Long Short Term Memory Neural Networks

Short Overview and Examples

Ralph Schlosser

https://github.com/bwv988

February 2018

Overview

Agenda

- RNN
- Vanishing / Exploding Gradient Problem
- LSTM
- Keras
- Outlook
- Demo

Links

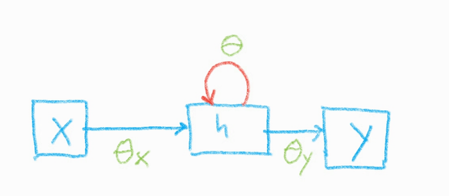
- Git repo: `https://github.com/bwv988/lstm-neural-net-tests`
- Demo: `https://www.kaggle.com/ternaryrealm/lstm-time-series-explorations-with-keras`

RNN

- Recurrent Neural Networks (RNN) are an extension to traditional feed forward NN.
- Original application: Sequence data, e.g.:
  - Music, video
  - Words in a sentence
  - Financial data
  - Image patterns
- Main advantage over traditional (D)NNs: Can **retain state** over a period of time.
- There are other tools to model sequence data, e.g. Hidden Markov Models.
- But: Becomes computationally unfeasible for modelling large time dependencies.
- Today, RNNs often outperform classical sequence models.

Elements of a simple RNN

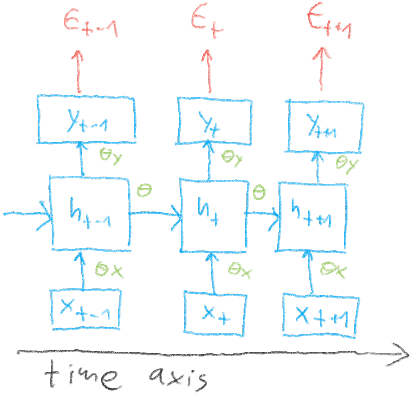
- Input layer: x with weight θ_x.
- Hidden, recursive layer (feeds back into itself): h with weight θ.
- Output layer: y with weight θ_y.
- Arbitrary (e.g. RELU) activation function $\phi(\cdot)$.



$$h_t = \theta\phi(h_{t-1}) + \theta_x x_t$$
$$y_t = \theta_y \phi(h_t)$$

Unrolling the Recursion

We can see how this is applicable to sequence data when unrolling the recursion:



Vanishing / Exploding Gradient Problem

- Training the RNN means: Perform **backpropagation** to find optimal weights.
- Need to minimize the error (or loss) function E wrt., say, parameter θ.
- Optimization problem for S steps:

$$\frac{\partial E}{\partial \theta} = \sum_{t=1}^{S} \frac{\partial E_t}{\partial \theta}$$

- Applying the chain rule gives that for a particular time step t, and looking at θ_k happening in layer k:

$$\frac{\partial E_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial \theta}$$

Vanishing / Exploding Gradient Problem

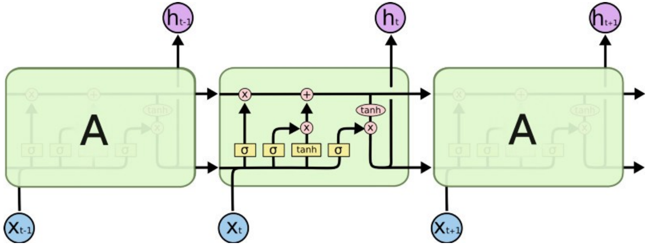
- The issue is with the term $\frac{\partial h_t}{\partial h_k}$.
- Further maths shows (omitting many, many details):

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq c^{t-k}$$

- Here: c is some constant term related to θ and the choice of the activation function ϕ.
- Problem:
  - $c < 1$: Gradients tend to zero (vanish).
  - $c > 1$: Gradients will tend to infinity (explode).
- Impact of vanishing gradients to RNN: Can't "remember" impacts of long sequences.

LSTM

- Variant of RNNs that introduce a number of special, internal gates.
- Internal gates help with the problem of learning relationships between both long and short sequences in data.
- Con: Introduces many more internal parameters which must be learned.
  – **Time consuming**
- Pro: Introduces many more internal parameters which must be learned.
  – **Flexible**



Source: https://blog.statsbot.co/time-series-prediction-using-recurrent-neural-networks-lstms-807fa6ca7f

LSTM Gates

- Input gate i:
  - Takes previous output h_{t-1} and current input x_t.
  - $i_t \in (0,1)$
  - $i_t = \sigma(\theta_{xi} x_t + \theta_{ht} h_{t-1} + b_i)$
- Forget gate f:
  - Takes previous output h_{t-1} and current input x_t.
  - $f_t \in (0,1)$
  - $f_t = \sigma(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f)$
  - If $f_t = 0$: **Forget** previous state, otherwise pass through prev. state.
- Read gate g:
  - Takes previous output h_{t-1} and current input x_t.
  - $g_t \in (0,1)$
  - $g_t = \sigma(\theta_{xg} x_t + \theta_{hg} h_{t-1} + b_g)$

LSTM Gates

- Cell gate c:
  - New value depends on f_t, its previous state c_{t-1}, and the read gate g_t.
  - Element-wise multiplication: $c_t = f_t \odot c_{t-1} + i_t \odot g_t$.
  - We can learn whether to **store** or **erase** the old cell value.
- Output gate o:
  - $o_t = \sigma(\theta_{xo} x_t + \theta_{ho} h_{t-1} + b_o)$
  - $o_t \in (0, 1)$
- New output gate h:
  - $h_t = o_t \odot \tanh(c_t)$
  - Will be fed as input into next block.
- **Intuition**:
  - We learn when to retain a state, or when to forget it.
  - Parameters are constantly updated as new data arrives.

Practical Part

Let's see this in action *sans* some of the more technical details. ;)

The practical examples are based on Keras: `https://keras.io/`

First a few words on Keras.

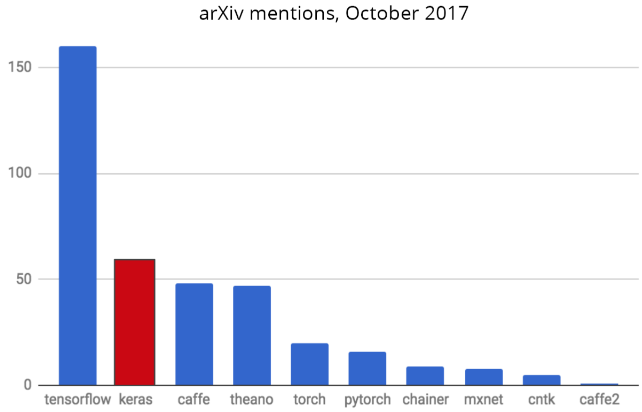
Keras

- Consistent and simple high-level APIs for Deep Learning in Python.
- Focus on getting stuff done w/o having to write tons of lines of code.
- Fantastic documentation!
- Has abstraction layer for multiple Deep Learning backends:
  - Tensorflow
  - CNTK
  - Theano (has reached its final release)
  - mxnet (experimental?)
- Comparable in its ease of use to `sklearn` in Python, or `mlr` in R.

Keras Runs Everywhere

- On iOS, via Apple's CoreML (Keras support officially provided by Apple).
- On Android, via the TensorFlow Android runtime. Example: Not Hotdog app.
- In the browser, via GPU-accelerated JavaScript runtimes such as Keras.js and WebDNN.
- On Google Cloud, via TensorFlow-Serving.
- In a Python webapp backend (such as a Flask app).
- On the JVM, via DL4J model import provided by SkyMind.
- On Raspberry Pi.

Keras & Ranking in the ML Community

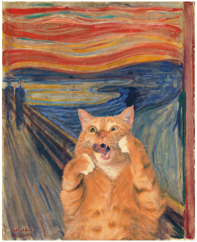


Source: https://keras.io/why-use-keras/

Outlook

- Some interesting, more recent advances with LSTM.
- LSTMs are *Turing-complete*.
- As a result: Can produce any output a human-made computer program could produce, given sufficient units and weights (and of course time, money, computational power).
- DNNs are often called universal function approximators; LSTMs are universal program approximators.

O M G



Is the end of human-made software nigh???? ;)

Neural Turing Machines: LSTMs and other techniques can be leveraged to learn (as of yet simple) algorithms from data:

https://arxiv.org/pdf/1410.5401.pdf

Demo

Let's run this one on Kaggle:

- `https://www.kaggle.com/ternaryrealm/lstm-time-series-explorations-with-keras`

References

- Main source for this presentation – Nando de Freitas brilliant lecture: https://www.youtube.com/watch?v=56TYLaQN4N8
- Ilya Sutskever PhD thesis: http://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf
- "A Critical Review of Recurrent Neural Networks for Sequence Learning": https://arxiv.org/abs/1506.00019
- Why RNNs are difficult to train: https://arxiv.org/pdf/1211.5063.pdf
- Original LSTM paper: https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735
- Keras documentation: https://keras.io/
- Nice blog post explaining LSTMs: https://blog.statsbot.co/time-series-prediction-using-recurrent-neural-networks-ls