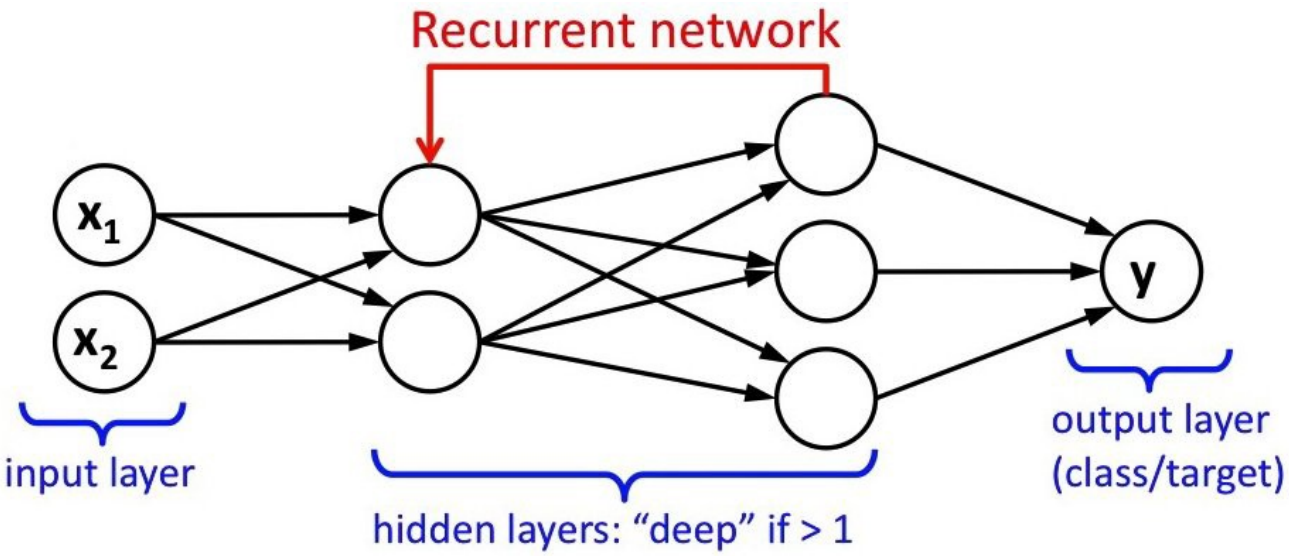
Implementation of RNN, LSTM, and GRU

Chandra Churh Chatterjee Jul 25, 2019 · 5 min read



Recurrent Neural Network

The Recurrent neural networks are a class of artificial neural networks where the connection between nodes form a directed graph along a temporal sequence. Unlike the feed-forward neural networks, the recurrent neural networks use their internal state memory for processing sequences. This dynamic behavior of the Recurrent neural networks allows them to be very useful and applicable to audio analysis, handwritten recognition, and several such applications.

Simple RNN implementation in Keras.

Mathematically the simple RNN can be formulated as follows:

$$h(t) = f_H\left(W_{IH}x(t) + W_{HH}h(t-1)\right)$$

$$y(t) = f_O\left(W_{HO}h(t)\right)$$

Where x(t) and y(t) are the input and output vectors, W_{ih}, W_{hh}, and W_{ho} are the weight matrices and f_h and f_o are the hidden and output unit activation functions.

The implementation of RNN with 2 Simple RNN layers each with 32 RNN cells followed by time distribute dense layers for 10 class classification can be illustrated as follows:

```
def get_model(_rnn_nb, _fc_nb):

spec_start = Input((256,256))
 spec_x = spec_start
 for _r in _rnn_nb:
 spec_x = SimpleRNN(_r, activation='tanh', dropout=dropout_rate,
recurrent_dropout=dropout_rate, return_sequences=True)(spec_x)

for _f in _fc_nb:
 spec_x = TimeDistributed(Dense(_f))(spec_x)
 spec_x = Dropout(dropout_rate)(spec_x)

spec_x = TimeDistributed(Dense(10))(spec_x)
 out = Activation('sigmoid', name='strong_out')(spec_x)

_model = Model(inputs=spec_start, outputs=out)
 _model.compile(optimizer='Adam', loss='binary_crossentropy',metrics
= ['accuracy'])
 _model.summary()
 return _model
```

The parameters:

```
rnn_nb = [32, 32] # Number of RNN nodes. Length of rnn_nb = number
of RNN layers
fc_nb = [32] # Number of FC nodes. Length of fc_nb = number of FC
layers
dropout_rate = 0.5 # Dropout after each layer
```

The model summary is as follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 256, 256)          0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 256, 32)           9248
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 256, 32)           2080
_________________________________________________________________
time_distributed_4 (TimeDist (None, 256, 32)           1056
_________________________________________________________________
dropout_3 (Dropout)          (None, 256, 32)           0
_________________________________________________________________
time_distributed_5 (TimeDist (None, 256, 10)           330
_________________________________________________________________
strong_out (Activation)      (None, 256, 10)           0
=================================================================
Total params: 12,714
Trainable params: 12,714
Non-trainable params: 0
```

Summary.

LSTM implementation in Keras.

LSTM, also known as the Long Short Term Memory is an RNN architecture with feedback connections, which enables it to perform or compute anything that a Turing machine can.

A single LSTM unit is composed of a cell, an input gate, an output gate and a forget gate, which facilitates the cell to remember values for an arbitrary amount of time. The gates control the flow of information in and out the LSTM cell.

The hidden state h_t for an LSTM cell can be calculated as follows:

$$
\begin{aligned}
i_t &= \sigma\left(x_t U^i + h_{t-1} W^i\right) \\
f_t &= \sigma\left(x_t U^f + h_{t-1} W^f\right) \\
o_t &= \sigma\left(x_t U^o + h_{t-1} W^o\right) \\
\tilde{C}_t &= \tanh\left(x_t U^g + h_{t-1} W^g\right)
\end{aligned}
$$

$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Here, ***i, f, o*** are called the input, forget and output gates, respectively. Note that they have the exact same equations, just with different parameter matrices (***W*** is the recurrent connection at the previous hidden layer and current hidden layer, ***U*** is the weight matrix connecting the inputs to the current hidden layer).

The Keras implementation of LSTM with 2 layers of 32 LSTM cells each for the above-mentioned task of 10 class classification can be illustrated as follows:

```
def get_model(_rnn_nb, _fc_nb):

spec_start = Input((256,256))
 spec_x = spec_start
 for _r in _rnn_nb:
 spec_x = LSTM(_r, activation='tanh', dropout=dropout_rate,
recurrent_dropout=dropout_rate, return_sequences=True)(spec_x)

for _f in _fc_nb:
 spec_x = TimeDistributed(Dense(_f))(spec_x)
 spec_x = Dropout(dropout_rate)(spec_x)

spec_x = TimeDistributed(Dense(10))(spec_x)
 out = Activation('sigmoid', name='strong_out')(spec_x)

_model = Model(inputs=spec_start, outputs=out)
 _model.compile(optimizer='Adam', loss='binary_crossentropy',metrics
= ['accuracy'])
 _model.summary()
 return _model
```

The parameters:

```
rnn_nb = [32, 32] # Number of RNN nodes. Length of rnn_nb = number
of RNN layers
fc_nb = [32] # Number of FC nodes. Length of fc_nb = number of FC
```

```
layers
dropout_rate = 0.5 # Dropout after each layer
```

The model Summary is as follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 256, 256)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 256, 32)           36992
_________________________________________________________________
lstm_2 (LSTM)                (None, 256, 32)           8320
_________________________________________________________________
time_distributed_6 (TimeDist (None, 256, 32)           1056
_________________________________________________________________
dropout_4 (Dropout)          (None, 256, 32)           0
_________________________________________________________________
time_distributed_7 (TimeDist (None, 256, 10)           330
_________________________________________________________________
strong_out (Activation)      (None, 256, 10)           0
=================================================================
Total params: 46,698
Trainable params: 46,698
Non-trainable params: 0
_________________________________________________________________
```

Summary.

GRU implementation in Keras.

The GRU, known as the Gated Recurrent Unit is an RNN architecture, which is similar to LSTM units. The GRU comprises of the reset gate and the update gate instead of the input, output and forget gate of the LSTM.

The reset gate determines how to combine the new input with the previous memory, and the update gate defines how much of the previous memory to keep around. If we set the reset to all 1's and update gate to all 0's we again arrive at our plain RNN model.

For the GRU the hidden state h_t can be calculated as follows:

$$z_t = \sigma\left(x_t U^z + h_{t-1} W^z\right)$$
$$r_t = \sigma\left(x_t U^r + h_{t-1} W^r\right)$$

$$h_t = \tanh\left(x_t U^n + (r_t * h_{t-1}) W^n\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Here ***r*** is a reset gate, and ***z*** is an update gate.

The implementation of the GRU can be illustrated as follows:

```
def get_model(_rnn_nb, _fc_nb):

spec_start = Input((256,256))
 spec_x = spec_start
 for _r in _rnn_nb:
 spec_x = GRU(_r, activation='tanh', dropout=dropout_rate,
recurrent_dropout=dropout_rate, return_sequences=True)(spec_x)

for _f in _fc_nb:
 spec_x = TimeDistributed(Dense(_f))(spec_x)
 spec_x = Dropout(dropout_rate)(spec_x)

spec_x = TimeDistributed(Dense(10))(spec_x)
 out = Activation('sigmoid', name='strong_out')(spec_x)

_model = Model(inputs=spec_start, outputs=out)
 _model.compile(optimizer='Adam', loss='binary_crossentropy',metrics
= ['accuracy'])
 _model.summary()
 return _model
```

The parameters:

```
rnn_nb = [32, 32] # Number of RNN nodes. Length of rnn_nb = number
of RNN layers
fc_nb = [32] # Number of FC nodes. Length of fc_nb = number of FC
layers
dropout_rate = 0.5 # Dropout after each layer
```

The model Summary:

```
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 256, 256)          0
_________________________________________________________________
gru_5 (GRU)                  (None, 256, 32)           27744
_________________________________________________________________
gru_6 (GRU)                  (None, 256, 32)           6240
_________________________________________________________________
time_distributed_10 (TimeDis (None, 256, 32)           1056
_________________________________________________________________
dropout_6 (Dropout)          (None, 256, 32)           0
_________________________________________________________________
time_distributed_11 (TimeDis (None, 256, 10)           330
_________________________________________________________________
strong_out (Activation)      (None, 256, 10)           0
=================================================================
Total params: 35,370
Trainable params: 35,370
Non-trainable params: 0
_________________________________________________________________
```

Summary.

Comparison of the networks.

- From my experience, **GRUs train faster** and perform better than LSTMs on **less training data** if you are doing language modeling (not sure about other tasks).
- **GRUs are simpler** and thus easier to modify, for example adding new gates in case of additional input to the network. It's just less code in general.
- **LSTMs** should, in theory, **remember longer sequences** than GRUs and outperform them in tasks requiring modeling long-distance relations.
- The **GRUs** also have less parameter complexity than LSTM which can be seen from the model summaries above.
- The simple RNNs only have simple recurrent operations without any gates to control the flow of information among the cells.