Space Weather

Supporting Information for

Forecasting GICs and geoelectric fields from solar wind data using LSTMs: application in Austria

R. L. Bailey[1], R. Leonhardt[1], C. Möstl[2], C. Beggan[3], M. A. Reiss[2,3], A. Bhaskar[4], A. J. Weiss[2,5]

[1]Conrad observatory, Zentralanstalt f¨ur Meteorologie und Geodynamik, Vienna, Austria
[2]Space Research Institute, Austrian Academy of Sciences, Graz, Austria
[3]British Geological Survey, Edinburgh, UK
[4]Space Physics Laboratory, ISRO/Vikram Sarabhai Space Centre, Trivandrum, India
[5]Institute of Physics, University of Graz, Universit¨atsplatz 5, 8010 Graz, Austria

Contents of this file

Additional Supporting Information (Files uploaded separately)

None.

Introduction

This document contains supporting information for the manuscript, “Forecasting GICs and geoelectric fields from solar wind data using LSTMs: application in Austria” by Bailey, R. L. et al., submitted to *Space Weather*.

This document describes the model architecture and hyper parameters used for training LSTMs for two purposes:

1) LSTM-E: An LSTM for predicting the geoelectric field.
2) LSTM-GIC: An LSTM for predicting substation-specific GICs in the Austrian power grid.

The LSTMs were trained using the Python package *keras*. The code used to define the custom BasicAttention layer is included in this supporting information, otherwise all layers and objects referred to in the diagrams are *keras*-specific objects.

Figure S1 - LSTM-E diagram.

The general structure of the model for forecasting geoelectric fields has two branches that branch out from the initial two LSTM layers that process the input features initially. These both go into separate but identical BasicAttention layers. The left-hand side of the LSTM-E tackles a regression problem to predict the magnitude of the geoelectric field (ignoring direction), while the right-hand side of the LSTM deals with the classification problem of attempting to predict the direction of the field.

Two LSTMs of this type were trained: one for the x-component of the geoelectric field (LSTM-Ex) and one for the y-component of the geoelectric field (LSTM-Ey). Through hyper parameter tuning, different parameters were chosen for each variable.

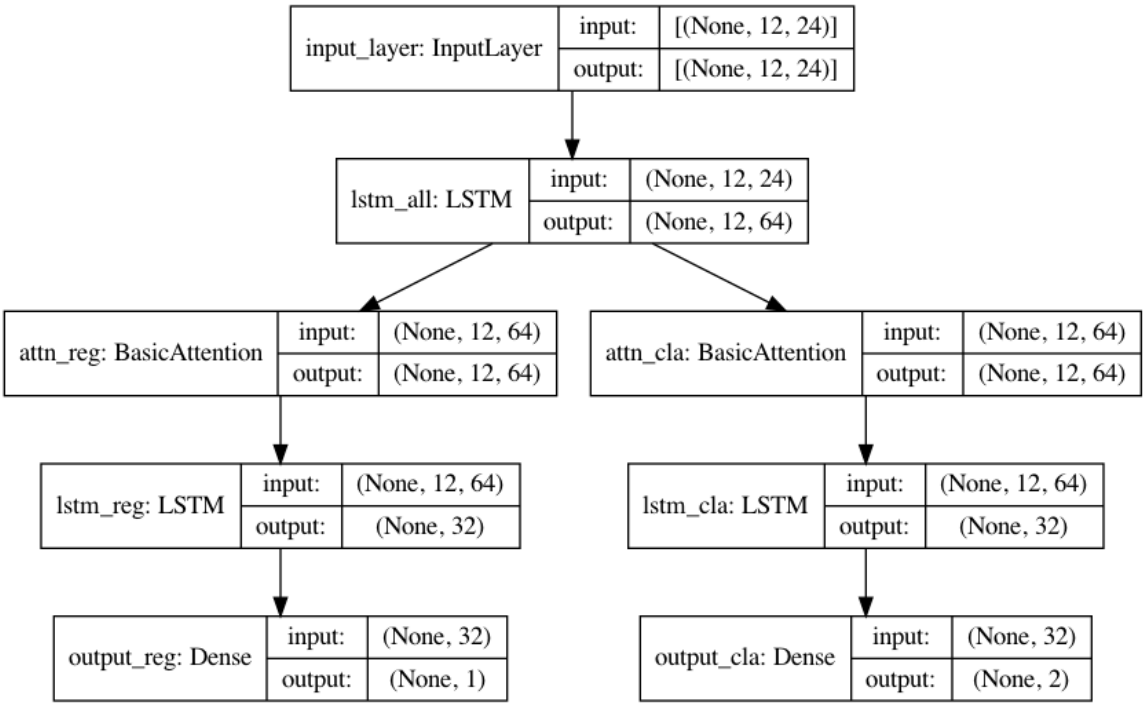

Table S2 - Hyperparameters for LSTM-E training. The hyper parameters used for training two different LSTMs: one for Ex and one for Ey.

	LSTM-Ex	LSTM-Ey
Loss weighting [regression, classification]	[1000, 1]	[2000, 1]
Batch size	64	32
Epochs	10	10
Fraction of LSTM dropout	0.1	0.1
Number of LSTM hidden states	32	32

Figure S3 - LSTM-GIC diagram.
In comparison to the LSTM used to predict the geoelectric field, in the case of GICs observed at specific transformers we ignore the classification problem and instead focus onto on predicting the magnitude. The samples go through one LSTM layer before being put through an Attention layer, which returns the sequences. These are fed into another LSTM layer before being reduced to a single value as output using a feed-forward Dense layer.

Two LSTMs of this type were trained: one for GICs seen in a substation near Vienna (LSTM-GIC1) and one for GICs seen in a substation near Salzburg (LSTM-GIC5). Through hyper parameter tuning, different parameters were chosen for each target variable.

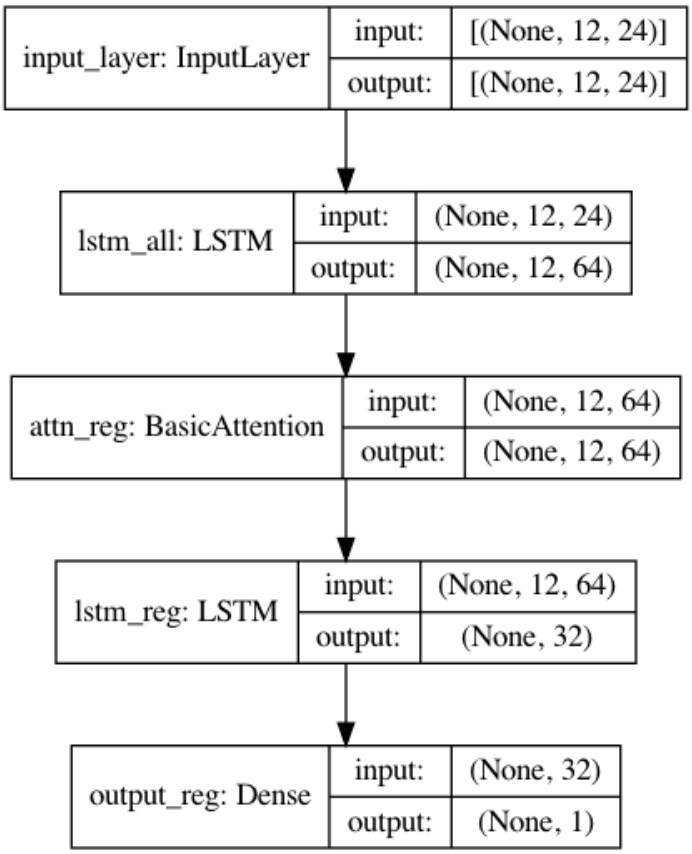

Table S4 - Hyperparameters for LSTM-GIC training. The hyper parameters used for training two different LSTMs: one for GICs at substation #1 near Vienna (GIC1) and for a power grid substation near Salzburg (GIC5).

	LSTM-GIC1	LSTM-GIC5
Batch size	32	64
Epochs	10	20
Fraction of LSTM dropout	0.1	0.3
Number of LSTM hidden states	32	64

Text S5 - Python Object: BasicAttention layer. This is the Python script for a custom Attention layer object included in the LSTM architecture.

```
class BasicAttention(Layer):
    '''Basic Self-Attention Layer built using this resource:
    https://towardsdatascience.com/create-your-own-custom-atten-
tion-layer-understand-all-flavours-2201b5e8be9e'''

    def __init__(self, return_sequences=True, n_units=1,
w_init='normal', b_init='zeros', **kwargs):
        self.return_sequences = return_sequences
        self.n_units = n_units
        self.w_init = w_init
        self.b_init = b_init
        super(BasicAttention,self).__init__(**kwargs)

    def build(self, input_shape):
        self.n_features = input_shape[-1]
        self.seq_len = input_shape[-2]

        self.W=self.add_weight(name="att_weight", shape=(self-
.n_features,self.n_units),
                               initializer=self.w_init)
        self.b=self.add_weight(name="att_bias", shape=(self.se-
q_len,self.n_units),
                               initializer=self.b_init)

        super(BasicAttention,self).build(input_shape)

    def call(self, x):

        e = K.tanh(K.dot(x,self.W)+self.b)
        a = K.softmax(e, axis=1)
        output = x*a

        if self.return_sequences:
            return output

        return K.sum(output, axis=1)

    def get_config(self):
        config = super(BasicAttention, self).get_config()
        config["return_sequences"] = self.return_sequences
        config["n_units"] = self.n_units
        config["w_init"] = self.w_init
        config["b_init"] = self.b_init
        #config["name"] = self.name
        return config
```

```
@classmethod
def from_config(cls, config):
    return cls(**config)
```