Do LSTMs Learn Syntax?

Ted Briscoe

January 16, 2019

Language Modelling

- N-gram based models with smoothing
- Structured language models
- Neural (RNN) Language Models

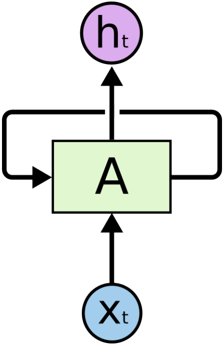
- 1 Billion Words Training Data
- Perplexity, e.g. $2^{bits/word}$, halved by LSTM models
- Word embeddings = word similarity
- Unlimited but selective left context

Recurrent Neural Networks

- Logistic Regression classification via Sigmoid (0 - 1) function; minimize log-likelihood via gradient descent to optimize weights
- Mutilayer Perceptron / Feedforward Networks, non-linear representation learning, iterative gradient-based stochastic optimization (non-convex), backpropagation
- Recurrent Neural Networks, feedforward networks with recurrent connections, variable length sequences, backpropagation through time (BPTT)

Resource:
`https://www.cl.cam.ac.uk/teaching/1819/DataSciII/-materials.html`

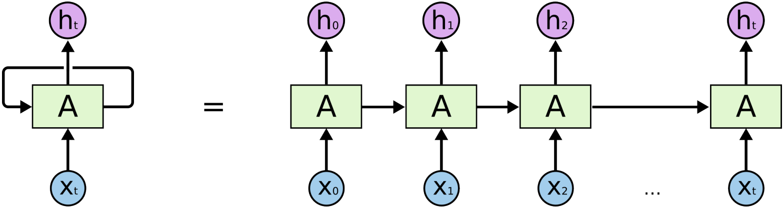
RNN Recurrence



Output becomes part of input at t_{+1}

(All Images courtesy of Christopher Olah's blog, Understanding LSTMs)

RNN Recurrence Unrolled



Same parameters / composition function (matrix + tanh)
different contexts and inputs

BPTT (Simplified)

For each training instance of length k (and each timestep):

1. Forward propagate the inputs through the unfolded network
2. Compute the error
3. Back propagate the derivatives of the error across the unfolded network
4. Sum the weight changes in the k instances of the recurrent units
5. Roll up the network and update all the weights
6. Repeat for each timestep

RNN Language Model Components / Training

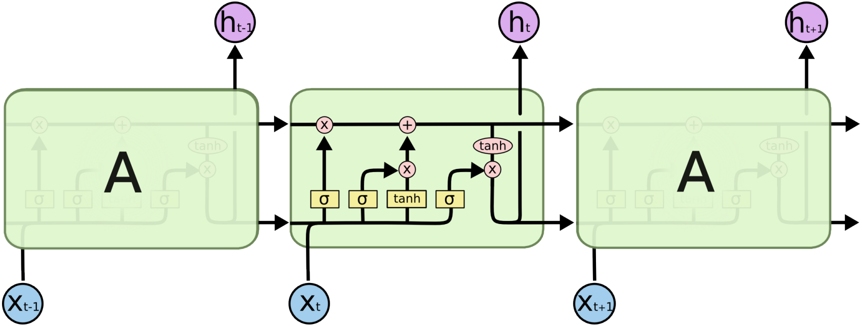
1. Input Word Embeddings – One hot / continous vectors
2. Recurrent Sentence Embedding – Identity Matrix + Sigmoid
3. Output SoftMax (‘Multinomial Sigmoid’) over Vocabulary – One hot vector

1. At each time step forward propagate
2. Train to maximize log of softmax(h_t) (= MLE)
3. BPTT error derivatives to sentence and word embeddings

RNN / BPTT Problems

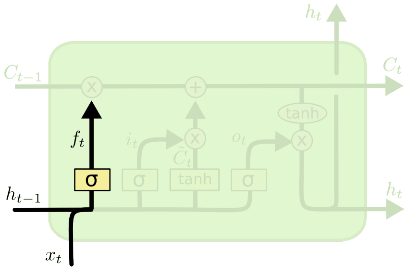
- Recurrent weights >1 network becomes chaotic (truncate weights / gradients)
- Recurrent weights <1 exponential decay (iterated weight multiplication $+$ derivative of saturated tanh/sigmoid)
- Depending on whether trying to model local / global context effects, a single recurrent weight may both increase / decrease during training – weight conflict
- Difficult and slow to train (Elman – starting small)
- Composition function too simple (despite Turing approx.)

Long Short Term Memory



Memory cell no longer the identity matrix + sigmoid/tanh, but now a ‘gated copy’ of the previous hidden state
= more complex composition function

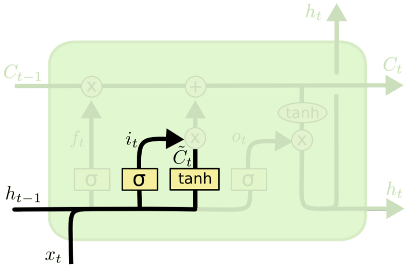
Forget Gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Looks at previous output vector and current input vector and outputs a vector or 0/1s that determines which parts of the context vector are kept

Input Gate / Context Vector

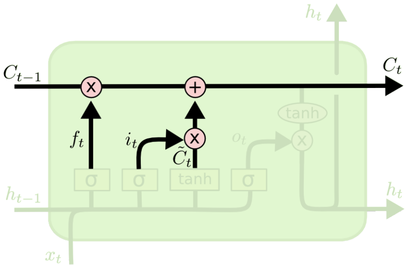


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Input gate same as as forget gate but applied to tanh (-1 - 1) vector of new context vector derived by applying composition function weights to input and previous output

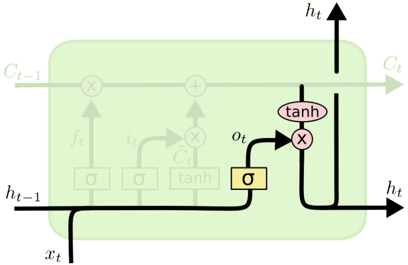
Context Update



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Pointwise add old state vector after forgetting by new context vector after filtering = new state vector

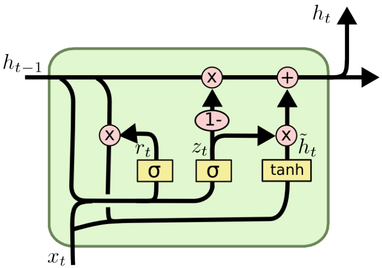
Output Gate and Vector



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Output gate same as other gates then scale (tanh) state vector and pointwise multiply by output vector to derive next hidden / output vector

Gated Recurrent Unit



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

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

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

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

Forget + Input Gates = Update Gate

Learned part of composition function is 3/4 weight vectors

What do LSTM Lg Models Learn?

- Black box so perturb test data to pretrained model
- How much left context used?
- What type of contextual information is used?
- av. 200 previous words (PWs) of context
- 1% increase in perplexity with only 150 PWs of context
- Mostly affects infrequent (content) words
- Word order tracked for about 20 PWs
- Shuffling word order beyond 50 PWs no effect
- Replacing words out to 50 PWs does
- Beyond 20 PWs omitting function words no effect
- Remember target words that have already occured within 50 PWs

Why focus on LSTM Lg Models for the Topic?

- Unsupervised training / Large datasets
- Not directly training for parsing / syntax
- Natural Task(?) Shannon's game
- Sequential to hierarchical – NL is a mostly sequential realisation of a hierarchical 'language of thought'
- Steve Pulman, Wheeler Lecture, 2018 argues that a Recursive Neural Model captures syntax and imposes the right sort of prior inductive bias for hierarchical structure, but such models tend also to be trained on treebanks...

Resource:
`https://www.cl.cam.ac.uk/seminars/wheeler/stephen-pulman/-s`
and more technical version: Pulman-Wheeler-Extra-Slides on Topic page

References

Melis, G., Dyer, C., and Blumson, P. On the State of the Art of Evaluation in Neural Language Models, ICLR, 2018
https://arxiv.org/pdf/1707.05589.pdf

Khandelwal, U., He, H., Qi, P. and Jurafsky, D. Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context, ACL 2018
https://arxiv.org/pdf/1805.04623.pdf