Hyperparameters of RNN

quantra

Table of Contents

The hyperparameters of an RNN and LSTM are the same. So, let us understand the various hyperparameters that are unique to them.

Model Training:

Before you train a model, you need to understand the various components involved in that process. Apart from fetching the data and scaling it. We need to specify the model architecture before you compile and fit the train data to it. When using an RNN architecture, we will pass the past 'x' days data as the input at every time step, and the algorithm will train on this data and arrive at the trend. In the example provided in the next unit, we have used the past 20 days data as the input at every time step.

The first step in the model creation is declaring the model type. In deep learning we use a sequential model. It is called so because we decide the sequence of layers the data should pass through. Keras deep learning library gives us the flexibility to tune parameters used in every layer.

Following the sequential layer, we have an RNN layer.

Syntax:

```
keras.layers.SimpleRNN(units, activation='tanh', use_bias=True,
                       kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal',
                       bias_initializer='zeros', kernel_regularizer=None,
                       recurrent_regularizer=None, bias_regularizer=None,
                       activity_regularizer=None, kernel_constraint=None,
                       recurrent_constraint=None, bias_constraint=None, dropout=0.0,
                       recurrent_dropout=0.0, return_sequences=False, return_state=False,
                       go_backwards=False, stateful=False, unroll=False)
```

- recurrent_initializer: Initializer function for the recurrent_kernel weights matrix, used for the linear transformation of the recurrent state. When initializing the weight matrix in RNNs, it's not uncommon to use random uniform or random normal initialization.

- recurrent_regularizer: Regularizer function applied to the recurrent_kernel weights matrix. This is applicable to only the RNN layer.

Constraints:

A constraint is a limitation on one of the parameters that we want to train/monitor. Constraining the weight matrix directly is a kind of regularization used in deep learning. For example, you can use the maxnorm(m) function, where m is a positive number. It will scale your weight matrix by a factor such that the square root of the sum of squares of the weights, also called the norm of the weight matrix is equal to m. This constraint technique works well in combination with a dropout layer.

- recurrent_constraint: Constraint function applied to the recurrent_kernel weights matrix.

Dropout layers can be added after the activation function layer, this holds true for an RNN layer, where the dropout is applied, by randomly switching off the weights of the past information. The dropout layer has a single hyperparameter, which signifies the ratio of nodes to be dropped in a particular layer.

- recurrent_dropout: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

Return_sequence:

When we set the hyperparameter return_sequence=TRUE, the output of the layer will be a sequence of the same length as the input, but if the return_sequence=FALSE then the output will be just one value corresponding to the last output.

- return_sequences: Boolean. Whether to return the last output in the output sequence, or the full sequence.

Return_state:

The sequential information is preserved in the recurrent network's hidden state, which manages to span many time steps as it cascades forward to affect the processing of each

new example. An event downstream in time depends upon, and is a function of, one or more events that came before it. return_state finds correlations between events separated by many moments, and these correlations are called “long-term dependencies”. When we set return_state to True, we are essentially passing the last hidden state of the last layer. This is useful when you are adding another RNN or LSTM layer after this layer.

- return_state: Boolean. Whether to return the last state in addition to the output.
- go_backwards: Boolean (default False). If True, the model will process the input sequence backwards and return the reversed sequence.

A stateful RNN or LSTM essentially maintains the order of training for every batch. What this means is that after the first batch is trained the weights learnt from this are applied to the next batch in the sequence. Let us say that we are predicting the direction of the market using the past data. Every datapoint in the train data consists of the past 20 days data. When we set the stateful parameter to true then the weights obtained after the training the first sample in a batch is updated only after training the first sample of the next batch, but not the second sample in the same batch. So, the sequence of the data matters, and there is a one to one correspondence between the train data elements of different batches.

- stateful: Boolean (default False). If True, the last state for each sample at index ‘i’ in a batch will be used as initial state for the sample of index ‘i’ in the following batch.

Example:

Batch 1 data:

Date	Open	High	Low	Close
22-06-2018 15:04	1291.5	1292.85	1291.5	1292.6
22-06-2018 15:05	1292.3	1292.3	1289.2	1290
22-06-2018 15:06	1290	1291.05	1289.9	1290
22-06-2018 15:07	1290	1290.7	1289.5	1290
22-06-2018 15:08	1290	1290	1288.5	1288.9
22-06-2018 15:09	1288.95	1291.7	1288.9	1290
22-06-2018 15:10	1290.6	1294.2	1290.6	1294.2

Batch 2 data:

Date	Open	High	Low	Close
22-06-2018 15:11	1294.2	1295	1293.2	1293.2
22-06-2018 15:12	1293.2	1295.35	1293.2	1295
22-06-2018 15:13	1296.25	1298	1296.15	1297
22-06-2018 15:14	1297	1297	1295.6	1296.6
22-06-2018 15:15	1296.6	1297.35	1296	1297.35
22-06-2018 15:16	1297.35	1297.7	1296	1296.05
22-06-2018 15:17	1296.4	1297	1296.05	1297

In the above two figures, the weights/kernel obtained after passing the first row of Batch 1 will be updated after the first row of Batch 2 data is passed. This process of one to one correspondence between the elements of different batches is achieved using the stateful hyperparameter.

- unroll: Boolean (default False). If True, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.

Please go through the Keras documentation for RNNs and LSTMs to understand the new updates or features added to these layers.