Part of Speech Tagging with LSTM Networks

Project Presentation

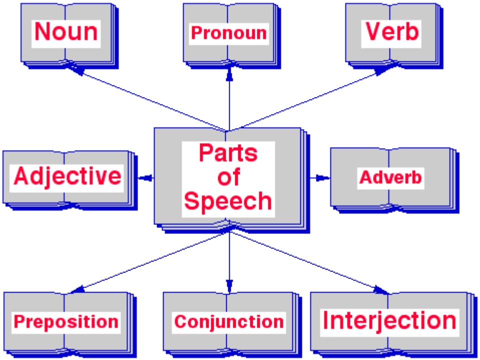
Zeming Lin

Department of Computer Science at University of Virginia

04/30/2015

Table of Contents

Part of Speech



Penn Treebank Dataset

- We use 93915 words, from NLTK. Only consider sentences with length > 4.
- Already tokenized.
- Example:
  - Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 .
  - NNP NNP , CD NNS JJ , MD VB DT NN IN DT JJ NN NNP CD .

State of the art

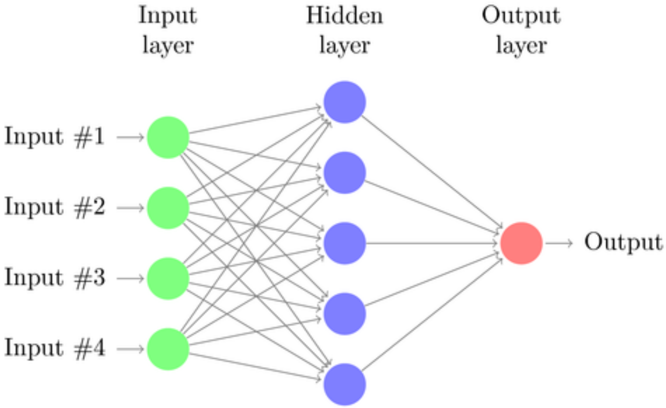
Author	Model	Accuracy
Brants (2000)	Hidden Markov Model	96.46%
Giménez and Márquez (2004)	SVM	97.16%
Spoustová et al. (2009)	Averaged Perceptron	97.44%
Manning (2011)	Dependency Network	97.32%
Søgaard (2011)	Condensed Nearest Neighbors	97.50%

State of the art

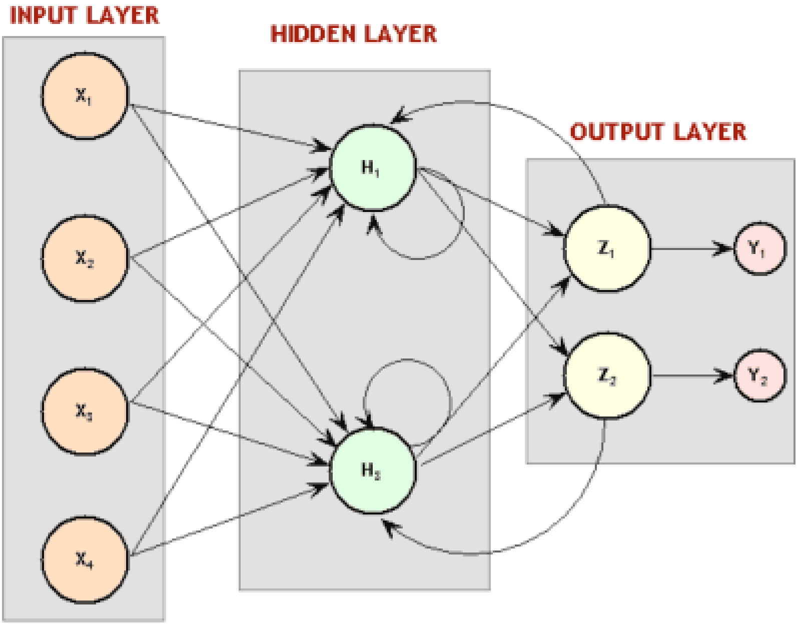
- Human disagreement is ~3.5%
- Why is this interesting?
  - Machines often make very obvious mistakes
  - Single error tends to cascade to downstream modules for NLP

Table of Contents

Neural Networks

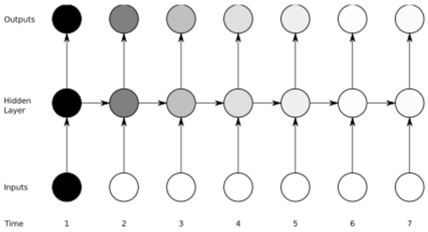


Recurrent Networks



Recurrent networks

- Hard to train!
- Backpropagation through time is used to approximate training



Recurrent Networks

- BPTT algorithm not guaranteed to converge to a *local* minimum
  - Very sensitive to learning rate changes
- Exploding / vanishing gradients

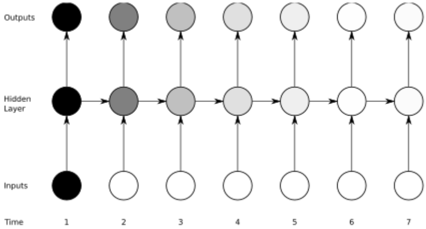
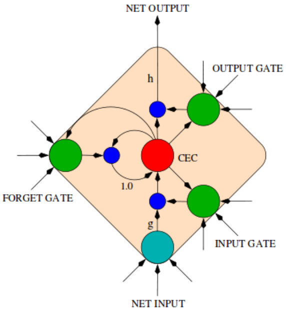


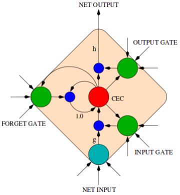
Table of Contents

Long-short term memory

- Fixes the gradients problem, so we can train on longer time steps!
- LSTM Cell:

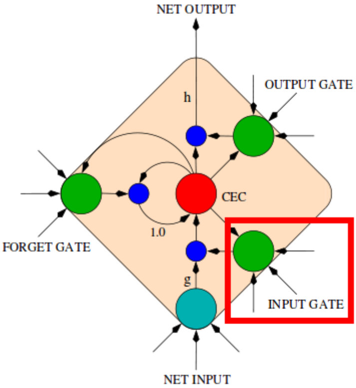


LSTM Cell



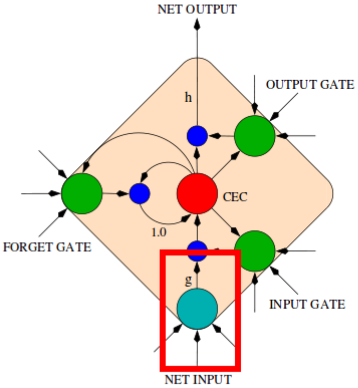
$$
\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
\tilde{C}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
C_t &= i_t \odot \tilde{C}_t + f_t \odot C_{t-1} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + V_o C_t + b_f) \\
h_t &= o_t \odot \tanh C_t
\end{aligned}
$$

LSTM Cell



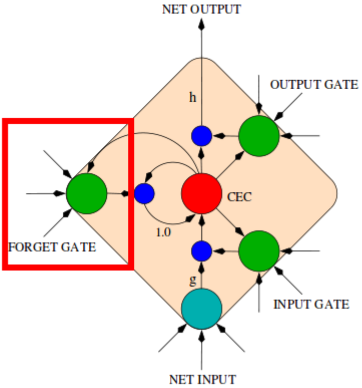
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

LSTM Cell



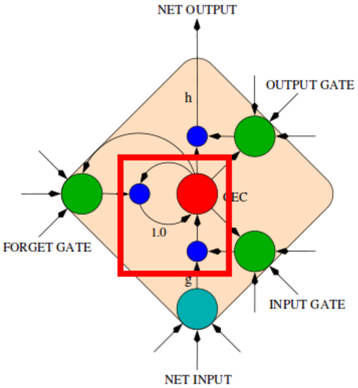
$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

LSTM Cell



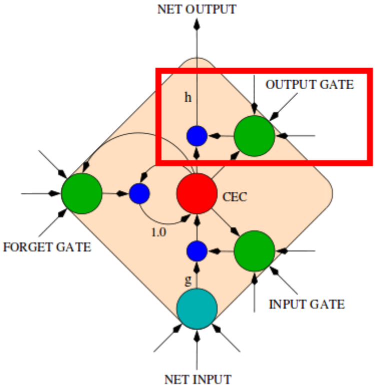
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

LSTM Cell



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

LSTM Cell



$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o C_t + b_f)$$
$$h_t = o_t \odot \tanh C_t$$

LSTM Network

Error gradients no longer vanish / explode!

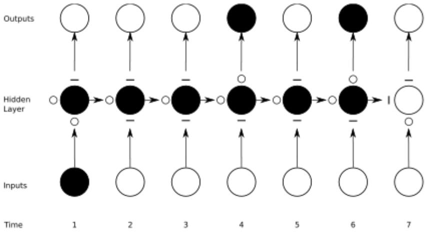
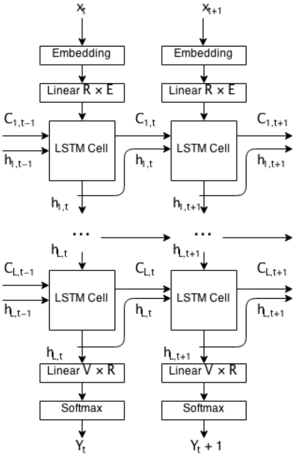


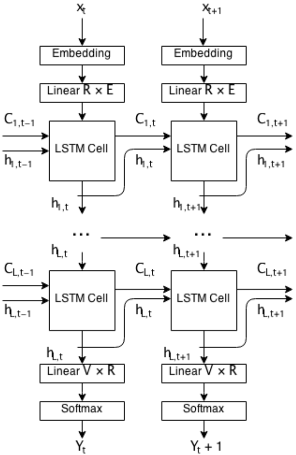
Table of Contents

Layers



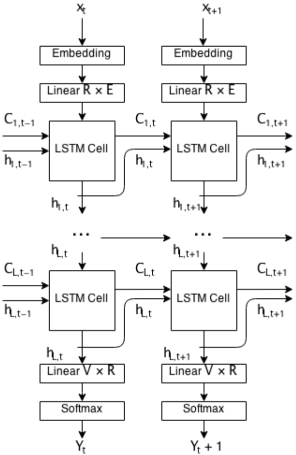
- Embedding is a $E = 50$ dim vector, trained from wikipedia, lookuptable of 130k by 50.

Layers



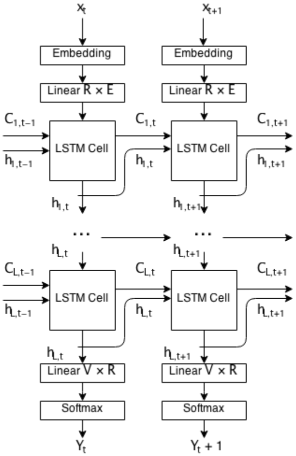
- Embedding is a $E = 50$ dim vector, trained from wikipedia, lookuptable of 130k by 50.
- R is the size of output

Layers



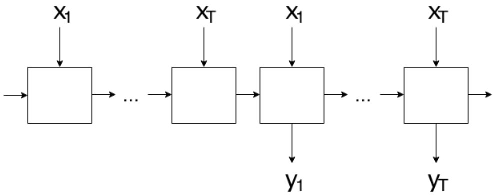
- Embedding is a $E = 50$ dim vector, trained from wikipedia, lookuptable of 130k by 50.
- R is the size of output
- C is the memory of the network, the “error carousel”

Layers



- Embedding is a $E = 50$ dim vector, trained from wikipedia, lookuptable of 130k by 50.
- R is the size of output
- C is the memory of the network, the "error carousel"
- V is number of tags to label, or 46.

Running scheme



- Run sequence through twice: Only consider the second run through
  - "Read entire sequence" before considering POS labels.
  - 2-time slowdown, but ~1-2% extra accuracy

Table of Contents

Results

L	R	T	Accuracy	Speed (wps)
2	100	40	.942	319
2	250	40	.952	88
2	500	40	.953	25
2	100	400	.942	363
2	100	10	.932	394
3	100	40	.936	239
4	100	40	.924	171

- Each network has L layers
- Consider T-length sequences
- Cells memory of R units.

Table of Contents

Discussion

L	R	T	Accuracy	Speed (wps)
2	100	40	.942	319
2	250	40	.952	88
2	500	40	.953	25
2	100	400	.942	363
2	100	10	.932	394
3	100	40	.936	239
4	100	40	.924	171

- More layers == worse performance?
- Increase number of training iterations?

Discussion

L	R	T	Accuracy	Speed (wps)
2	100	40	.942	319
2	250	40	.952	88
2	500	40	.953	25
2	100	400	.942	363
2	100	10	.932	394
3	100	40	.936	239
4	100	40	.924	171

- High T doesn't impact, but low T does
- Memory units R had large impact, $100 \rightarrow 250$ gave 1% accuracy!

Future Work

- Find the full Treebank dataset, see if we get state of the art 97.5% results
- Test larger models, use GPU to parallelize matrix computation
- Batch gradient descent to parallelize training, can use Mapreduce

Thank you!