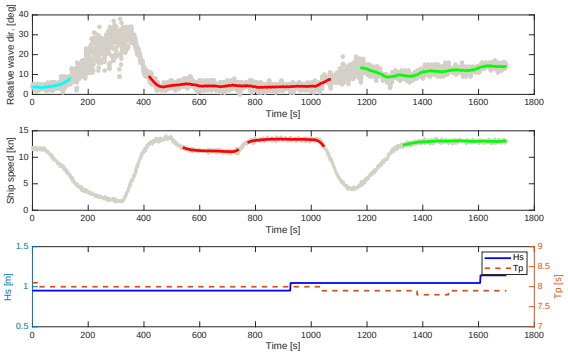


FIGURE 8: Extraction of useful chunks of data from a time trace.

service measurement data. Figures 9 and 10 illustrate the mean and standard deviation of each sample for 6-DOF ship motions, while Fig. 11 illustrates the Hs, Tp, Dp and Vs for each sample. Note that the samples were chronologically ordered in the sense that the sample with the index value of n was collected later than the sample with index $n-1$, and earlier than the sample with index $n+1$. The time distance between consecutive samples may vary. Figure 12 shows the histogram of Dp, where the unbalanced nature of the data can be clearly observed.

4 RESULTS

Three networks, CNN-REG from section 2.6.1, MLSTM-CNN from section 2.6.2, and Sliding Puzzle from section 2.6.3 have been trained with the measured data. Splitting into training and validation sets has been done both before and after shuffling the data.

4.1 Training methodology

We have trained our networks with stochastic gradient descent (SGD) [24] utilizing the Keras library [25] with the TensorFlow-GPU backend [26] running on a NVidia GeForce GTX 980 with 2048 CUDA cores and 4GB memory. The processor was the Intel Xeon CPU E5-1630 v4 with 3.7 GHz. In SGD, we used a momentum of 0.9 with a decay of 0. The loss function was the mean squared error (MSE).

For each neural network, a hyperparameter tuning study was carried out to optimize the performance and efficiency of the networks. The hyperparameters include both network architecture parameters and training parameters, such as learning rates. Since the architecture of the networks were different, the hyperparameter tuning study varied between the networks in required effort. Training was done with 80% of the data, leaving 20% of the data for validation. The split into the training and validation set was done chronologically, selecting a consecutive block of samples for validation and training on the rest, and shuffled, in which

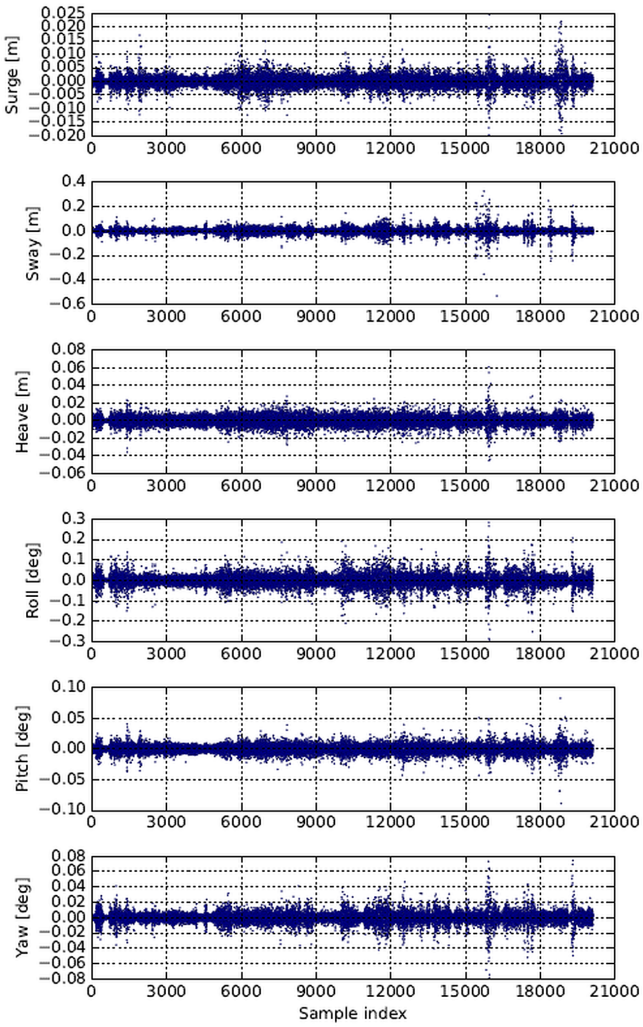


FIGURE 9: Mean of the ship motions of each sample from the in-service measurement data.

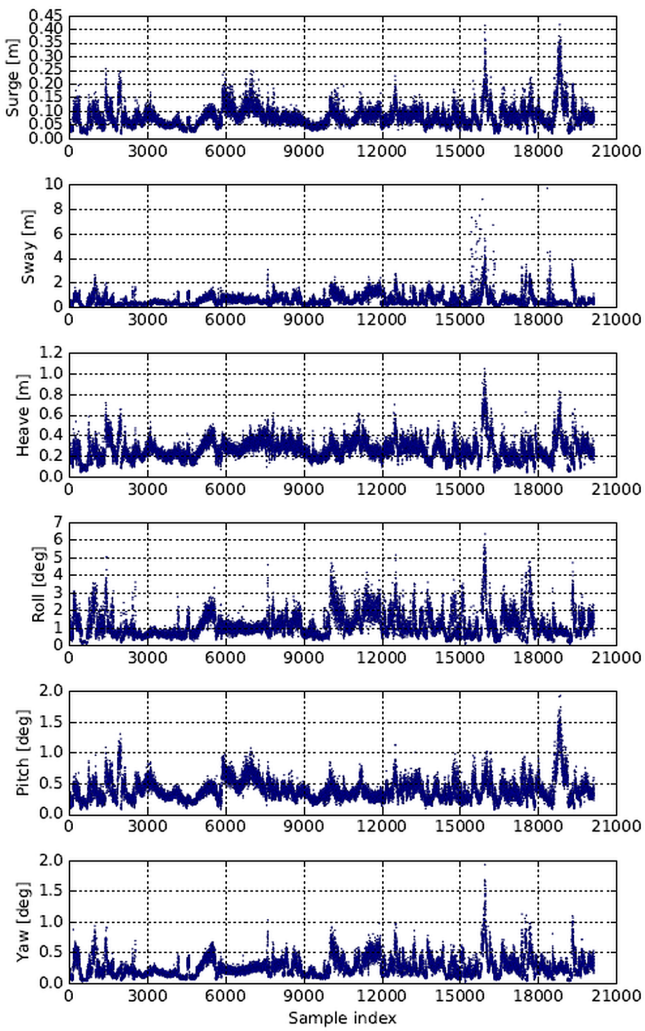


FIGURE 10: Standard deviation of the ship motions of each sample from the in-service measurement data.

case the data is shuffled before splitting.

4.2 Evaluation metrics

The results are reported as the 95% error level. This means that 95% of the predictions errors is less than or equal to this value.

4.3 Results from the measured data

Figure 13 shows the convergence of the training and validation losses from the three neural networks. These losses are presented as the average and spread of 5-fold cross validation. The histograms of Fig. 14 show how the validation errors are distributed and figure 15 shows how the predictions are distributed over the full domain. Table 1 shows a comparison of the three networks applied to both datasets.

The histograms in Fig. 14 clearly show better performance for the shuffled dataset. Also, the performance differences between the networks is more pronounced for this dataset. Errors are distributed symmetrically. Fig. 13 shows the same performance differences in terms of validation loss, but adds to that differences in performance for the 5 parts of the 5-fold cross validation.

5 DISCUSSION

The results presented in section 4 show that the machine learning approach works, that there are some noticeable differences in performance between the three networks and that the way the data is used to train and evaluate the performance has quite some impact on the results.

To fully evaluate the results, these aspects will be discussed in

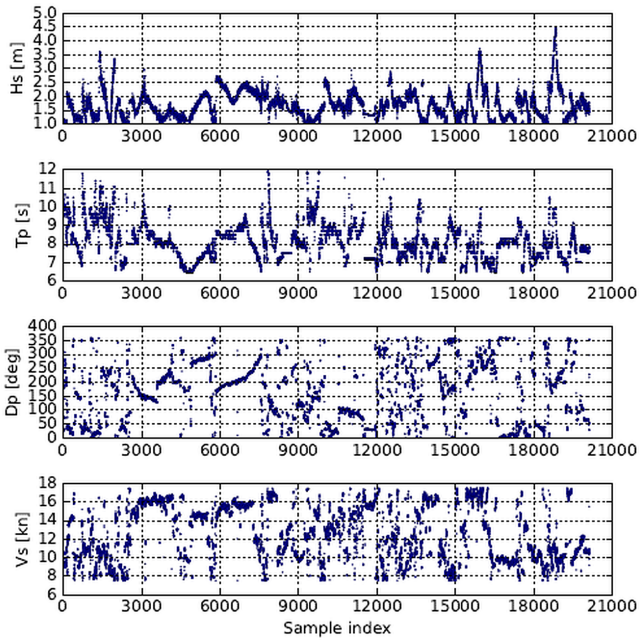


FIGURE 11: Hs, Tp, Dp and Vs of each sample from the in-service measurement data.

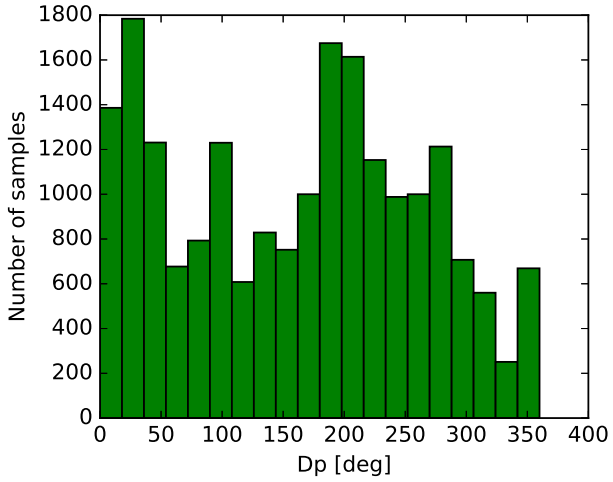


FIGURE 12: Histogram of Dp from the in-service measurement data.

the next few sections, together with a comparison to state of the art methods for sea state estimation.

TABLE 1: Results from the three neural networks applied to the shuffled and chronological data.

Train./Valid. data	95% level test (°)	Mean of error (°)	Std of error (°)	Network name
Shuffled	31.98	0.24	17.31	CNN-REG
	27.52	0.13	14.77	MLSTM-CNN
	21.67	0.40	13.07	Sliding puzzle
Chronological	40.77	-0.72	21.38	CNN-REG
	38.20	0.56	20.16	MLSTM-CNN
	38.29	0.54	19.36	Sliding puzzle

5.1 Comparison to spectral sea state estimation from ship motions

The model driven spectral methods use known response functions to map between sea state and ship motions. Although this approach works well for simulated data, it has some difficulty to find the relative wave direction for measured data. Our best methods seem to compare well to the sea trial results reported in [27] and [5].

The main difference is the need for data. While the model driven method could work for any ship in any condition, the performance of our data driven method depends heavily on the availability of good quality measured data. However, this dependency may change in the future. Neural networks have the capability of generalization, meaning that, for this problem, data coming from multiple different ships can be used to train a general network that can be used with ships that were not in the training set. This idea is explored in [28] for simulated data.

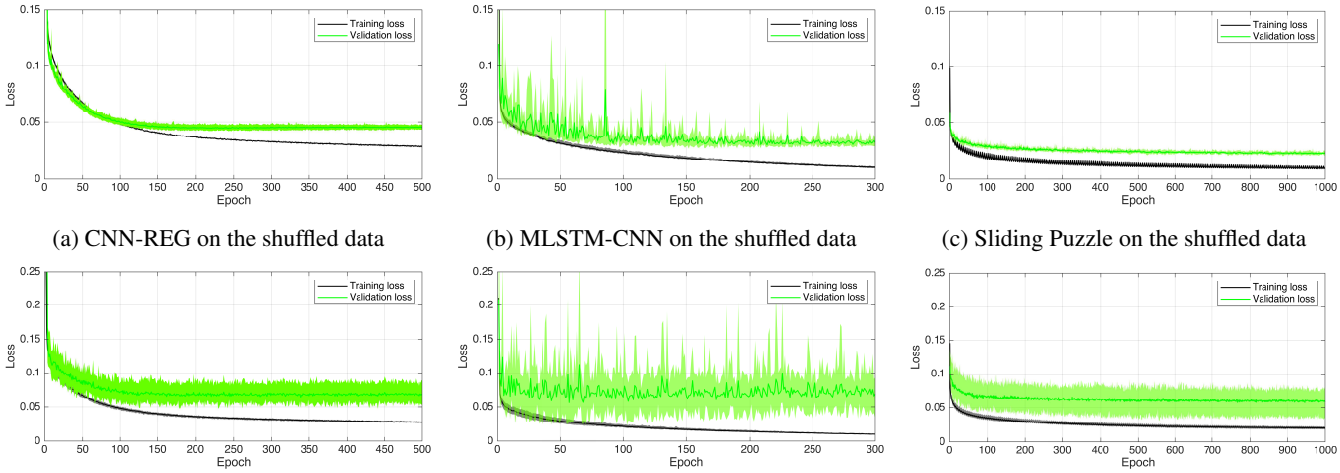
5.2 Neural network structures

Both MLSTM-CNN and Sliding Puzzle improve on the original CNN-REG. Even though MLSTM-CNN adds complexity in a parallel track, the global average pooling greatly reduces the complexity of the convolutional part. Furthermore, the parallel track supplies a different view on the data, counteracting overfitting on features as well. The Sliding Puzzle network also deliberately reduces the complexity by limiting the convolutional stage and collapsing filter results by different forms of global pooling. This way both methods use a natural form of regularization, fitting the properties of the data[4].

5.3 Chronology of the data

The measured data was collected over two years, in which the ship encountered many different sea states. Within the data set,

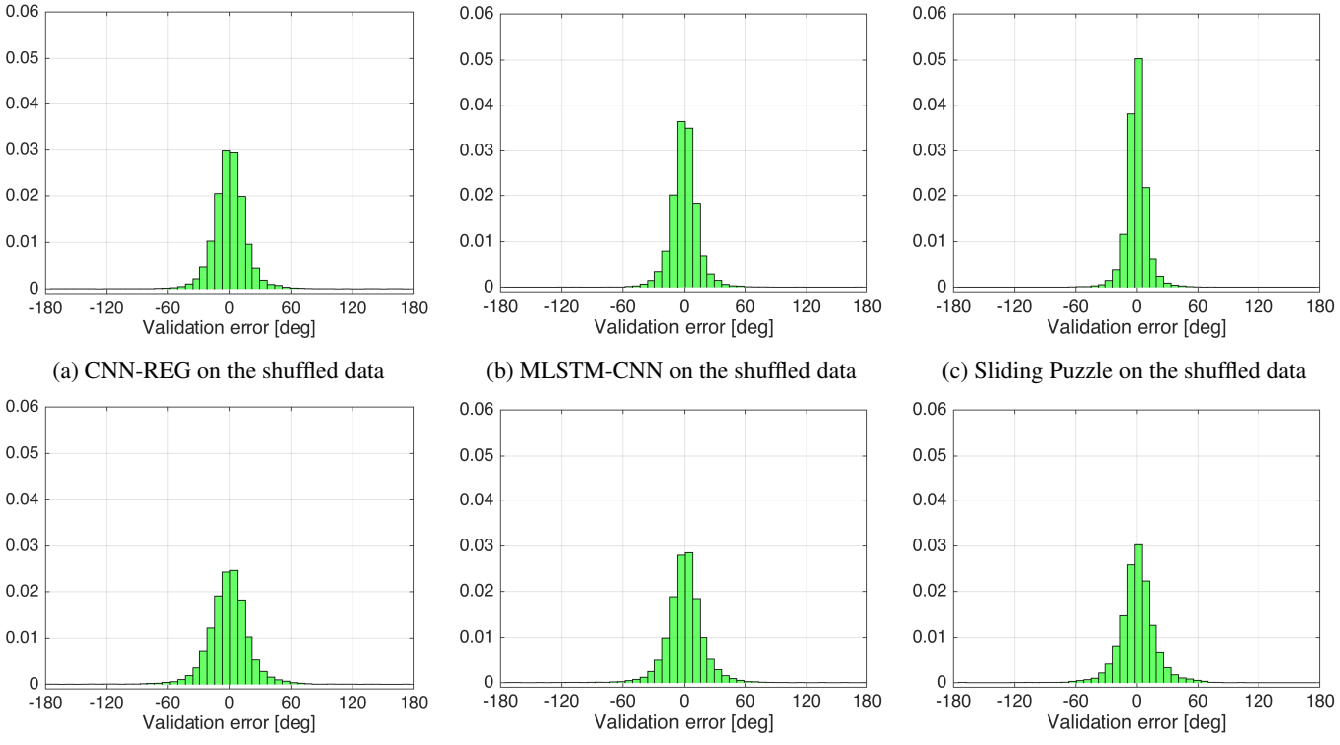
[4]Designing neural networks to general properties of the data is different from data snooping, as no sample knowledge is used, just knowledge about the way the data is acquired.



(a) CNN-REG on the shuffled data (b) MLSTM-CNN on the shuffled data (c) Sliding Puzzle on the shuffled data

(d) CNN-REG on the chronological data (e) MLSTM-CNN on the chronological data (f) Sliding Puzzle on the chronological data

FIGURE 13: Training and validation losses from the three networks on the shuffled and chronological measured data. Note the much larger spread in validation losses for the chronological data.



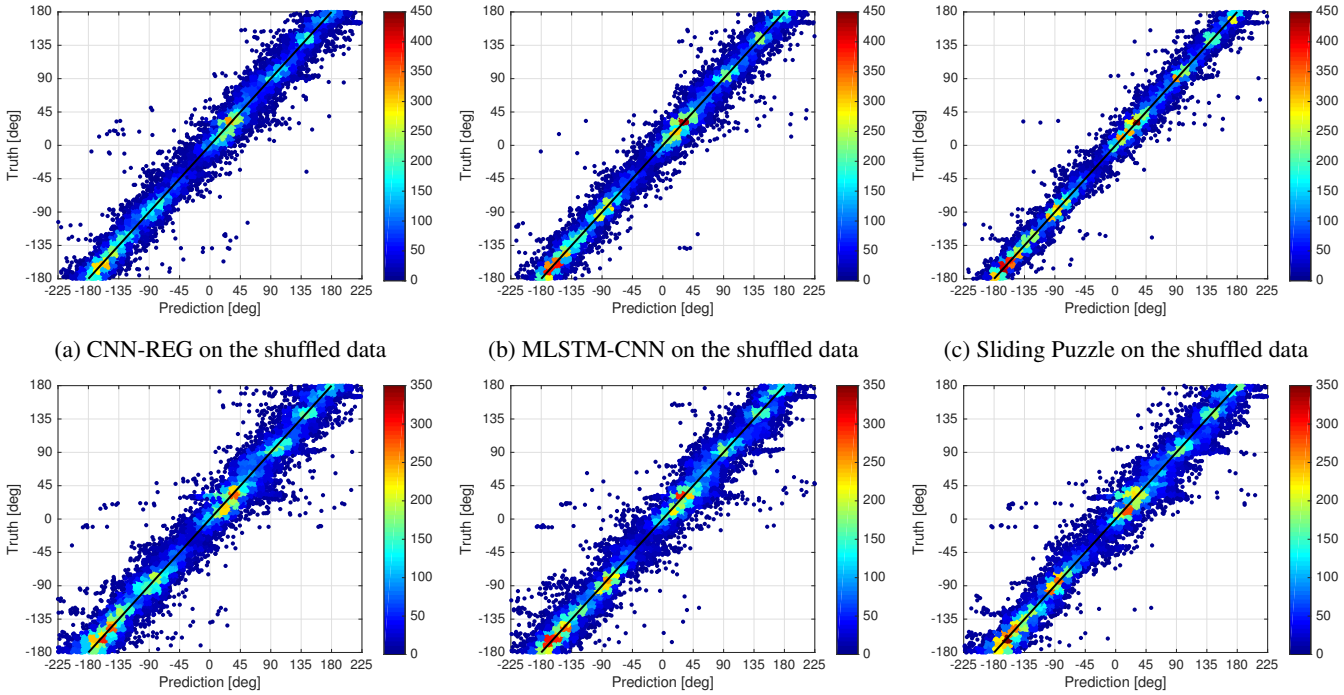
(a) CNN-REG on the shuffled data (b) MLSTM-CNN on the shuffled data (c) Sliding Puzzle on the shuffled data

(d) CNN-REG on the chronological data (e) MLSTM-CNN on the chronological data (f) Sliding Puzzle on the chronological data

FIGURE 14: Histograms of the validation errors from the three networks on the shuffled and chronological measured data.

the relative wave direction can be estimated reliably. However, it is important to know how good the predictions will be when new data is seen. The 5-fold cross validation shows that there are no problems and it is clear that no samples have been duplicated, so there is no contamination of the validation sets. Still there is a problem.

Keeping the data ordered chronologically before splitting the set into a training and validation set shows rather different validation



(a) CNN-REG on the shuffled data

(b) MLSTM-CNN on the shuffled data

(c) Sliding Puzzle on the shuffled data

(d) CNN-REG on the chronological data

(e) MLSTM-CNN on the chronological data

(f) Sliding Puzzle on the chronological data

FIGURE 15: Truth and prediction values from the three networks on the shuffled and chronological measured data. Note that the outliers are few, as most predictions are concentrated around the identity line.

results. Since this is a clear measure of the expected performance on new, unseen data, the previous case must have had some sort of contamination. Since we are using time series data for regression, we enter a realm in which the rules are different. Since the conditions the ship sailed in gradually change over time, the situation we have here is best described by a simple analogy. Suppose a network needs to be trained to classify different breeds of cats in images. The data set that is given contains an equal number of images for each breed, grouped per breed. If the network is trained on at least some images for every breed, it will be able to accurately classify any image from the validation set. If some breeds were absent in the training set, performance on the validation set will deteriorate. This will happen if the validation set consists of consecutive samples from the original chronological set. A common solution is to shuffle the dataset before splitting. Similarly, if we do not shuffle the ship motion/sea state dataset before splitting, some weather patterns will not be present in the training set, as they simply only occurred at specific times.

In a classification problem, the dataset is supposed to contain all classes. In our regression problem, with a dataset that grows over time, we do not have that guarantee. So, splitting before shuffling gives an estimate of the performance on new data. If the performance is significantly worse than for the shuffled set, the data set is not complete yet, turning this into an important measure. The score on the shuffled set is still a good indication of the possible performance on a complete set.

6 CONCLUSIONS

The methods presented in this paper show good results and compare well to established methods. The data used to achieve this does show some shortcomings though and does not cover all expected sea states. More good quality data is needed to improve methods based on measured data.

The way the data is used for training is crucial when evaluating method performance. We have seen that there is quite a difference in performance between using the data chronologically and shuffled. Even though shuffling is common practice, it is to be used carefully when using data that is acquired over time. The advice is to use both training strategies to get insight in both the possible performance and the actual performance when applied to new data.

REFERENCES

[1] C. Stredulinsky, D., and M. Thornhill, E., 2011. “Ship motion and wave radar data fusion for shipboard wave measurement”. *Journal of Ship Research,* ***55***, 06, pp. 73–85.

[2] Iseki, T., and Ohtsu, K., 2000. "Bayesian estimation of directional wave spectra based on ship motions". *Control Engineering Practice,* ***8***, 02, pp. 215–219.

[3] Tannuri, E. A., Sparano, J. V., Simos, A. N., and Cruz, J. J. D., 2003. "Estimating directional wave spectrum based on stationary ship motion measurements". *Applied Ocean Research,* ***25***, pp. 243–261.

[4] Nielsen, U., 2006. "Estimations of on-site directional wave spectra from measured ship responses". *Marine Structures,* ***19***, 01, pp. 33–69.

[5] Nielsen, U. D., Brodtkorb, A. H., and Srensen, A. J., 2018. "A brute-force spectral approach for wave estimation using measured vessel motions". pp. 101–121. Exported from https://app.dimensions.ai on 2019/01/07.

[6] Spiegel, S., Gaebler, J., Lommatzsch, A., De Luca, E., and Albayrak, S., 2011. "Pattern recognition and classification for multivariate time series". In Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data, SensorKDD '11, ACM, pp. 34–42.

[7] Fu, Y., 2015. *Human Activity Recognition and Prediction*, 1st ed. Springer.

[8] Geurts, P., 2001. "Pattern extraction for time series classification". In Principles of Data Mining and Knowledge Discovery, L. De Raedt and A. Siebes, eds., Springer Berlin Heidelberg, pp. 115–127.

[9] Pavlovic, V., Frey, B. J., and Huang, T. S., 1999. "Time-series classification using mixed-state dynamic bayesian networks". In Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149), Vol. 2, pp. 609–615 Vol. 2.

[10] Reiss, A., and Stricker, D., 2011. "Introducing a modular activity monitoring system". In 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 5621–5624.

[11] Kampouraki, A., Manis, G., and Nikou, C., 2009. "Heartbeat time series classification with support vector machines". *IEEE Transactions on Information Technology in Biomedicine,* ***13***(4), July, pp. 512–518.

[12] Haselsteiner, E., and Pfurtscheller, G., 2000. "Using time-dependent neural networks for eeg classification". *IEEE Transactions on Rehabilitation Engineering,* ***8***(4), Dec, pp. 457–463.

[13] Xing, Z., Pei, J., and Keogh, E., 2010. "A brief survey on sequence classification". *SIGKDD Explor. Newsl.,* ***12***(1), Nov., pp. 40–48.

[14] Yi, S., Ju, J., Yoon, M., and Choi, J., 2017. "Grouped convolutional neural networks for multivariate time series". *CoRR,* ***abs/1703.09938***.

[15] Binkowski, M., Marti, G., and Donnat, P., 2017. "Autoregressive convolutional neural networks for asynchronous time series". *CoRR,* ***abs/1703.04122***.

[16] Borovykh, A., Bohte, S. M., and Oosterlee, C. W., 2017. "Conditional time series forecasting with convolutional neural networks".

[17] Bernacki, M., and Wodarczyk, P., 2005. Principles of training multi-layer neural network using backpropagation. `http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html`. [Online; accessed 7-January-2019].

[18] He, K., Zhang, X., Ren, S., and Sun, J., 2015. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification".

[19] Hochreiter, S., and Schmidhuber, J., 1997. "Long short-term memory". *Neural Comput.,* ***9***(8), Nov., pp. 1735–1780.

[20] Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T., 2014. "Long-term recurrent convolutional networks for visual recognition and description". *CoRR,* ***abs/1411.4389***.

[21] Hu, J., Shen, L., and Sun, G., 2017. "Squeeze-and-excitation networks". *CoRR,* ***abs/1709.01507***.

[22] Nair, V., and Hinton, G. E., 2010. "Rectified linear units improve restricted boltzmann machines". In Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10, Omnipress, pp. 807–814.

[23] Karim, F., Majumdar, S., Darabi, H., and Harford, S. "Multivariate LSTM-FCNs for time series classification".

[24] Kiefer, J., and Wolfowitz, J., 1952. "Stochastic estimation of the maximum of a regression function". *Ann. Math. Statist.,* ***23***(3), 09, pp. 462–466.

[25] Chollet, F., et al., 2015. Keras.

[26] Abadi, M., et al., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[27] Bispo, I., N. Queiroz Filho, A., Tannuri, E., and N. Simos, A., 2016. "Motion-based wave inference: Monitoring campaign on a turret fpso". In Proc. ASME 2016 35th International Conference on Ocean, Offshore and Arctic Engineering OMAE2016, paper OMAE2016-54956.

[28] Mak, B., and Düz, B., 2019. "Ship As A Wave Buoy - Using simulated data to train neural networks for real time estimation of relative wave direction". In Proc. ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering OMAE2019, paper OMAE2019-96225.