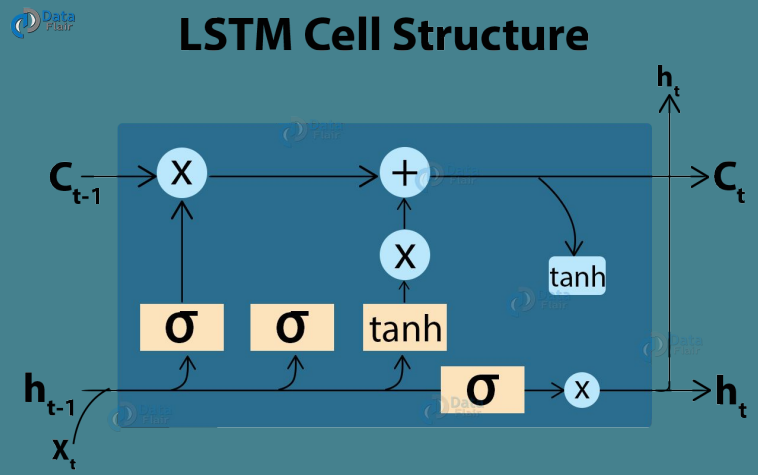
LONG SHORT-TERM MEMORY



Group 01

Thisum (E/15/315)
Ayesh (E/15/073)
Tharindu (E/15/181)
Dinelka (E/15/281)

Based on:

Paper: Long Short-term Memory

Neural Computation, 9(8):1735-1780, 1997

Authors:

Sepp Hochreiter

Jürgen Schmidhuber

Overview

Introduction

Recurrent Neural Networks

Previous Work

Problem Identification

Long Short-term Memory

LSTM’s Limitations and Advantages

LSTM contributions

Introduction

- This paper reviews a problem analyzed by Hochreiter (1991) with Recurrent Neural Networks.

- **The Problem:** In real time recurrent learning, the error signals “flowing backward in time” tend to either (1) Blow up or (2) vanish.

- **The Remedy:** “Long Short-term Memory”(LSTM) a novel recurrent network architecture in conjunction with an appropriate gradient-based learning algorithm.

Recurrent Neural Networks

- A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data.

- These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning.

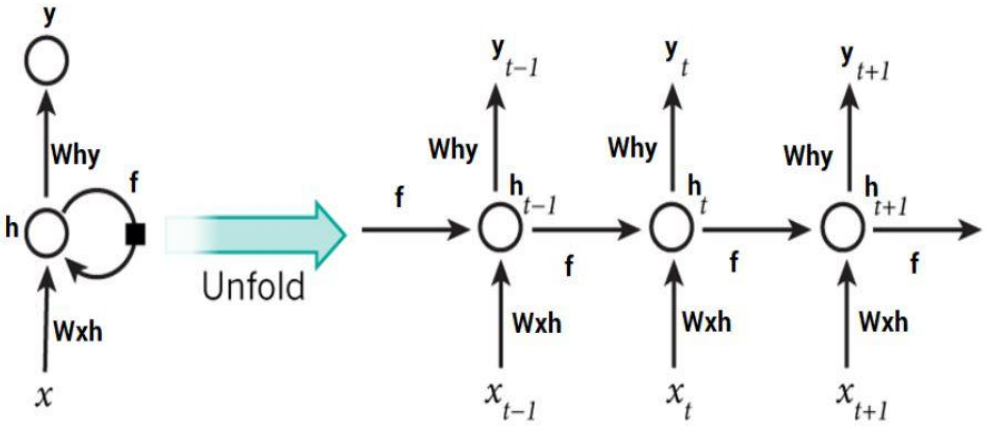
- They are incorporated into popular applications such as Siri, voice search, and Google Translate.

Basic Structure of a Simpler RNN(without the bias term):

Formula for calculating current state:

$$h_t = f(h_{t-1}, x_t)$$

h_t - current state
h_t-1 - previous state
x_t - input state



Formula for applying Activation function(tanh):

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

W_hh - Weight at recurrent neuron
W_xh - Weight at input neuron

Formula for calculating output:

$$y_t = W_{hy}h_t$$

y_t - output
W_hy - Weight at output layer

Training of a simpler RNN

- A single time step of the input is provided to the network.
- Then calculate its current state using set of current input and the previous state.
- The current *ht* becomes *ht-1* for the next time step.
- One can go as many time steps according to the problem and join the information from all the previous states.
- One can calculate the output in each single time step or once all the time steps are completed the final current state can be used to calculate the output.
- The output is then compared to the actual output, the target output and the error is generated.
- The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

The principal of Backpropagation

- Any network structure can be trained with backpropagation when desired output patterns exist and each function, that has been used to calculate the actual output patterns is differentiable.

- As with conventional gradient descent (or ascent), backpropagation works by, for each modifiable weight, calculating the gradient of a cost (or error) function with respect to the weight and then adjusting it accordingly.

- A simple recurrent network (SRN; (Elman, 1990)) has activation feedback which embodies short-term memory. A state layer is updated not only with the external input of the network but also with activation from the previous forward propagation.

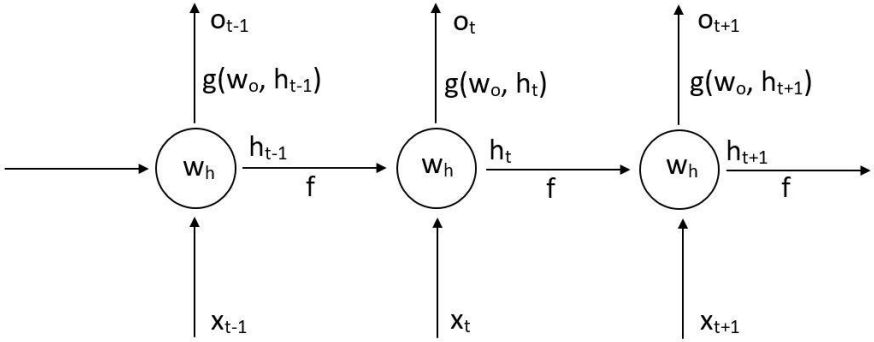
- The feedback is modified by a set of weights as to enable automatic adaptation through learning(backpropagation)

Backpropagation Through Time(BPTT)

Hidden state and Output functions of simple RNN:

$$h_t = f(x_t, h_{t-1}, w_h),$$
$$o_t = g(h_t, w_o),$$

x_t - Input state
h_t - Hidden state
h_t-1 - Previous state
w_h - Weight of hidden layer
o_t - Output state
w_o - Weight of output layer



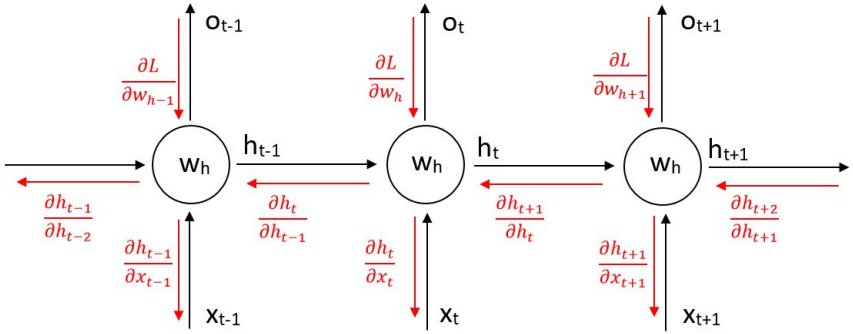
Total Loss across all the T time steps:

$$L(x_1, \ldots, x_T, y_1, \ldots, y_T, w_h, w_o) = \frac{1}{T} \sum_{t=1}^{T} l(y_t, o_t).$$

L - Total Loss
y_t - Predicted output
l - Loss function #cross entropy loss or summed squared error
T - Total Time

Gradient with regard to parameters w_h, of the Loss function L:

$$\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l(y_t, o_t)}{\partial w_h}$$
$$= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_h)}{\partial h_t}\frac{\partial h_t}{\partial w_h}.$$



h_t depends on both h_t-1 and w_h, where computation of h_t-1 also depends on w_h, thus using the chain rule yields,

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial w_h}.$$

Satisfying a_0 = 0 and a_t= b_t + c_t.a_t-1, Let's assume,

$$a_t = b_t + \sum_{i=1}^{t-1}\left(\prod_{j=i+1}^{t} c_j\right) b_i.$$

here;

$$a_t = \frac{\partial h_t}{\partial w_h},$$
$$b_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h},$$
$$c_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}},$$

Finally,

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1}\left(\prod_{j=i+1}^{t}\frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}}\right)\frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}.$$

**recursively, this chain get very long whenever t is large*

Previous Work

This section will focus on recurrent nets with time-varying inputs:

- Gradient-descent variants
  - Elman (1988), Fahlman (1991), Williams (1989), Schmidhuber (1992a), Pearlmutter (1989)
  - Suffer from the same problems as BPTT and RTRL
- Time-delays
  - Lang et al. (1990), Plate(1993), deVries and Principe (1991)
  - Practical for short time lags only
- Time constants
  - Mozer (1992), deVries and Principe's(1991), Sun et al.'s alternative approach (1993)
  - Deal with long time lags
  - The net input, tends to perturb the stored information, which makes long-term storage impractical.
- Ring's Approach
  - Ring (1993)
  - Proposed a method for bridging long time lags
  - This approach can sometimes be extremely fast

- Bengio et al.'s approaches
  - Bengio et al. (1994)
  - investigate methods such as simulated annealing, multi-grid random search, time-weighted pseudo-Newton optimization, and discrete error propagation.
- Kalman filters
  - Puskorius and Feldkamp (1994)
  - Use Kalman filter techniques to improve recurrent net performance
  - Not useful for very long minimal time lags.
- Second order nets
  - Watrous and Kuhn (1992)
  - Use multiplicative units (MUs)
  - Watrous and Kuhn's architecture does not enforce constant error flow and is not designed to solve long time lag problems.
  - It has fully connected second-order sigma-pi units, while the LSTM architecture's MUs are used only to gate access to constant error flow.

- Simple weight guessing
  - Schmidhuber and Hochreiter (1996), Hochreiter and Schmidhuber (1996, 1997), Bengio (1994), Bengio and Frasconi (1994), Miller and Giles (1993), Lin et al. (1995)
  - Simply randomly initialization of all network weights until the resulting net happens to classify all training sequences correctly
  - Weight guessing is a good algorithm for the problems, that are very simple
- Adaptive sequence chunkers
  - Schmidhuber(1992b, 1993), Mozer (1992)
  - Do have a capability to bridge arbitrary time lags, but only if there is local predictability across the subsequences causing the time lags
  - The performance of chunker systems, however, deteriorates as the noise level increases and the input sequences become less compressible.(LSTM does not suffer from this problem)

Problem Identification

Exponentially Decaying Error

Conventional BPTT (e.g. Williams and Zipser 1992). Output unit k's target at time t is denoted by $d_k(t)$. Using mean squared error, k's error signal is

$$\vartheta_k(t) = f'_k(net_k(t))(d_k(t) - y^k(t)),$$

where

$$y^i(t) = f_i(net_i(t))$$

is the activation of a non-input unit i with differentiable activation function f_i,

$$net_i(t) = \sum_j w_{ij} y^j(t-1)$$

is unit i's current net input, and w_{ij} is the weight on the connection from unit j to i. Some non-output unit j's backpropagated error signal is

$$\vartheta_j(t) = f'_j(net_j(t)) \sum_i w_{ij} \vartheta_i(t+1).$$

From: LSTM

Neural Computation 9(8):1735{1780, 1997

Page 3

Outline of Hochreiter Analysis:

Suppose we have a fully connected net whose non-input unit indices range from 1 to *n*. Let us focus on local error flow from unit *u* to unit *v*. The error occurring at an arbitrary unit *u* at time step *t* is propagated “back into time" for *q* time steps, to an arbitrary unit *v*. This will scale the error by the following factor:

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \begin{cases} f'_v(net_v(t-1))w_{uv} & q=1 \\ f'_v(net_v(t-q)) \sum_{l=1}^{n} \frac{\partial \vartheta_l(t-q+1)}{\partial \vartheta_u(t)} w_{lv} & q>1 \end{cases} .$$

With $l_q=v$ and $l_o=u$, we obtain;

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}}$$

The sum of the n^{q-1} terms, $\prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}}$ determines the Total error backflow.

Considering this equation:

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}}$$

Gradient Blow up:

If,

$$|f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}}| > 1.0$$

For all *m*, then the largest product **increases exponentially** with *q*. That is, the error blows up, and conflicting error signals arriving at unit v can lead to oscillating weights and unstable learning. **Gradient clipping** can be identified as a solution for this problem.

Vanishing Gradient:

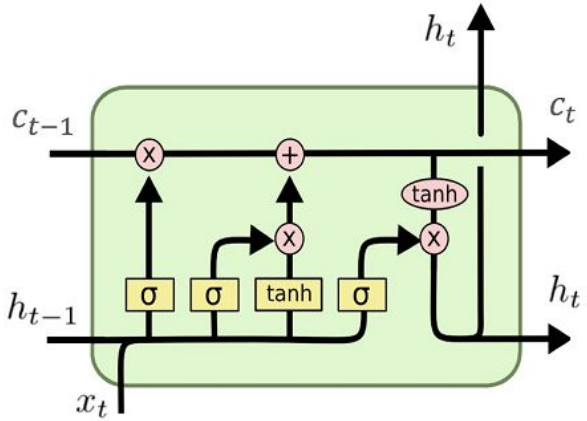
If, $$|f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}}| < 1.0$$

for all m, then the largest product **decreases exponentially** with *q*. That is, the error vanishes, and nothing can be learned in acceptable time.

Long Short-Term Memory

Memory cells and gate units: The target is to construct an architecture that allows for constant error flow through special, self-connected units.

- Input to the LSTM
- Hidden state
- Cell state
- Forget gate
- Input gate
- Output gate
- Output of the LSTM
- Weights and Biases



LSTM
(Long-Short Term Memory)

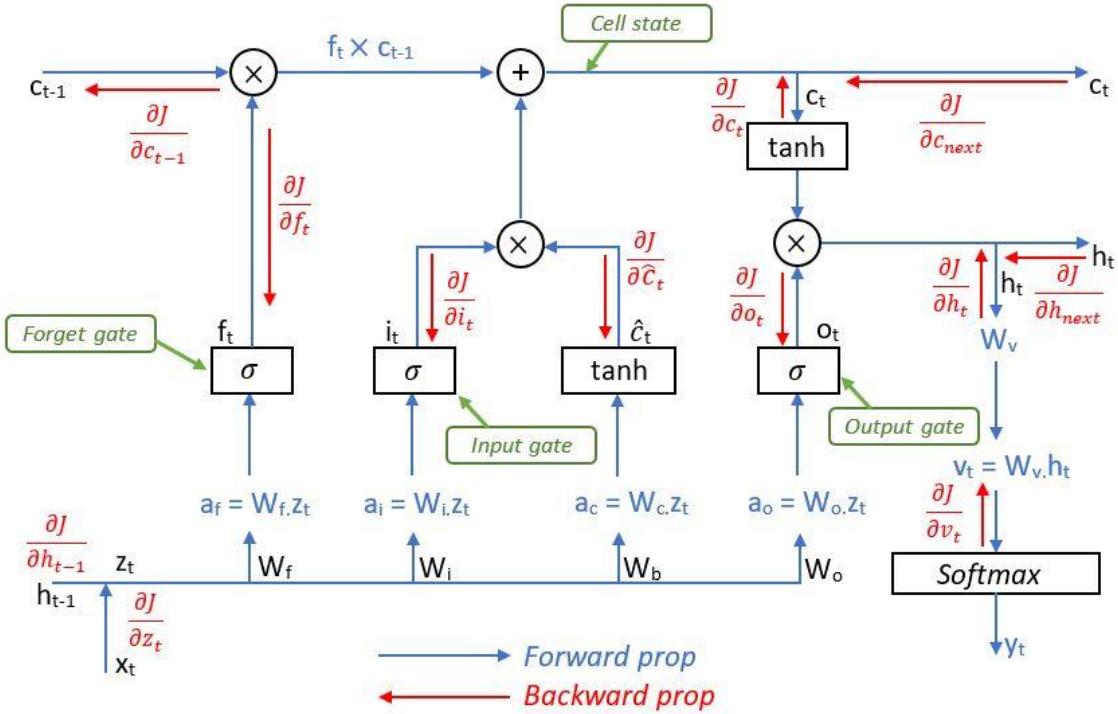
Detailed Representation of LSTM Cell:

Forget Gate: Decides what information is to be thrown away or kept from the last step

Input Gate: Decides what information is going to be stored in the cell state

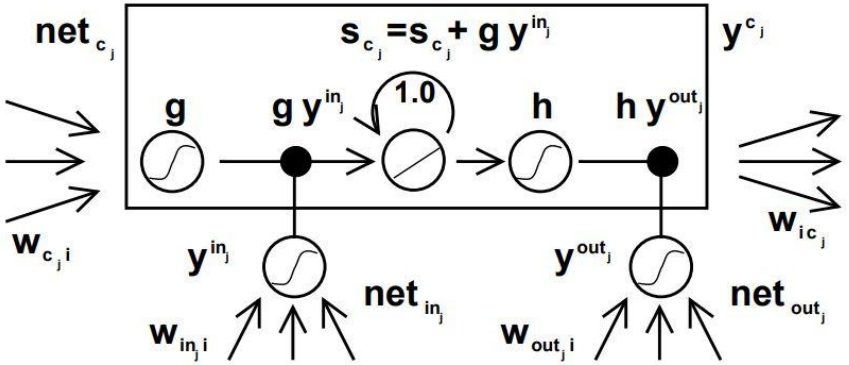
Cell State: Serves as the memory of an LSTM

Output Gate: Outputs a filtered cell state values



LSTM Architecture:

- Network Topology
- Memory cell and blocks
- Learning



Refer: LSTM 1997, Page 7

LSTM’s Limitations and Advantages

Limitations of LSTM

- Efficient truncated backprop version of the LSTM algorithm will not easily solve problems similar to “strongly delayed XOR problems".
- Each memory cell block needs two additional units increasing the number of weights.
- Problems due to its constant error flow through CECs within memory cells
- Practical inability to precisely count discrete time steps

Advantages of LSTM

- LSTM's ability to bridge very long time lags.
- LSTM can handle noise, distributed representations, and continuous values.
- The best remedy for the problems discussed above
- No need for parameter fine tuning
- The LSTM algorithm's update complexity per weight and time step is essentially that of BPTT(backpropagation through time)

Contributions

- **2009:** An LSTM based model won the **ICDAR** connected handwriting recognition competition. Three such models were submitted by a team lead by **Alex Graves**. One was the most accurate model in the competition and another was the fastest.
- **2013:** LSTM networks were a major component of a network that achieved a record 17.7% **phoneme** error rate on the classic **TIMIT** natural speech dataset.
- **2014:** Kyunghyun Cho et al. put forward a simplified variant called **Gated Recurrent Unit** (GRU).
- **2015:** Google started using an LSTM for speech recognition on Google Voice.According to the official blog post, the new model cut transcription errors by 49%.
- **2016:** Google started using an LSTM to suggest messages in the Allo conversation app.In the same year, Google released the **Google Neural Machine Translation system** for Google Translate which used LSTMs to reduce translation errors by 60%. Apple announced in its Worldwide Developers Conference that it would start using the LSTM for quicktype in the iPhone and for Siri. Amazon released **Polly**, which generates the voices behind Alexa, using a bidirectional LSTM for the text-to-speech technology.

References

- https://www.bioinf.jku.at/publications/older/2604.pdf
- https://d2l.ai/chapter_recurrent-neural-networks/bptt.html#backpropagation-through-time-in-detail
- https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html
- https://towardsdatascience.com/lstm-gradients-b3996e6a0296