RNN in Keras

examples

LSTM input shape (1)

- 3-dimensional
- (X,Y,Z) where
  - X = # of samples, Y = # of timesteps, Z = dimensions of 1 input data item
- Example
  - suppose your input is an array [14,5,6,1,77] of numbers, i.e. you have 5 samples (0 should not be in your domain!)
  - choose your timestep size (on what do you base your prediction – say 2)
  - generate timestep data as [0,14],[14,5],[5,6],[6,1],[1,77]
  - final shape [[0,14],[14,5],[5,6],[6,1],[1,77]]

LSTM input shape (2)

- Example
  - suppose your input is a numpy array X=[14,5,6,1,77] of numbers, i.e. you have 5 samples (0 should not be in your domain!)
  - if you do not know what your time step is, start with 1
  - generate timestep data as [14],[5],[6],[1],[77]
    - use reshape(X.shape[0], 1, X.shape[1]) to get the final shape [[14],[5],[6],[1],[77]]
    - note: RNNs are used for ‘remembering’ long sequences, using timesteps=1 is not logical

LSTM layer

```
LSTM(n_neurons, input_shape=(timesteps, inputDims), return_sequences=False)
```

unless stacking several layers

GRU layer

```
GRU(input_dim, output_dim, return_sequences=False)
```

input/output dimensions
similar to Dense() layers

unless stacking several
layers

Bidirectional LSTM

- Uses connections/data feed in both time directions
- Use only if needed (unlikely for language, for example)

```
Bidirectional(LSTM(n_neurons, return_sequences=True),
input_shape=(n_timesteps, inputDim))
```

TimeDistributed layer (1)

- Primarily used for many-to-many seq-to-seq problems
- **The input must be (at least) 3D**
  - This often means that you will need to configure your last LSTM layer prior to your TimeDistributed wrapped Dense layer to return sequences (set return_sequences=True)
- **The output will be 3D**
  - This means that if your TimeDistributed wrapped Dense layer is your output layer and you are predicting a sequence, you will need to resize your Y array into a 3D vector

TimeDistributed layer (2)

- Use the TimeDistributed on the output layer to wrap a fully connected Dense layer:

```
model.add(TimeDistributed(Dense(outputDim)))
```

- The output value highlights that we intend to output one time step from the sequence for each time step in the input
- The TimeDistributed achieves this trick by applying the same Dense layer (same weights) to the LSTMs outputs for one time step at a time
    - In this way, the output layer only needs one connection to each LSTM unit (plus one bias).
- For this reason, the number of training epochs needs to be increased to account for the smaller network capacity

Stacking LSTMs

To stack k LSTM layers, first (k-1) layers have to return sequences by setting return_sequences=True:

```
model.add(LSTM(n_neurons, input_shape=(time_steps, inputDim), return_sequences=True))
model.add(LSTM(n_neurons, input_shape=(time_steps, inputDim), return_sequences=True))
…
model.add(LSTM(n_neurons, input_shape=(time_steps, inputDim), return_sequences=False))
```