Named Entity Recognition with Long Short-Term Memory

James Hammerton
Alfa-Informatica, University of Groningen
Groningen, The Netherlands
james@let.rug.nl

Abstract

In this approach to named entity recognition, a recurrent neural network, known as Long Short-Term Memory, is applied. The network is trained to perform 2 passes on each sentence, outputting its decisions on the second pass. The first pass is used to acquire information for disambiguation during the second pass. SARDNET, a self-organising map for sequences is used to generate representations for the lexical items presented to the LSTM network, whilst orthogonal representations are used to represent the part of speech and chunk tags.

1 Introduction

In this paper, Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) is applied to named entity recognition, using data from the Reuters Corpus, English Language, Volume 1, and the European Corpus Initiative Multilingual Corpus 1. LSTM is an architecture and training algorithm for recurrent neural networks (RNNs), capable of remembering information over long time periods during the processing of a sequence.

LSTM was applied to an earlier CoNLL shared task, namely clause identification (Hammerton, 2001) although the performance was significantly below the performance of other methods, e.g. LSTM achieved an fscore of 50.42 on the test data where other systems' fscores ranged from 62.77 to 80.44. However, not all training data was used in training the LSTM networks. Better performance has since been obtained where the complete training set was used (Hammerton, unpublished), yielding an fscore of 64.66 on the test data.

2 Representing lexical items

An efficient method of representing lexical items is needed. Hammerton (2001; unpublished) employed lexical space (Zavrel and Veenstra, 1996) representations of the words which are derived from their co-occurrence statistics. Here, however, a different approach is used. A SARDNET (James and Miikkulainen, 1995), a self-organising map (SOM) for sequences, is trained to form representations of the words and the resulting representations reflect the morphology of the words.

James and Miikkulainen (1995) provide a detailed description of how SARDNET operates. Briefly, the SARDNET operates in a similar manner to the standard SOM. It consists of a set of inputs and a set of map units. Each map unit contains a set of weights equal in size to the number of inputs. When an input is presented, the map unit with the closest weights to the input vector is chosen as the winner. When processing a sequence, this winning unit is taken out of the competition for subsequent inputs. The activation of a winning unit is set at 1 when it is first chosen and then multiplied by a decay factor (here set at 0.9) for subsequent inputs in the sequence. At the beginning of a new sequence all map units are made available again for the first input. Thus, once a sequence of inputs has been presented, the map units activated as winners indicate which inputs were presented and the activation levels of those units indicate the order of presentation. An advantage of SARDNET is that it can generalise naturally to novel words.

The resulting representations are real-valued vectors, reflecting the size of the map layer in the SARDNET (enough to represent words of upto length N where N is the size of the map). A SARDNET was trained over a single presentation of all the distinct words that appear in the training and development data for English and a separate SARDNET was trained on all the distinct words appearing in the training data for German. The generalisation of the map to novel words was just as good with the German map as with the English map, suggesting training on the map only on the English training data words would make little difference to performance. Initially the neighbourhood was set to cover the whole SARDNET and the learning rate was set at 0.4. As each word was presented, the neighbourhood and learning rate were reduced in lin-

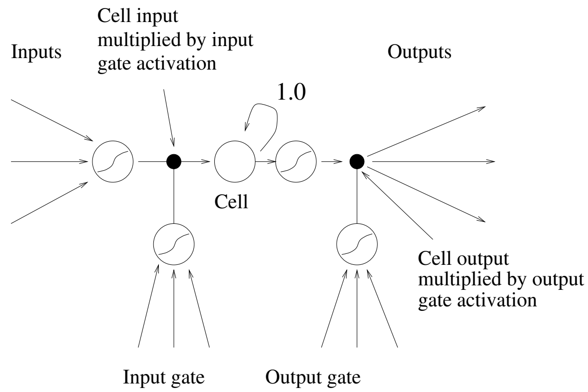

Figure 1: A single-celled memory block

ear increments, so that at the end of training the learning rate was zero and the neighbourhood was 1. Both the English and German experiments used a SARDNET with 64 units.

3 Long Short-Term Memory (LSTM)

An LSTM network consists of 3 layers, an input layer, a recurrent hidden layer and an output layer. The hidden layer in LSTM constitutes the main innovation. It consists of one or more memory blocks each with one or more memory cells. Normally the inputs are connected to all of the cells and gates. The cells are connected to the outputs and the gates are connected to other cells and gates in the hidden layer.

A single-celled memory block is illustrated in Figure 1. The block consists of an input gate, the memory cell and an output gate. The memory cell is a linear unit with self-connection with a weight of value 1. When not receiving any input, the cell maintains its current activation over time. The input to the memory cell is passed through a squashing function and gated (multiplied) by the activation of the input gate. The input gate thus controls the flow of activation into the cell.

The memory cell's output passes through a squashing function before being gated by the output gate activation. Thus the output gate controls the activation flow from cells to outputs. During training the gates learn to open and close in order to let new information into the cells and let the cells influence the outputs. The cells otherwise hold onto information unless new information is accepted by the input gate. Training of LSTM networks proceeds by a fusion of back-propagation through time and real-time recurrent learning, details of which can be found in (Hochreiter and Schmidhuber, 1997).

In artificial tasks LSTM is capable of remembering information for up-to 1000 time-steps. It thus tackles one of the most serious problems affect the performance of recurrent networks on temporal sequence processing tasks.

Tag	Vector Rep.
B-LOC	0 0 1 1 0 0 0
B-MISC	0 0 1 0 1 0 0
B-ORG	0 0 1 0 0 1 0
B-PER	0 0 1 0 0 0 1
I-LOC	1 0 0 1 0 0 0
I-MISC	1 0 0 0 1 0 0
I-ORG	1 0 0 0 0 1 0
I-PER	1 0 0 0 0 0 1
O	0 1 0 0 0 0 0

Table 1: Vector representations used for output tags

LSTM has recently been extended (Gers and Schmidhuber, 2000) to include forget gates which can learn to modify the cell contents directly and peephole connections which connect the cell directly to the gates, thus enabling them to use the cells' contents directly in their decisions. Peephole connections are not used here, but forget gates are used in some experiments.

4 Experiments

The LSTM networks used here were trained as follows:

- Each sentence is presented word by word in two passes. The first pass is used to accumulate information for disambiguation in the second pass. In the second pass the network is trained to output a vector representation (see Table 1) of the relevant output tag. During the first pass the network is just trained to produce "0.1"s at all its outputs. Note that the binary patterns listed in Table 1 are converted to "0.1"s and "0.9"s when used as target patterns. This technique has been found to improve performance.

Net	Hidden	Opts	Wts
Net1	8x6		13543
Net2	8x6	int	13543
Net3	8x6	int,look	18087
Net4	8x6	int,list	13898
Net5	8x6	int,look,list	18442
Net6	8x5	int2,lex	15318
Net7	8x5	int2,lex,FG	15270
Net8	8x5	int2,list,lex,FG	15625

Table 2: Networks used in the experiments here.

- The inputs to the networks are as follows:
  - The SARDNET representations of the current word and optionally the next word (or a null vector if at the end of a sentence).

    With some nets, the lexical space representation of the current word is also used. This involves computing, for each word, the frequencies with which the most frequent 250 words appear either immediately before or immediately after that word in the training set. The resulting 500 element vectors (250 elements each for the left and right context) are normalised then mapped onto their top 25 principal components.

  – An orthogonal representation of the current part of speech (POS) tag. However, for some networks, the input units the POS tag is presented to perform a form of time integration as follows. The units are updated according to the formula $I(t) = 0.5 \times I(t-1) + P(t)$, where $I(0) = 0$, $P(t)$ is the pattern representing the current POS tag, and $t = 1, ..., n$ where n is the length of the current sequence of inputs (twice the length of the current sentence due to the 2 pass processing). By doing this the network receives a representation of the sequence of POS tags presented thus far, integrating these inputs over time.
  – An orthogonal representation of the current chunk tag, though with some networks time integration is performed as described above.
  – One input indicates which pass through the sentence is in progress.
  – Some networks used a list of named entities (NEs) as follows. Some units are set aside corresponding to the category of NE, 1 unit per category. If the current word occurs in a NE, the unit for that NE's category is activated. If the word occurs in more than one NE, the units for all the NEs' categories are activated. In the case of the English data there were 5 categories of NE (though one category "MO" seems to arise from an error in the data).

- The networks were trained with a learning rate of 0.3, no momentum and direct connections from the input to the output layers for 100 iterations. Weight updating occurred after the second pass of each sentence was presented. The best set of weights during training were saved and used for evaluation with the development data.

- The results reported for each network are averaged over 5 runs from different randomised initial weight settings.

Table 2 lists the various networks used in these experiments. The "Net" column lists the names of the networks used. The "Opts" column indicates whether word lists are used (list), a 1 word lookahead is used (look), lexical space vectors are used (lex), whether the units for the POS tags use time integration as described above (int) and whether time integration is performed on both the units for POS tags and the units for chunk tags (int2). Additionally, it indicates whether forget gates were used (FG). The "Hidden" column gives the size of the hidden layer of the network (i.e. 8x6 means 8 blocks of 6 cells). The "Wts" column gives the number of weights used.

Net	Precision	Recall	Fscore	Range
Net1	61.42%	46.64%	52.98	49.16–54.30
Net2	62.42%	49.70%	55.30	53.75–56.92
Net3	62.80%	48.02%	54.41	52.24–55.74
Net4*	*75.27%*	*64.61%*	*69.53*	*68.55–70.60*
Net5*	75.03%	65.13%	69.73	68.05–70.58
Net6	67.92%	57.17%	62.08	59.26–64.14
Net7	68.04%	58.59%	62.95	61.25–64.86
Net8*	**76.37%**	**66.27%**	**70.96**	**69.46–72.88**
Basel.	78.33%	65.23%	71.18	n/a

Table 3: Results of named entity recognition on English development data for networks trained on the English training data. Results are averaged over 5 runs using different initial weights. * indicates use of the list of NEs. Italics indicate best result reported on first submission, whilst bold indicates best result achieved overall.

Table 3 gives the results for extracting named entities from the English development data for the networks. The "Precision", "Recall" and "Fscore" columns show the average scores across 5 runs from different random weight settings. The "Range" column shows the range of fscores produced across the 5 runs used for each network. The Precision gives the percentage of named entities found that were correct, whilst the Recall is the percentage of named entities defined in the data that were found. The Fscore is (2*Precision*Recall)/(Precision+Recall).

Most options boosted performance. The biggest boosts came from the lexical space vectors and the word lists. The use of forget gates improved performance despite leading to fewer weights being used. Lookahead seems to make no significant difference overall. Only Net8 gets above baseline performance (best fscore = 72.88), but the average performance is lower than the baseline.

Table 4 gives the results for the best network broken down by the type of NE for both the English development and testing data. This is from the best performing run for Net8. Table 4 also depicts the best result from 5 runs of a network configured similarly to Net7 above, using the German data. This did not employ a list of NEs and the lemmas in the data were ignored. The fscore of 43.50[1] is almost 13 points higher than the baseline of 30.65. With the German test set the fscore is 47.74, 17 points higher

[1] The average fscore on the German development set was 40.80 and the range was 36.47–43.50.

than the baseline of 30.30.

5 Conclusion

A LSTM network was trained on named entity recognition, yielding an fscore just above the baseline performance on English and significantly above baseline for German. Whilst the just-above-baseline performance for English is disappointing, it is hoped that further work will improve on these results. A number of ways of boosting performance will be looked at including:

- Increasing the size of the hidden layers will increase the power of the networks at the risk of overfitting. Increasing training times may also increase performance, again at the risk of overfitting.

- Increasing the informativeness of the lexical representations. Given that the number of elements used here is less than the number of characters in the character sets, there should be some scope for boosting performance by increasing the size of the SARDNETs. The representations of different words will then become more distinct from each other.

  The lexical space vectors were derived from a context of +/- 1 word, where in earlier work on clause splitting a context of +/- 2 words was used. Using the larger context and/or using more than 25 of the top principal components may again boost performance by incorporating more information into the vectors.

- Further exploitation of the word lists. Whilst the networks are made aware of which categories of named entity the current word can belong to, it is not made aware of how many named entities it belongs to or of what positions on the named entities it could occupy.

Acknowledgements

The LSTM code used here is a modified version of code provided by Fred Cummins. The training of the SARDNETs was done using the PDP++ neural network simulator (`http://www.cnbc.cmu.edu/Resources/PDP++/PDP++.html`).

This work is supported by the Connectionist Language Learning Project of the High Performance Computing/Visualisation centre of the University of Groningen.

English devel.	Precision	Recall	$F_{\beta=1}$
LOC	88.17%	82.80%	85.40
MISC	83.56%	74.95%	79.02
ORG	71.83%	62.19%	66.67
PER	70.65%	52.93%	60.52
Overall	78.95%	67.67%	72.88

English test	Precision	Recall	$F_{\beta=1}$
LOC	79.41%	78.60%	79.00
MISC	70.20%	66.10%	68.09
ORG	69.16%	47.80%	56.53
PER	49.11%	27.15%	34.97
Overall	69.09%	53.26%	60.15

German devel.	Precision	Recall	$F_{\beta=1}$
LOC	60.15%	41.91%	49.40
MISC	86.96%	9.90%	17.78
ORG	56.19%	29.25%	38.47
PER	55.75%	51.25%	53.40
Overall	58.36%	34.68%	43.50

German test	Precision	Recall	$F_{\beta=1}$
LOC	64.69%	40.00%	49.43
MISC	61.61%	10.30%	17.65
ORG	54.43%	28.59%	37.49
PER	66.45%	58.66%	62.31
Overall	63.49%	38.25%	47.74

Table 4: Performance of best network from Table 3 on English development and test data by type of NE, and performance of the best run of a network configured similarly to Net7 on German development and test data.

References

F. A. Gers and J. Schmidhuber. 2000. Long Short-Term Memory Learns Context-Free and Context-Sensitive Languages. Technical Report IDSIA-03-00, IDSIA, Switzerland.

J.A. Hammerton. 2001. Clause identification with Long Short-Term Memory. In W. Daelemans and R. Zajac, editors, *Proceedings of the workshop on Computational Natural Language Learning (CoNLL 2001), ACL 2001, Toulouse, France*.

J. A. Hammerton. unpublished. Towards scaling up connectionist language learning: Connectionist Shallow Parsing. Unpublished manuscript.

S. Hochreiter and J. Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

D. L. James and R. Miikkulainen, 1995. *SARDNET: A Self-0rganizing Feature Map for Sequences*, pages 577–584. MIT Press, Cambridge, MA.

J. Zavrel and J. Veenstra. 1996. The language environment and syntactic word class acquisition. In Koster C. and Wijnen F., editors, *Proceedings of the Groningen Assembly on Language Acquisition (GALA '95)*.