# Error Entropy Minimization for LSTM Training*

Luís A. Alexandre[1] and J.P. Marques de Sá[2]

[1] Department of Informatics and IT-Networks and Multimedia Group, Covilhã, University of Beira Interior, Portugal,
lfbaa@di.ubi.pt
[2] Faculty of Engineering and INEB, University of Porto, Portugal,
jmsa@fe.up.pt

**Abstract.** In this paper we present a new training algorithm for the Long Short-Term Memory (LSTM) recurrent neural network. This algorithm uses entropy instead of the usual mean squared error as the cost function for the weight update. More precisely we use the Error Entropy Minimization approach, were the entropy of the error is minimized after each symbol is present to the network. Our experiments show that this approach enables the convergence of the LSTM more frequently than with the traditional learning algorithm. This in turn relaxes the burden of parameter tuning since learning is achieved for a wider range of parameter values. The use of EEM also reduces, in some cases, the number of epochs needed for convergence.

## 1 Introduction

One of the most promising machines for sequence learning is the Long Short-Term Memory (LSTM) recurrent neural network [1,2,3,4]. In fact, it has been shown that LSTM outperforms traditional recurrent neural networks (RNNs) such as Elman, Back-Propagation Through Time (BPTT) and Real-Time Recurrent Learning (RTRL) in problems where the need to retain information for long time intervals exists [1].

Traditional RNNs suffer from the problem of loosing error information pertaining to long time lapses. This occurs because the error signals tend to vanish over time [5]. LSTM is able to deal with this problem since it protects error information from decaying using gates.

Usually error backpropagation for neural network learning is made using MSE as the cost function. The authors of [6] introduced an error-entropy minimization algorithm that used Renyi's quadratic entropy. They applied their approach to problems of time-series prediction (MacKey Glass chaotic time series) and non-linear system identification. Note that these problems do not need to retain information for long time lapses. They are usually solved using Time Delay Neural Networks (TDNNs).

In [7], the authors propose the use of the minimization of the error entropy instead of MSE as a cost function for classification purposes. In terms of the

* This work was supported by the Portuguese FCT-Fundação para a Ciência e Tecnologia (project POSC/EIA/56918/2004).

S. Kollias et al. (Eds.): ICANN 2006, Part I, LNCS 4131, pp. 244–253, 2006.
© Springer-Verlag Berlin Heidelberg 2006

entropy measure, two approaches have been tested both with good results when compared to MSE: in [7,8] Renyi's quadratic entropy was used. In [9] the EEM algorithm was used with Shannon's entropy.

In this paper we adapt the LSTM for learning long time lapse problems using EEM and present several experiments that show the benefits that can be obtained from such an approach.

The rest of the paper is organized as follows: the next section presents LSTM, the following section shows how EEM can be incorporated into the learning algorithm of the LSTM. Section IV presents the experiments and the last section contains the conclusions.

## 2 LSTM

The LSTM was proposed in [1]. It results from the use of gates to keep the non-linearities (the transfer functions) in the neurons from making the error information pertaining to long time lapses vanish. Note that the use of gates is only one possibility to avoid this problem that affects traditional RNNs: other approaches can be found in [5], pp. 776.

In the following brief discussion we are referring to the original proposition in [1]: other approaches have been proposed [2,3,4].

The main element of the LSTM is the block: a block is a set of cells and two gates (see fig. 1). The gates are called input and output gates.

Each cell (see fig. 2) is composed of a central linear element called the CEC (Constant Error Carousel), two multiplicative units that are controlled by the block's input and output gates, and two non-linear functions $g()$ and $h()$.

The CEC is responsible for keeping the error unchanged for an arbitrarily long time lapse. The multiplicative units controlled by the gates decide when the error should be updated.

A LSTM network consists of the normal input and output layers. Typically hidden layers may also be used but the distinguishing characteristic is the use between the input and output layers of one or more blocks of cells. Each block may have an arbitrary number of cells.

The input layer is connected to all the gates and to all the cells and to the output layer. The gates and the cells have input connections from all cells and all gates.

For a detailed description, of the learning algorithm see [1].

## 3 EEM for LSTM

### 3.1 The EEM

The idea behind EEM is to replace the MSE, as the cost function of a learning system, with the entropy of the error.

In [6] it is shown that the minimization of the error entropy (in particular, Renyi's entropy) results in the minimization of the divergence between the joint

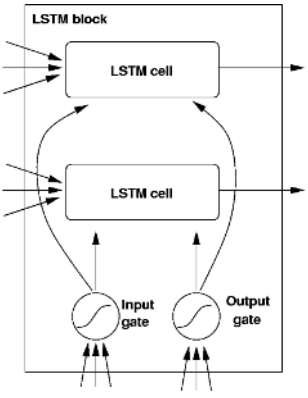


**Fig. 1.** An LSTM block

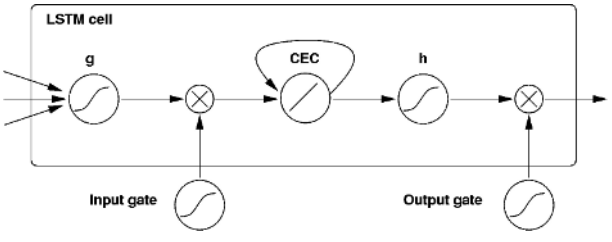


**Fig. 2.** An LSTM cell (also visible are the input and output gates)

pdfs of input-target and input-output signals. This suggests that the distribution of the output of the system is converging to the distribution of the targets.

Also, when the entropy is minimized, for the classification case and under certain mild conditions, implies that the error must be zero (see proof in [8]).

Let the error $e(j) = T(j) - y(j)$ represent the difference between the target $T$ of the $j$ output neuron and its output $y$, at a given time $t$ (not given explicitly to keep the notation simple).

We will replace the MSE of the variable $e(j)$ for its EEM counterpart.

First it is necessary to estimate the pdf of the error. For this we use the Parzen window approach

$$\hat{f}(e(j)) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{e(j) - e(i)}{h}\right) \tag{1}$$

where $h$ represents the bandwidth of the kernel $K$ and $n$ is the number of neurons in the output layer.

The kernel used is the Gaussian kernel given by

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

Renyi's quadratic entropy is given by

$$H_{R2}(x) = -\log\left(\int_C (f(x))^2 dx\right) \tag{2}$$

where $C$ is the support of $x$ and $f(\cdot)$ is its density function.

Note that equation (2) can be seen as the logarithm of the expected value of the pdf: $-\log E[f(x)]$. This justifies the use of the following estimator for $H_{R2}$

$$\hat{H}_{R2}(x) = -\log\left(\frac{1}{n}\sum_{i=1}^{n} f(x_i)\right)$$

Once we put the estimator of the pdf from expression (1) into this last expression, where $x$ is also replaced by the error $e(j)$, we get the final practical expression of our cost function

$$\hat{H}_{R2}(e(j)) = -\log\left(\frac{1}{n^2 h}\sum_{i=1}^{n}\sum_{u=1}^{n} K\left(\frac{e(i)-e(u)}{h}\right)\right) \tag{3}$$

Note that instead of the time complexity for the MSE which is $O(n)$, the EEM approach has $O(n^2)$ complexity.

### 3.2 Application of EEM to LSTM Learning

In this paper we modified the gradient-based learning algorithm of LSTM presented in [1] and replace the use of the MSE as the cost function by the EEM.

The change in the derivation presented in [1] occurs in the following expression (the backpropagation error seen at the output neuron $k$):

$$E(k) = f'(net(k))(T(k) - y(k)) \tag{4}$$

where $f(\cdot)$ is the sigmoid transfer function, $f'(\cdot)$ represents its derivative, $net(k)$ is the activation of the output neuron $k$ at time $t$, $T(k)$ is the target for the output neuron $k$ at time $t$ and $y(k)$) is the output of neuron $k$ at time$t$ (there are differences in the variable names to make this expression coherent with the rest of the paper).

This expression becomes (see the appendix for the derivation)

$$E(k) = Qf'_k(net(k))\sum_{i=1}^{n}\exp\left(-\frac{a(i,k)}{2h^2}\right)a(k,i) \tag{5}$$

where $Q$ stands for a constant term and $a(i,k)$ is

$$a(i,k) = T_i(t) - y_i(t) - T_k(t) + y_k(t) = e(i) - e(k)$$

Of course this change will affect all the backpropagated errors.

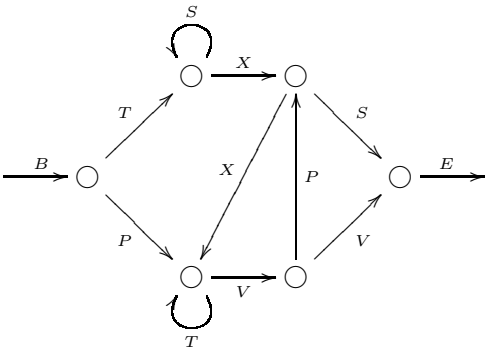


**Fig. 3.** Reber grammar

## 4 Experiments

In this section we present several experiments that compare the performance of LSTM learning with MSE versus EEM.

We use three standard data sets for this purpose: the Reber grammar problem, the embedded Reber grammar and the context-sensitive grammar $A^n B^n$.

### 4.1 Reber Grammar

The finite automata in figure 3 generates strings from the grammar known as the Reber grammar.

The strings are generated by starting at B, and moving from node to node, adding the symbols in the arcs to the string. The symbol E marks the end of a string. When there are two arcs leaving a node, one is chosen randomly with equal probability. This process generates strings of arbitrary length.

We conducted several experiments were the goal was the prediction of the next valid symbol of a string, after the presentation of a given symbol. For instance, if the network receives a starting symbol B it has to predict that the possible next symbols are P or T. If the network is able to correctly predict all possible symbols of all strings both from the training and test sets, using less than 250.000 sequences for learning, we say that it converged.

We used a set with 500 strings from the Reber grammar for training and a different set with 500 strings for testing. For each topology and value of the parameter $h$ we repeated the process 100 times. The change was the random initialization of the network weights. Tables 1 and 2 present the results. They show the percentage of the trained networks that were able to learn perfectly both the training and test sets, and the average and standard deviation of the number of sequences that were used for training.

We tested the two topologies: the strings are codified in a 1-out-of-7 coding, so both input and output layers have 7 neurons. The topologies did not use any traditional neurons in the hidden layer. In the first case two blocks were used,

**Table 1.** Results for the experiments with the Reber grammar for the topology (7:0:2(2,1):7). ANS stands for the Average Number of Sequences necessary for convergence.

| | Learning rate=0.1 | | Learning rate=0.2 | | Learning rate=0.3 | |
|---|---|---|---|---|---|---|
| | ANS (std) [$10^3$] | % conv. | ANS (std) [$10^3$] | % conv. | ANS (std) [$10^3$] | % conv. |
| MSE | 15.1 (24.5) | 38 | 74.9 (116.5) | 63 | 61.0 (111.5) | **56** |
| EEM h=1.3 | 81.8 (115.4) | 36 | 42.6 (51.5) | 11 | 113.6 (135.6) | 7 |
| EEM h=1.4 | 45.6 (68.2) | 45 | 70.6 (93.8) | 11 | 61.8 (63.6) | 10 |
| EEM h=1.5 | 26.0 (43.9) | 54 | 84.1 (120.2) | 29 | 47.2 (39.1) | 13 |
| EEM h=1.6 | 28.4 (43.1) | **66** | 58.0 (84.2) | 37 | 135.1 (160.1) | 15 |
| EEM h=1.7 | 23.0 (25.9) | 64 | 54.9 (87.8) | 40 | 96.9 (135.8) | 30 |
| EEM h=1.8 | 75.8 (51.8) | 30 | 60.1 (96.8) | 50 | 66.0 (111.8) | 33 |
| EEM h=1.9 | 78.0 (110.1) | 62 | 53.6 (94.1) | 61 | 48.7 (66.2) | 33 |
| EEM h=2.0 | 49.3 (77.6) | 58 | 57.6 (109.0) | **67** | 57.4 (83.7) | 51 |

**Table 2.** Results for the experiments with the Reber grammar for the topology (7:0:2(2,2):7). ANS stands for the Average Number of Sequences necessary for convergence.

| | Learning rate=0.1 | | Learning rate=0.2 | | Learning rate=0.3 | |
|---|---|---|---|---|---|---|
| | ANS (std) [$10^3$] | % conv. | ANS (std) [$10^3$] | % conv. | ANS (std) [$10^3$] | % conv. |
| MSE | 19.2 (26.1) | 41 | 46.0 (77.2) | 53 | 30.1 (50.7) | 61 |
| EEM h=1.3 | 57.2 (67.5) | 43 | 44.3 (58.8) | 18 | 28.4 (12.9) | 5 |
| EEM h=1.4 | 20.2 (27.8) | 55 | 70.5 (112.4) | 22 | 108.9 (130.5) | 10 |
| EEM h=1.5 | 33.6 (55.3) | 59 | 68.1 (109.8) | 36 | 31.7 (32.7) | 19 |
| EEM h=1.6 | 26.5 (41.9) | **65** | 38.3 (47.2) | 42 | 58.4 (82.7) | 25 |
| EEM h=1.7 | 32.3 (57.2) | 53 | 48.4 (83.8) | 48 | 55.6 (82.9) | 26 |
| EEM h=1.8 | 24.5 (48.8) | 60 | 54.4 (88.3) | 61 | 43.4 (82.0) | 40 |
| EEM h=1.9 | 51.8 (76.7) | 49 | 70.6 (134.5) | 66 | 62.1 (108.1) | **66** |
| EEM h=2.0 | 44.6 (79.3) | 48 | 48.3 (85.8) | **68** | 44.3 (70.2) | 41 |

one with one cell and the other with two cells (table 1). In the second case, both blocks had two cells (table 2). Each table shows the results for learning rates of 0.1, 0.2 and 0.3.

The MSE line refers to the use of the original learning algorithm.

The results are discussed in section 4.4.

### 4.2 Embedded Reber Grammar

The second set of experiments uses the Embedded Reber Grammar (ERG): it is generated according to figure 4.

This grammar produces two types of strings: BT<reber string>TE and BP<reber string>PE. In order to recognize these strings, the learning machine has to be able to distinguish them from strings such as BP<reber string>TE and BP<reber string>TE. To do this it is essential to remember the second symbol

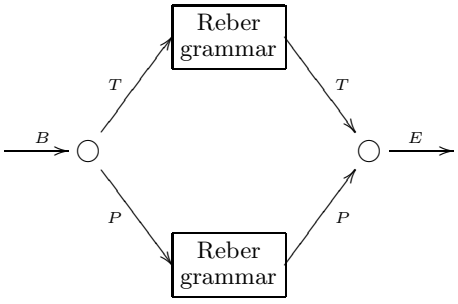


**Fig. 4.** Embedded Reber grammar

**Table 3.** Results for the experiments with the embedded Reber grammar

| | Learning rate=0.1 | | Learning rate=0.3 | |
|---|---|---|---|---|
| | ANS (std) [$10^3$] | % conv. | ANS (std) [$10^3$] | % conv. |
| MSE | 44.4(48.4) | 8 | 66.3 (40.1) | 6 |
| EEM h=1.3 | 79.6 (94.5) | 7 | - | 0 |
| EEM h=1.4 | 13.1 (6.5) | 8 | - | 0 |
| EEM h=1.5 | 49.9 (53.8) | 12 | 327 (-) | 1 |
| EEM h=1.6 | 65.7 (46.9) | 8 | 62.0 (74.5) | 3 |
| EEM h=1.7 | 41.2 (26.9) | 8 | 183.7 (137.4) | 3 |
| EEM h=1.8 | 60.6 (35.8) | 12 | 102.4 (101.4) | 5 |
| EEM h=1.9 | 124.3 62.4 | 13 | 76.9 (125.0) | **7** |
| EEM h=2.0 | 143.2 (111.9) | **14** | 84.6 (87.6) | **7** |

in the sequence such that it can be compared with the second last symbol. Notice that the length of the sequence can be arbitrarily large.

This problem is no longer learnable by an Elman net but a RTRL can learn it with some difficulty, since, as opposed to the RG problem, there is the need to retain information for long time lapses.

In this case the experiments were similar to the ones reported in the previous section but the learning rates used were 0.1 and 0.3. The train and test sets had also 500 strings each. This time only one topology was used: 7 neurons in the input and output layer (the codification is the same as in the RG example), and four blocks each with 3 cells. The experiments were also repeated 100 times. The results are in table 3.

### 4.3 $A^nB^n$ Grammar

This data set consists in a series of strings from the context-sensitive grammar $A^nB^n$. Valid strings consist of $n$ $A$ symbols followed exactly by $n$ $B$ symbols.

The network is trained with only the correct strings for $n$ from 1 up to 10. We consider that the network converged if it is able to correctly classify all the

**Table 4.** Results for the experiments with the grammar $A^nB^n$

| | Test $n = 1, \ldots, 50$ | | Test $n = 1, \ldots, 100$ | |
|---|---|---|---|---|
| | Average n. seq. (std) [$10^3$] | % conv. | Average n. seq. (std) [$10^3$] | % conv. |
| MSE | 4.93 (2.80) | 17 | 4.90 (2.41) | 12 |
| EEM h=1.3 | 10.31 (7.40) | 18 | 8.75 (6.48) | 13 |
| EEM h=1.4 | 11.67 (8.11) | 19 | 11.90 (7.94) | 12 |
| EEM h=1.5 | 15.31 (8.25) | 28 | 16.08 (7.76) | 18 |
| EEM h=1.6 | 17.06 (8.62) | 36 | 17.51 (8.46) | 25 |
| EEM h=1.7 | 18.77 (8.78) | 45 | 18.33 (8.33) | 29 |
| EEM h=1.8 | 20.74 (8.74) | **50** | 19.69 (7.83) | **35** |
| EEM h=1.9 | 20.80 (8.38) | 48 | 21.09 (7.39) | **35** |
| EEM h=2.0 | 22.19 (8.20) | 49 | 22.50 (7.52) | 33 |

strings in both the training and test sets, using less than 50.000 sequences for learning.

In the first experiment we used for the test set the correct strings for $n = 1 \ldots 50$. In the second experiment we used the strings $n = 1 \ldots 100$. In both experiments the topology of the network was three neurons in the input layer, two blocks each with one neuron and three neurons in the output layer. Both experiments were repeated 100 times for a learning rate of 1.0. The results obtained are in table 4.

### 4.4 Discussion

For the Reber grammar experiments, the EEM improved the percentage of convergence in all six sets of experiments except for the first topology with learning rate 0.3. In one case (first topology and learning rate=0.2) it not only outperformed MSE but there was also a reduction in the number of sequences necessary for network converged from $74.9 \times 10^3$ to $57.6 \times 10^3$.

It can also be seen that for MSE then increase in the value of the learning rate was good in terms of percentage of convergence, specially in the case of the second topology, whereas for the EEM the best results were obtained for both topologies with a learning rate of 0.2.

In the case of the experiments with the ERG, and for the learning rate 0.1, two benefits were found from the application of the EEM: in two cases ($h = 1.4$ and 1.7) we were able to obtain the same percentage of convergence but with a smaller number of required training sequences. In four other cases, the percentage of convergence increased from 8 to 12, 13 and 14%. In these cases the number of necessary sequences also increased when compared to the use of MSE. When the learning rate was set at 0.3, there were two situations ($h = 1.3$ and 1.4) where the EEM was not able to converge. Although the best result was still obtained with the EEM for h=1.9 and 2.0. In these experiments it is apparent that the increase of the learning rate was not beneficial either for MSE nor for the EEM.

Finally, the experiments with the $A^nB^n$ grammar confirmed that the use of the EEM is beneficial in terms of increasing the convergence percentage: from the 16

sets of 100 repetitions only one had the same performance of the MSE; all other sets improved. Again, the number of necessary training sequences increased.

## 5 Conclusions

In this paper a new learning algorithm for LSTM was introduced. It uses the EEM as a cost function instead of the MSE. From the experiments made we conclude that this approach is beneficial since there is a sustainable increase in the convergence percentage of the networks. This improvement comes with the cost of a longer training time since the EEM algorithm is slower than the MSE and also the networks tend to need more training epochs to achieve perfect learning.

When using pdf estimation with kernels, the problem of the value to choose for the kernel bandwidth is always present. In this paper we used fixed values during the training stage. We intend to use adaptive approaches to the setting of $h$ so that it can adapt dynamically during training.

## References

1. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8) (1997) 1735–1780
2. Gers, F., Schmidhuber, J., Cummins, F.: Learning to forget: Continual prediction with LSTM. Neural Computation **12**(10) (2000) 2451–2471
3. Gers, F., Schmidhuber, J.: Recurrent nets that time and count. In: Proc. IJCNN'2000, Int. Joint Conf. on Neural Networks, Como, Italy (2000)
4. Pérez-Ortiz, J., Gers, F., Eck, D., Schmidhuber, J.: Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. Neural Networks **16**(2) (2003) 241–250
5. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edition. Prentice Hall (1999)
6. Erdogmus., D., Principe, J.: An error-entropy minimization algorithm for supervised training of nonlinear adaptive systems. IEEE Trans. Signal Processing **50**(7) (2002) 1780–1786
7. Santos, J., Alexandre, L., Sereno, F., Marques de Sá, J.: Optimization of the error entropy minimization algorithm for neural network classification. In: ANNIE 2004, Intelligent Engineering Systems Through Artificial Neural Networks. Volume 14., St.Louis, USA, ASME Press Series (2004) 81–86
8. Santos, J., Alexandre, L., Marques de Sá, J.: The error entropy minimization algorithm for neural network classification. In Lofti, A., ed.: Proceedings of the 5th International Conference on Recent Advances in Soft Computing, Nottingham, United Kingdom (2004) 92–97
9. Silva, L., Marques de Sá, J., Alexandre, L.: Neural network classification using Shannon's entropy. In: 13th European Symposium on Artificial Neural Networks - ESANN 2005, Bruges, Belgium (2005) 217–222

## Appendix

Here we derive expression (5). The term $(T(k) - y(k))$ in equation (4) comes from the derivative of the MSE

$$\frac{1}{n}\sum_{i=1}^{n}(T(i) - y(i))^2$$

w.r.t. the output $y_k$.

What we are going to do is find this same derivative, but now for expression (3). We note that since the logarithm in expression (3) is a monotonically increasing function, to minimize it is the same as to minimize its operand. So, we will find the partial derivative of the operand, which is given by

$$\frac{1}{n^2 h\sqrt{2\pi}}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial}{\partial y_k}\exp\left(-\frac{(e(i)-e(j))^2}{2h^2}\right)$$

$$= \frac{1}{n^2 h\sqrt{2\pi}}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial}{\partial y_k}\exp\left(-\frac{(T(i)-y(i)-T(j)+y(j))^2}{2h^2}\right) \quad (6)$$

Now, when $i = k$ the derivative of the inner term becomes

$$\exp\left(-\frac{(T(k)-y(k)-T(j)+y(j))^2}{2h^2}\right)\left(-\frac{1}{2h^2}\right)2(T(k)-y(k)-T(j)+y(j))(-1) \quad (7)$$

Likewise, if $j = k$, the derivative becomes

$$\exp\left(-\frac{(T(i)-y(i)-T(k)+y(k))^2}{2h^2}\right)\left(-\frac{1}{2h^2}\right)2(T(i)-y(i)-T(k)+y(k)) \quad (8)$$

Expressions (7) and (8) yield the same values since they only differ in sign in the term that is squared and the dummy variables $i$ and $j$ have both the same range (from 1 to $n$). This allows us to write the derivative of the operand of (3) as

$$Q\sum_{i=1}^{n}\exp\left(-\frac{(T(i)-y(i)-T(k)+y(k))^2}{2h^2}\right)(T(i)-y(i)-T(k)+y(k)) \quad (9)$$

where

$$Q = \frac{2}{n^2 h^3\sqrt{2\pi}}$$