Deep Learning and Recurrent Neural Networks

LSTM in TensorFlow

Angelo Porrello, Davide Abati

December 5, 2018

University of Modena and Reggio Emilia

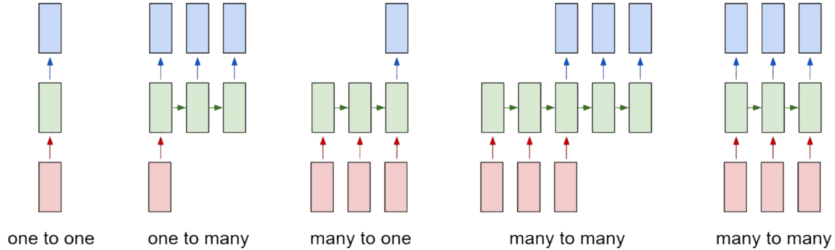
Agenda

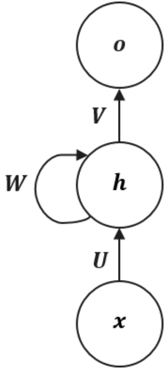
Introduction

Recurrent Neural Networks

Recurrent neural networks (RNN) are **specialized for processing sequences**. Similarly, we saw that convolutional neural networks feature specialized architecture for processing images.

RNNs boast a **much wider API with respect to feedforward neural networks**. Indeed, these models can deal with *sequences* in the input, in the output or even both.



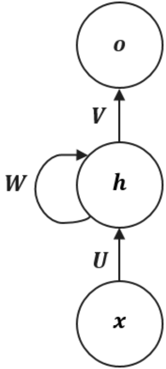


The vanilla RNN is provided with three sets of parameters:

- $\boldsymbol{U}$ maps inputs to the hidden state
- $\boldsymbol{W}$ parametrizes hidden state transition
- $\boldsymbol{V}$ maps hidden state to output

System dynamics is as simple as:

$$\begin{cases} \boldsymbol{h}^{(t)} = \phi(\boldsymbol{W}\,\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\,\boldsymbol{x}^{(t)}) \\ \boldsymbol{o}^{(t)} = \boldsymbol{V}\,\boldsymbol{h}^{(t)} \end{cases} \tag{1}$$



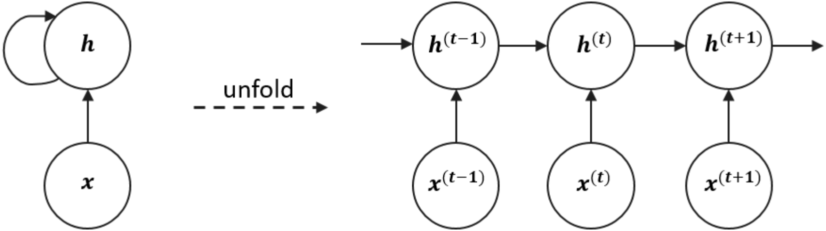
The hidden state $\boldsymbol{h}^{(t)}$ can be intuitively viewed as a *lossy* summary of the sequence of past inputs fed to the network, in which are stored the main task-relevant aspects of the past sequence of inputs up to time t.

Since the an input sequence of arbitrary length $(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)})$ is mapped into a fixed size vector $\boldsymbol{h}^{(t)}$, this summary is necessarily lossy.

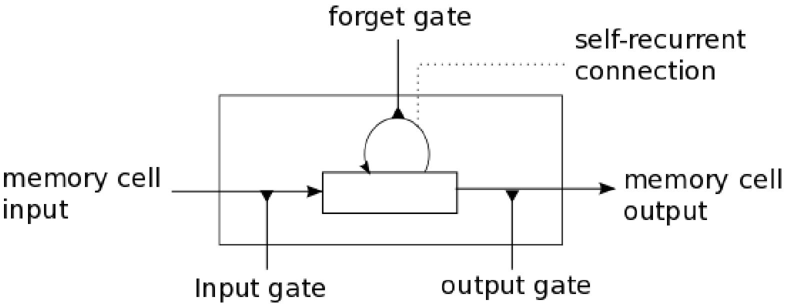
Unfolding the Computational Graph

A recurrent computational graph can be unfolded into a sequential computational graph with a repetitive structure.

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{t-1}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$



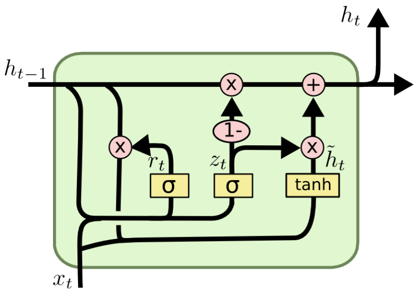
Long Short-Term Memory (LSTM) and **Gated Recurrent Unit (GRU)** are more complex recurrent architectures that have been proposed [2, 1] to overcome the issues in the gradient flow and to ease the learning of long-term dependencies thanks to the introduction of **learnable gating mechanisms**.



Let's see **how update equations look like for a LSTM model**. Please notice that LSTM framework, notation is usually slightly different than form vanilla RNN.

$$\begin{cases} \boldsymbol{i} = \sigma(\boldsymbol{x}^{(t)}\boldsymbol{U}_i + \boldsymbol{s}^{(t-1)}\boldsymbol{W}_i) \\ \boldsymbol{f} = \sigma(\boldsymbol{x}^{(t)}\boldsymbol{U}_f + \boldsymbol{s}^{(t-1)}\boldsymbol{W}_f) \\ \boldsymbol{o} = \sigma(\boldsymbol{x}^{(t)}\boldsymbol{U}_o + \boldsymbol{s}^{(t-1)}\boldsymbol{W}_o) \\ \boldsymbol{g} = \tanh(\boldsymbol{x}^{(t)}\boldsymbol{U}_g + \boldsymbol{s}^{(t-1)}\boldsymbol{W}_g) \\ \boldsymbol{c}^{(t)} = \boldsymbol{c}^{(t-1)} \odot \boldsymbol{f} + \boldsymbol{g} \odot \boldsymbol{i} \\ \boldsymbol{s}^{(t)} = \tanh(\boldsymbol{c}^{(t)}) \odot \boldsymbol{o} \end{cases} \tag{2}$$

Here $\odot$ denotes element-wise multiplication.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

See http://colah.github.io/posts/2015-08-Understanding-LSTMs/ for an exhaustive explanation concerning LSTM and GRU networks.

LSTM in TensorFlow

Cell Support

An RNN cell (`RNNCell`), in the most abstract setting, is anything that has a state and performs some operation that takes a matrix of inputs.

- `BasicRNNCell`: The most basic RNN cell.
- `RNNCell`:Abstract object representing an RNN cell.
- `BasicLSTMCell`: Basic LSTM recurrent network cell.
- `LSTMCell`: LSTM recurrent network cell.
- `GRUCell`: Gated Recurrent Unit cell

HOWTO Construct Cells

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- `cell = tf.nn.rnn_cell.GRUCell(hidden_size, ...)`
- `outputs, state = tf.nn.dynamic_rnn(cell, x, ...)`: uses a `tf.While` loop to dynamically construct the graph when it is executed. Graph creation is faster and you can feed batches of variable size.

Synthetic Sequence Dataset

Synthetic Sequence Dataset

For this practice I prepared a **synthetic dataset** consisting in $\mathbf{2^{20}}$ **binary sequences**.

For each input sequence, the target is the number of ones in the sequence.

From an implementation standpoint, the target is encoded as one-hot vector. Thus, examples (x, y) from the dataset looks like the following:

input	target
00110011111000111101	000000000000100000000
01000010100001010000	000001000000000000000
11101110010111011110	000000000000001000000

The dataset can be found in `synthetic_dataset.py`.

Loading the data is as simple as:

```
from synthetic_dataset import SyntheticSequenceDataset
synthetic_dataset    = SyntheticSequenceDataset()
```

Synthetic data are automatically either generated or loaded from cache (if existent) the first time that dataset property `data` is accessed.

Learning to Count

Goal

Our task is to **count the number of ones in the binary sequences**.

The goal of this practice is to implement and train a **LSTM** [2] network to do so.

Useful Functions

To this purpose, you may find useful the following functions:

- `tf.contrib.rnn.LSTMCell`
- `tf.nn.dynamic_rnn`
- `tf.transpose`
- `tf.gather`
- `tf.layers.dense`

Please refer to the docs to know the exact API.

Good Luck!

References

[1] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio.
Learning phrase representations using rnn encoder-decoder for statistical machine translation.
arXiv preprint arXiv:1406.1078, 2014.

[2] S. Hochreiter and J. Schmidhuber.
Long short-term memory.
Neural computation, 9(8):1735–1780, 1997.