Reproduction of A CFCC-LSTM Model for Sea Surface Temperature Prediction

Reproducibility project - CS4240 Deep Learning

Group 86 - Matthijs Arnoldus (@: m.arnoldus-1@student.tudelft.nl, 4928091), Marius Birkhoff (@: M.J.H.Birkhoff@student.tudelft.nl, 4724259), Luuk Haarman (@: L.A.Haarman@student.tudelft.nl, 4931173)

Reproducibility is an important aspect of scientific research. This blog describes the process of reproducing the results of Yang et al's "A CFCC-LSTM Model for Sea Surface Temperature Prediction" by group 86 for TU Delft's Deep Learning course. The paper presented a combined fully connected long short-term memory convolutional neural network for predicting the sea surface temperature. The efficiency of their model was validated using data from the Bohai Sea in China and the China Ocean.

The goal of this project is to reproduce and verify the results of Table 1 and Figure 2 of Yung et al.'s paper. In our reproduction, the Bohai data set was used.

Data collection

The first step towards reproducing the paper was to get our hands on the correct data. Because Yung et al. did not specify where their data came from, we had a hard time finding a data set that would satisfy our needs. With some help from our TA and external supervisor, our eyes fell on the Physical Sciences Laboratory by the National Oceanic and Atmospheric Administration (PSL NOAA), which has over 40 years of worldwide data on sea surface temperature. Further exploration of the PSL NOAA led us to yearly datasets. Downloading them from 1981 up until 2021 resulted in 41 files of almost 18 Gigabytes in total.

Only a small portion of the retrieved data was required because we were evaluating using the Bohai Sea and not using the entire planet. This data transformation could be executed quite easily and would save a lot of storage and memory. By making use of the

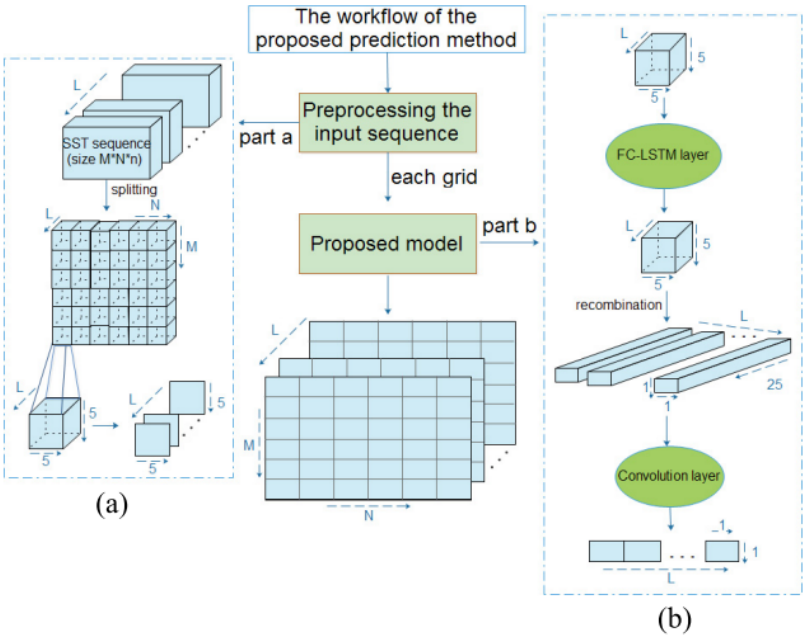
command line interface Climate Data Operators (CDO), we could merge all points in time into one file. Once done, the latitude and longitude dimensions could be reduced to only the Bohai Sea. The result set was 10 megabytes in size and contained 14732 points in time. The following commands were executed:

```
cdo -mergetime ./data/*.nc ./sst.worldwide.day.mean.1981-2021.nc

cdo -sellonlatbox,117,122,37,42 ./sst.worldwide.day.mean.1981-2021.nc
./sst.bohai.day.mean.1981-2021.nc
```

Model description

The paper proposes three models used for sea surface temperature prediction. The first model is an FC-LSTM model. This model takes a grid as input and returns a grid as output as well. It consists of a LSTM layer combined with a fully connected layer and is used for learning the temporal relationships (the effect of previous days on our prediction). The other two models are CFCC-LSTM, FC-LSTM models with an added convolutional layer. These convolutional layers are used for learning the spatial relationships (the effect of nearby temperature points on a center point). The difference between the two CFCC-LSTM models is that one uses average convolution, and the other uses weights that it can learn to optimize the prediction.



[1]

Figure from the paper that describes the pre-processing step as well as the model architecture.

Implementation

For the implementation of the model we used PyTorch, and the model was implemented in Google Colab. Our first steps were to perform the preprocessing of the data. This included the splitting of the data into three distinct lists. A list of latitude values, a list of longitude values, and a list of the SST (Sea surface temperature) values. The SST list used with latitude and longitude coordinates could be used to retrieve the temperature value of that specific point on a specific day. Because land temperatures are stored as extremely large negative values, we clip the temperatures in the SST list as between 0 and infinity. If we do not clip the temperatures the initial prediction errors will be too large for learning.

After this, some additional steps are required before the data is used for training. These steps include splitting the data into a training and validation set. We used an 80/20 split, so 80% of the data was used for training, and 20% for validation. When preparing the data for training, we used a sliding window with the length of our history length for the input values, and a sliding window with the length of our prediction length for the target values. So if we want to use 4 days to predict the 5th, we use a sliding window size of 4 and 1 for training and target values respectively. After

[1] Sourced from Yang et al. (2018) “A CFCC-LSTM Model for Sea Surface Temperature Prediction”

this, some transformations were applied to these samples to fit them to the model, namely flattening the grid of 5x5 values to a list of 25 values such that the LSTM layer could take them as input.

The FC-LSTM model was implemented as follows: a single LSTM layer would take the input and return an output with the same dimensions, this output would be fed into a fully connected Linear layer which would combine the history into a new grid with the number of days we want to predict. This means that when we use 4 days to predict the 5th, the Linear Layer would have 4 inputs and 1 output. After this, a ReLU transformation is done and the data is transformed into a 5x5 grid again. In the CFCC-LSTM Model, this transformation would not occur, but the output of the FC-LSTM model would be fed into a convolutional layer to predict the temperature of a single point instead of a grid.

During the training process, we used root mean squared error loss and an Adam optimizer with a learning rate of lr=0.001.

Below are our FC-LSTM class and CFCC-LSTM class. The CFCC-LSTM class extends the FC-LSTM class.

```
class FCLSTM(nn.Module):
  def __init__(self, input_size=25, hidden_size=25, history_length=4,
prediction_length=1, device='cpu'):
    super(FCLSTM, self).__init__()

    self.input_size = input_size
    self.hidden_size = hidden_size

    self.history_length = history_length
    self.prediction_length = prediction_length

    self.device = device

    self.lstm_layer = nn.LSTM(input_size, hidden_size, num_layers=1,
batch_first=True)
    self.fc_layer = nn.Linear(history_length * hidden_size, prediction_length *
hidden_size)
    self.relu = nn.ReLU()

  # takes in a grid of N x history_length x 25 and returns a grid of N x
prediction_length x 5 x 5
  def forward(self, input, reshape=True):
      num_samples = input.shape[0]
      h_t = torch.zeros(1, num_samples, self.hidden_size).to(self.device)
      c_t = torch.zeros(1, num_samples, self.hidden_size).to(self.device)
```

```
      output, (h_t, c_t) = self.lstm_layer(input, (h_t, c_t)) # output: N x
history_length x 25
      output = torch.flatten(output, start_dim=1) # = N x history_length * 25
      output = self.fc_layer(output) # = N x prediction_length * 25
      if reshape:
        output = torch.reshape(output, (num_samples, self.prediction_length,
int(np.sqrt(self.hidden_size)), int(np.sqrt(self.hidden_size)))) # N x
prediction_length x 5 x 5

      output = self.relu(output)
      return output
```

```
class CFCCLSTM(FCLSTM):
  def __init__(self, input_size=25, hidden_size=25, history_length=4,
prediction_length=1, device="cpu", mode="weighted"):
    super(CFCCLSTM, self).__init__(input_size, hidden_size, history_length,
prediction_length, device)

    self.conv_layer = nn.Conv1d(1, 1, kernel_size=hidden_size, stride=1,
padding=0)
    if mode == "average":
      self.conv_layer = nn.Conv1d(1,1, kernel_size=hidden_size, stride=1,
padding=0)
      with torch.no_grad():
        self.conv_layer.weight.data = torch.mul(torch.ones(1, 1, hidden_size),
1.0 / hidden_size)
        self.conv_layer.weight.requires_grad = False

  # takes in a grid of N x history_length x 25 and returns a grid of N x
prediction_length x 5 x 5
  def forward(self, input):
      output = super().forward(input, False) # N x prediction_length x 5 x 5

      # put data in 1 channel
      output = torch.unsqueeze(output, dim=1) # N x 1 x 25
      output = self.conv_layer(output) # N x 1 x 1
      output = torch.flatten(output, start_dim=1)

      return output
```

Hyperparameter tuning

The paper provides a few learning rate options. For us, these were much too high. This could be due to differences in our model architecture. Due to time constraints, hyperparameter tuning was mostly done only for the FC-LSTM model. After some tuning, a

learning rate of lr=0.001 seemed to perform well. This learning rate was also used for the CFCC-LSTM Models and seemed to perform well as well. Similar to the paper, we randomly initialised the weights of our models. The paper states that they achieved optimal performance with a fixed initialization scheme but does not provide these values.

We used the same number of epochs as the paper (50 and 100), but due to the differences in our model, we are not quite sure whether or not these numbers are optimal for our model. If we had had more time we would have run the model for longer iterations to see if a large jump in performance was possible. When we look at the gradients of our loss this is likely not the case. However, it might be the case that we ended up in local minima and a larger number of epochs could have helped us escape these local minima.

Results

The main goal of this reproduction was to reproduce Table I from the paper, as well as showing a figure similar to Figure 2 in the paper. Below, we show Figure 2 from the paper, as well as our

reproduction for this figure.

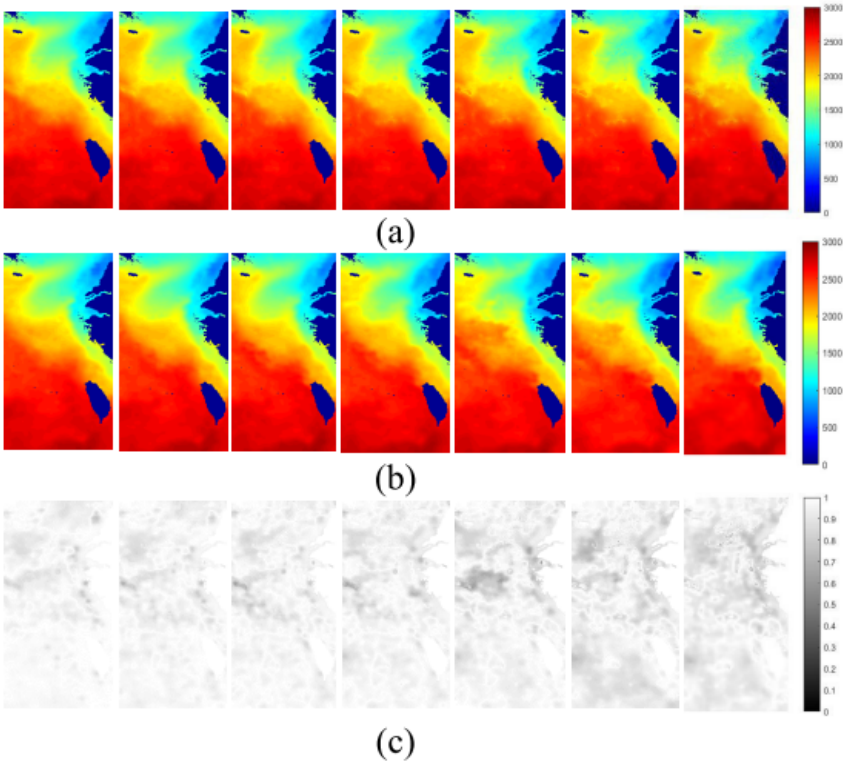
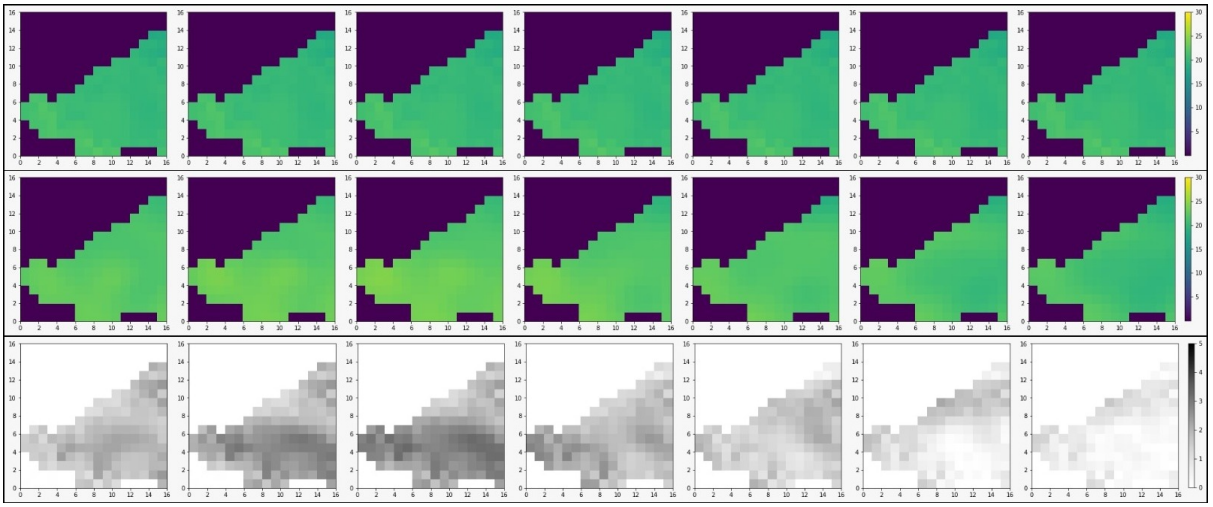


Fig. 2. Comparison between the predicted output and the ground truth for seven days on China Ocean data set. (a) Predicted SST data for seven days. (b) Ground truth SST data for seven days. (c) Difference between the ground truth and the predicted output.

Figure 2 from the paper



Our comparison between the predicted output and the ground truth for seven days on the Bohai Sea data set. The first row shows the predicted sea surface temperature, the second row shows the ground truth, and the third row shows the differences.

In both figures, the predicted output is compared to the ground truth values. In our reproduction we used the Bohai Sea data set instead of the China Ocean data set, as this was the only data set we had. Remarkable is the fact that our reproduction seems to perform better on days further in the future. However, it must be said that this figure just shows one prediction of seven days, so this might be due to randomness. The goal of this figure was solely to show an indication of how the model performs in comparison to the ground truth data.

In contrast to Figure 2, Table 1 from the paper clearly shows the performance of the model. This table compares the three versions of the model, FC-LSTM, Average CFCC-LSTM and Weighted CFCC-LSTM. Each model is used to predict 1 day using 4 previous days, and 7 days using 20 previous days. Training is done either for 50 or for 100 iterations, and for evaluation Root Mean Squared Error as well as their own accuracy function are used. The tables below show the results from the paper and the results obtained from our reproduction.

CFCC-LSTM Model	Metrics	Iterations (prediction days = 1,7)			
		50	100	50	100
Only FC-LSTM Layer	$RMSE$	0.1826	0.2281	0.5119	0.5351
	ACC	99.21	99.21	98.23	98.23
FC-LSTM Layer + Average Convolution	$RMSE$	**0.1420**	0.1716	0.4699	0.4725
	ACC	99.27	99.27	98.38	98.38
FC-LSTM Layer + Weighted Convolution	$RMSE$	0.1583	**0.1583**	**0.3242**	**0.3242**
	ACC	**99.32**	**99.32**	**98.41**	**98.41**

Table I in the paper

Model	Metric	Iterations (prediction days = 1, 7)			
		50	100	50	100
FC-LSTM	RMSE	0.798	0.666	0.865	0.762
	ACC	-1.997	-0.791	-2.223	-8.833
FC-LSTM + Average Convolution	RMSE	**0.494**	**0.416**	**0.726**	**0.713**
	ACC	0.83	0.896	-0.139	-0.275
FC-LSTM + Weighted Convolution	RMSE	0.513	0.425	0.908	0.756
	ACC	0.360	0.411	-1.278	-1.082

Table I results for our reproduction

As can be seen in the second table, our RMSE scores are similar to those of the paper, albeit a bit higher. This could be due to the differences in our model. The trends seem similar, however, our average convolution outperforms the weighted convolution slightly.

One would think that weighted convolution performs better, as it can also learn average convolution. However, it might be the case that the weighted convolution needs more time to train towards average convolution due to its random initialization.

Our accuracy values are much different, and not necessarily between 0 and 1. This is because of the way they formulate their accuracy, which is given by the following formula.

$$\mathrm{ACC} = 1 - \frac{\sum_{i=1}^{n} \left(\frac{|X_{\mathrm{model},i} - X_{\mathrm{obs},i}|}{X_{\mathrm{obs},i}} \right)}{n}$$

From this formula, it becomes clear that the accuracy does not bound the values, and will only be bounded when the absolute error is very small, and smaller than the original temperature predicted. In other words, the accuracy metric seems to us like a scaled-down version of mean absolute error and could be negative when the absolute error is large and, therefore, does not necessarily provide an insight into prediction accuracy that RMSE does not.

Reproduction hurdles

During the reproduction we encountered several difficulties, which could have been prevented by better documentation of methods and tools used in the original paper. We will describe what problems we encountered and how we have overcome these issues in our reproduction.

First of all, the dataset used was not provided in the paper. Because of this, we had to find a dataset of the sea temperatures in the Bohai Sea ourselves. This caused us to work with a wrong data set at first, which we found out only contained average data points over a year, instead of daily values. Also, we cannot be sure whether or not the data we used reflects the data used for the original paper, which can influence the results of our reproduction.

Next to that, not all information we needed was present in the paper and there were some ambiguities. This caused us to make some implementation decisions that we were not sure properly reflected the original model. For example, the usage of the word layer in the description of the FC-LSTM model could describe a single layer, such as a single LSTM layer, or rather that it’s a part of the complete FC-LSTM model. We made the decision to use a single layer for this. Because of this, we have a single LSTM layer,

which feeds into a single Linear layer which feeds into a single Conv1d Layer. We are not sure whether this was what the paper describes, or whether this solution performs as well as a model with multiple layers could. We see this as a possible reason for our model having a higher RMSE, and also why the provided learning rates are much different than the learning rates that performed well for our model. Also the paper does not describe any non-linear activations, although this is quite common in model architectures and in ours it proved beneficial. Finally, their training and validation details are not provided, so we went with an 80/20 split but we are not sure whether or not they did this or even used a validation set.

Some other missing implementation details could have had an effect as well. We used PyTorch, but we are not sure what language or tool was used in the original paper. This could have had minimal effects on the results.

Conclusion

To conclude, we were somewhat able to reproduce the results that we set out to, although our RMSE scores were somewhat higher than in the paper. Also, we argue that the accuracy metric is not much different from RMSE in the information it provides. The differences in score could be due to our implementation being different from the original in model architecture and hyperparameters. These differences occurred because we had to make some of our own implementation choices due to missing information and ambiguities in the paper. Our reproduction, thus, provides a model that is able to predict future sea surface temperatures, but could have been better if the paper provided more details of their hyperparameters and model architecture.

Individual Contributions

Matthijs Arnoldus	Model construction, model evaluation
Marius Birkhoff	Data gathering, data preprocessing, code refactoring
Luuk Haarman	Model construction, model evaluation

Acknowledgements

NOAA High Resolution SST data provided by the NOAA/OAR/ESRL PSL, Boulder, Colorado, USA, from their website: https://psl.noaa.gov/data/gridded/data.noaa.oisst.v2.highres.html.

Y. Yang, J. Dong, X. Sun, E. Lima, Q. Mu and X. Wang, "A CFCC-LSTM Model for Sea Surface Temperature Prediction," in *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 2, pp. 207-211, Feb. 2018, doi: 10.1109/LGRS.2017.2780843.