Derivations of Backpropagation Through Time Gradients for LSTM Neural Networks

Part of a Honors Thesis Regarding LSTM Neural Networks

By Moritz Nakatenus

December 2017

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Abstract

In this work, the gradients for training a LSTM cell will be derived fully analytically. Machine Learning frameworks like TensorFlow do compute the gradients using symbolic differentiation. However, to get a deeper understanding of LSTM Neural Networks it is useful to calculate the gradients by hand. This work also deals with the Backpropagation Through Time algorithm and the derivation of gradients for a softmax output layer.

Contents

1 Derivation of the Gradients of the Cross-Entropy Loss Function

This derivations will focus to result in a matrix-representation. That makes everything more clearly, because the indexing shrinks and there are less sum signs. Moreover, in many computer languages where neural networks are implemented, matrix representations are used.

Firstly, the derivation of the gradient for the multi-class case will be shown. For multi-class classification the output-layer will be a softmax output-layer

$$h_t \in \mathbb{R}^n, \hat{y}_t, s_t \in \mathbb{R}^m, V \in \mathbb{R}^{mxn}$$

$$\hat{y}_t = \text{softmax}(s_t) = \frac{e^{s_t}}{\sum_i e^{(s_t)_i}} \quad \text{with} \quad s_t = Vh_t, \quad \sum_i (\hat{y}_t)_i = 1 \quad \text{and} \quad (\hat{y}_t)_i \in [0,1] \quad \text{for} \quad i \in \{1,...,m\}.$$

The loss function is a multiclass cross-entropy loss function

$$E_T = -\sum_i (t_T)_i \cdot log((\hat{y}_T)_i) \quad \text{with} \quad \sum_i (t_T)_i = 1 \quad \text{and} \quad (t_t)_i \in [0,1] \quad \text{for} \quad i \in \{1,...,m\}$$

where t_t is the target-vector to learn.
To compute the gradient of the matrix V a vector notation will be used

$$\begin{aligned} &e_i, e_i^\tau \quad \text{is a unit vector or its transposed vector} \\ &e_i \otimes e_j \quad \text{is the tensor product of } e_i \text{ and } e_j \end{aligned}$$

The following gradients are derived according to the denominator-layout notation. That means, gradients of scalar functions with respect to vectors do result in a vector. And gradients of vectors with respect to scalar functions do result in a transposed vector.

For computing the gradient of matrix V the chain rule is applied. This has the advantage, that you can split the final gradient into subgradients, which can be reused.

$$\frac{\partial E_T}{\partial \hat{y_T}} = -\sum_i (t_T)_i \cdot \frac{\partial log((\hat{y}_T)_i)}{\partial \sum_{i'} (\hat{y}_T)_{i'}} \cdot e_i = -\sum_i (t_T)_i \cdot \frac{\partial log((\hat{y}_T)_i)}{\partial (\hat{y}_T)_i} \cdot e_i = -\sum_i \frac{(t_T)_i}{(\hat{y}_T)_i} \cdot e_i = -\text{diag}(\hat{y}_T)^{-1} t_T$$

$$\frac{\partial (\hat{y}_T)_i}{\partial (s_T)_k} = \begin{cases} \frac{e^{(s_T)_i}}{\sum_{i'} e^{(s_T)_{i'}}} - \left(\frac{e^{(s_T)_i}}{\sum_{i'}} e^{(s_T)_{i'}}\right)^2 = (\hat{y}_T)_i - (\hat{y}_T)_i{}^2 = (\hat{y}_T)_i \cdot (1 - (\hat{y}_T)_i) & \text{with } i = k \\ -\frac{e^{(s_T)_i} \cdot e^{(s_T)_k}}{\left(\sum_{i'} e^{(s_T)_{i'}}\right)^2} = -(\hat{y}_T)_i \cdot (\hat{y}_T)_k & \text{with } i \neq k \end{cases} \tag{1.1}$$

$$\begin{aligned} \frac{\partial \hat{y}_T}{\partial s_T} &= \sum_i \sum_k \frac{\partial (\hat{y}_T)_i}{\partial (s_T)_k} \cdot e_k \otimes e_i = \sum_i \frac{\partial (\hat{y}_T)_i}{\partial (s_T)_i} \cdot e_i \otimes e_i + \sum_i \sum_{k \neq i} \frac{\partial (\hat{y}_T)_i}{\partial (s_T)_k} \cdot e_k \otimes e_i \\ &= \sum_i (\hat{y}_T)_i - (\hat{y}_T)_i{}^2 \cdot e_i \otimes e_i - \sum_i \sum_{k \neq i} (\hat{y}_T)_i \cdot (\hat{y}_T)_k \cdot e_k \otimes e_i \\ &= -\sum_i \sum_k (\hat{y}_T)_i \cdot (\hat{y}_T)_k \cdot e_k \otimes e_i + \sum_i (\hat{y}_T)_i \cdot e_i \otimes e_i \\ &= -\hat{y}_T \hat{y}_T^\tau + \text{diag}(\hat{y}_T) \end{aligned}$$

$$
\begin{aligned}
\frac{\partial E_T}{\partial s_T} =& \frac{\partial E_T}{\partial \hat{y}_T}\frac{\partial \hat{y}_T}{\partial s_T} = \frac{\partial \hat{y}_T}{\partial s_T}\frac{\partial E_T}{\partial \hat{y}_T}, \quad \text{order of multiplication swaps because of the denominator-layout notation} \\
=&(-\hat{y}_T\hat{y}_T^\tau + \text{diag}(\hat{y}_T))(-\text{diag}(\hat{y}_T)^{-1}t_T) \\
=&\hat{y}_T\hat{y}_T^\tau\text{diag}(\hat{y}_T)^{-1}t_T - \text{diag}(\hat{y}_T)\text{diag}(\hat{y}_T)^{-1}t_T \\
=&\hat{y}_T\hat{y}_T^\tau\text{diag}(\hat{y}_T)^{-1}t_T - \mathbb{1}t_T \\
=&\hat{y}_T\hat{y}_T^\tau\text{diag}(\hat{y}_T)^{-1}t_T - t_T \\
=&\hat{y}_T\sum_i (t_T)_i - t_T \\
=&\hat{y}_T - t_T
\end{aligned}
$$

$$
x, q \in \{1, ..., m\}, \quad y, p \in \{1, ..., n\}
$$

$$
\begin{aligned}
\frac{\partial (s_T)_q}{\partial V} =& \sum_x\sum_y \frac{\partial (s_T)_q}{\partial V_{xy}} \cdot e_x \otimes e_y \\
=& \sum_x\sum_y\sum_p \frac{\partial V_{qp}}{\partial V_{xy}} \cdot h_p \cdot e_x \otimes e_y \\
=& \sum_x\sum_y\sum_p \delta_{qx}\cdot\delta_{py}\cdot h_p \cdot e_x \otimes e_y \\
=& \sum_x \delta_{qx}\sum_y\sum_p \delta_{py}\cdot h_p \cdot e_x \otimes e_y \\
=& \sum_y\sum_p \delta_{py}\cdot h_p \cdot e_q \otimes e_y \\
=& \sum_p h_p \cdot e_q \otimes e_p \\
\frac{\partial s_T}{\partial V} =& \sum_q \frac{\partial (s_T)_q}{\partial V} = \sum_q\sum_p h_p \cdot e_q \otimes e_p \otimes e_q
\end{aligned}
\tag{1.2}
$$

$$
\begin{aligned}
\frac{\partial E_T}{\partial V} =& \frac{\partial E_T}{\partial s_T}\frac{\partial s_T}{\partial V} = \frac{\partial s_T}{\partial V}\frac{\partial E_T}{\partial s_T} \\
=& \left(\sum_q\sum_p h_p \cdot e_q \otimes e_p \otimes e_q\right)(\hat{y}_T - t_T) \\
=& \sum_q\sum_p (h_p \cdot e_q) \otimes (e_p \cdot (\hat{y}_T - t_T)_q) \\
=& \sum_q\sum_p ((\hat{y}_T - t_T)_q \cdot e_q) \otimes (h_p \cdot e_p) \\
=& (\hat{y}_T - t_T) \otimes h \\
=& (\hat{y}_T - t_T)h^\tau
\end{aligned}
$$

The softmax output-layer is for the multiclass case. For binary classification (output probability for two classes) we can use the binary cross-entropy loss function

$$
E_t = -\sum_i ((t_t)_i \cdot log((\hat{y}_t)_i) + (1 - (t_t)_i) \cdot log(1 - (\hat{y}_t)_i)).
$$

$\hat{y}_t$ is now a conventional sigmoid output-layer, thus each component of $\hat{y}_t$ is a single binary classification

$$
\hat{y}_t = \sigma(s_t) = \frac{1}{1 + e^{-s_t}}.
$$

It turns out that the gradient of the binary cross-entropy loss function is the same as the gradient of the multi-class cross-entropy loss function. This is reasonable, because the binary cross-entropy loss function is a special case of the multi-class loss function. However, the derivation will not be shown, because it is quite similar to the above one.

2 Forwardprogation Rules for a LSTM Neural Network

Long Short-Term Memory Neural Networks were firstly introduced by Hochreiter and Schmidhuber in 1997 [2]. There are many variants of LSTM Neural Networks. The representation used in this work will deal with peephole connections. Peephole connections do connect the gates with the cell states [3]. Hence, the gates will consider information from the long short-term memory.

The following equations describing a LSTM forward propagation model [3].

Hints on notation:

- $\odot$ is the so called hadamard-product, which stands for the element-wise multiplication.
- $[a,b,c] = a+b+c \quad \text{or} \quad [a,b,c] = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$
  it depends on the application. If there is a need for just one matrix describing the propagation of all input values, the sum would be preferable. If we e.g. want to catch more information out of the data, the vectorized interpretation of the input data can be used. Thus, each matrix can be interpreted as a matrix, which is a concatenation of sub-matrices for each specific input (e.g. x_t, h_{t-1} and C_{t-1}).

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

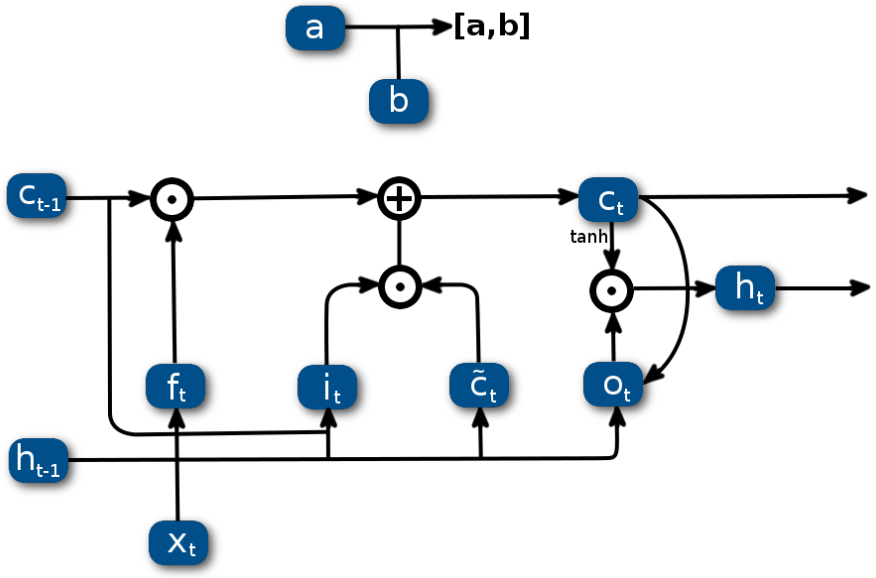

Figure 2.1: More intuitive illustration of the LSTM forward propagation model [3].

3 Backpropagation Through Time Equations for LSTM Neural Networks

3.1 Gradient of Error with Respect to the Output

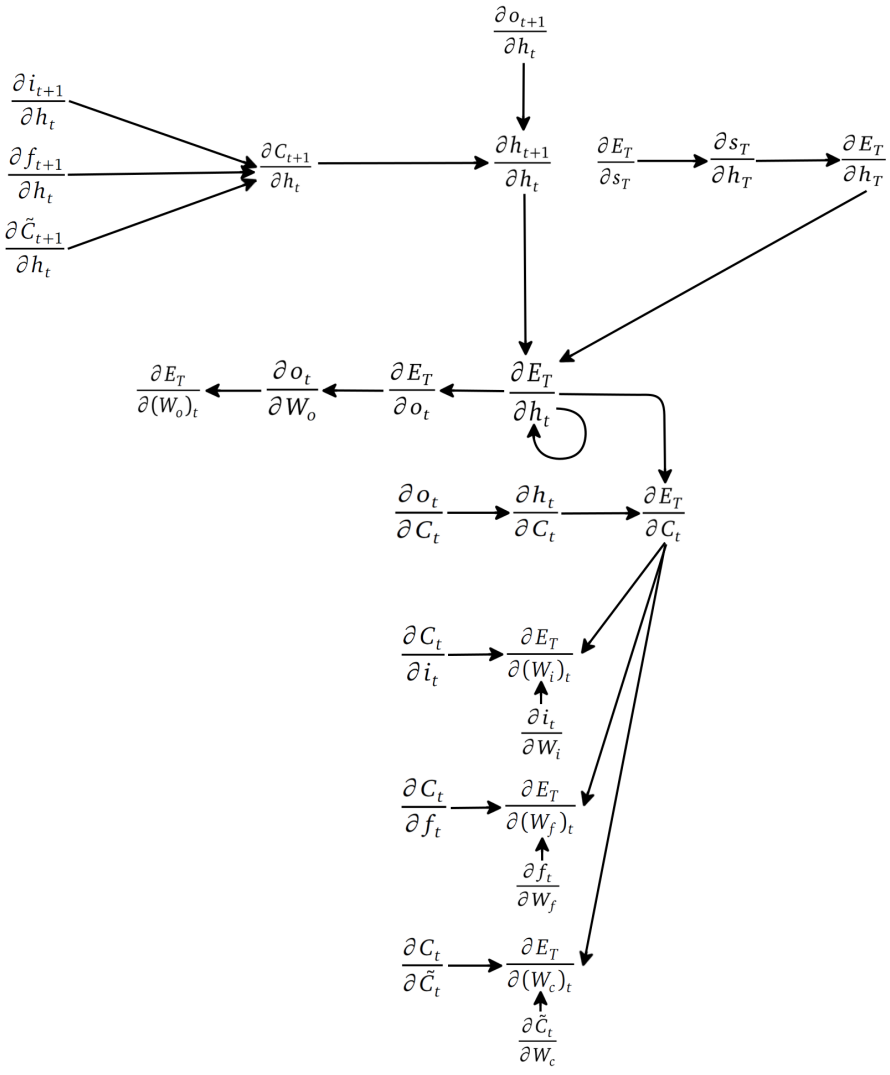

Figure 3.1: As the derivations of the several gradients are complex, here is an interdependency overview.

To calculate the gradients for a LSTM neural network, the gradient $\frac{\partial E_T}{\partial s_T}$ from section 1 is used

$$\frac{\partial E_T}{\partial s_T} = \hat{y}_T - t_T.$$

Hence, in this case a classification output is used. For any other output and error function, $\frac{\partial E_T}{\partial s_T}$ has to be computed accordingly.

For the derivation of the LSTM gradients, the chain rule is applied again. To calculate the gradients of all weight-matrices, we need the gradient $\frac{\partial E_T}{\partial h_t}$ at the top of each gradient chain.

$$p \in \{1, ..., u\}, \quad q \in \{1, ..., v\}, \quad s_t \in \mathbb{R}^m, h_t \in \mathbb{R}^n, C_t \in \mathbb{R}^n, x_t \in \mathbb{R}^n$$

as you can see, for simplyfication C_t, h_t and x_t live in the same dimension. However, for different dimensions the derivation of the gradients is basically the same.

$$\text{diag(x)} = \sum_p x_p \cdot e_p \otimes e_p$$

$$\frac{\partial (a(y) \odot b)}{\partial y} = \sum_p \sum_q \frac{\partial a(y)_p}{\partial y_q} \cdot b_p \cdot e_q \otimes e_p = \sum_p b_p \sum_q \frac{\partial a(y)_p}{\partial y_q} \cdot e_q \otimes e_p = \frac{\partial a(y)}{\partial y} \text{diag}(b)$$

$$i \in \{1, ..., m\}, j \in \{1, ..., n\}, k \in \{1, ..., n\}$$

$$\frac{\partial (h_T)_j}{\partial h_T} = e_j$$

$$\begin{aligned} \frac{\partial (s_T)_i}{\partial h_T} =& \frac{\partial (V h_T)_i}{\partial h_T} = \sum_j V_{ij} \cdot \frac{\partial (h_T)_j}{\partial h_T} = \sum_j V_{ij} \cdot e_j \\ \frac{\partial s_T}{\partial h_T} =& \sum_i \frac{\partial (s_T)_i}{\partial h_T} \cdot e_i^\tau = \sum_i \sum_j V_{ij} \cdot e_j e_i^\tau = \sum_i \sum_j V_{ij} \cdot e_j \otimes e_i = V^\tau \end{aligned} \tag{3.1}$$

$$\begin{aligned} \frac{\partial E_T}{\partial h_T} =& \frac{\partial E_T}{\partial s_T} \frac{\partial s_T}{\partial h_T} = \frac{\partial s_T}{\partial h_T} \frac{\partial E_T}{\partial s_T} \\ =& V^\tau (\hat{y}_T - t_T) \end{aligned}$$

$$\frac{\partial}{\partial arg} = \frac{\partial}{\partial arg_j} e_j$$

$$(\sigma')_j = \frac{\partial \sigma(arg)_j}{\partial arg_j} = \sigma(arg)_j (1 - \sigma(arg))_j$$

$$\begin{aligned} \sigma' =& \frac{\partial \sigma(arg)}{\partial arg} = \sum_j \sum_k \frac{\partial \sigma(arg)_j}{\partial arg_k} \cdot e_k \otimes e_j = \sum_j \sum_k \frac{\partial \sigma(arg)_j}{\partial arg_k} \cdot \delta_{jk} \cdot e_k \otimes e_j = \sum_j \frac{\partial \sigma(arg)_j}{\partial arg_j} e_j \otimes e_j \\ =& \text{diag}\left(\frac{\partial}{\partial arg} \odot \sigma \right) \end{aligned} \tag{3.2}$$

x_{t+1} and C_{t+1} do not depend on h_t, hence they are seen as constants and thus vanish computing the gradients.

$$o_{t+1} = \sigma(W_o [x_{t+1}, h_t, C_{t+1}] + b_o)$$

$$\begin{aligned} \frac{\partial o_{t+1}}{\partial h_t} =& \frac{\partial \sigma(W_o [x_{t+1}, h_t, C_{t+1}] + b_o)}{\partial h_t} = \frac{\partial \sigma(W_o [x_{t+1}, h_t, C_{t+1}] + b_o)}{\partial (W_o [x_{t+1}, h_t, C_{t+1}] + b_o)} \frac{\partial (W_o [x_{t+1}, h_t, C_{t+1}] + b_o)}{\partial h_t} \\ =& \frac{\partial (W_o [x_{t+1}, h_t, C_{t+1}] + b_o)}{\partial h_t} \frac{\partial \sigma(W_o [x_{t+1}, h_t, C_{t+1}] + b_o)}{\partial (W_o [x_{t+1}, h_t, C_{t+1}] + b_o)} = W_o^\tau \sigma' \end{aligned} \tag{3.3}$$

for $\frac{\partial W_o h_t}{\partial h_t} = W_o^\tau$ see derivation (3.1). For this result assuming $[x_{t+1}, h_t, C_{t+1}] = x_{t+1} + h_t + C_{t+1}$. If the representation is $[x_{t+1}, h_t, C_{t+1}] = \begin{bmatrix} x_{t+1} \\ h_t \\ C_{t+1} \end{bmatrix} = a$, the gradient changes slightly

$$\sum_u a_u \cdot e_u = h_t, \quad u \in U$$

$$\frac{\partial a_j}{\partial h_t} = \begin{cases} e_j, & \text{if } a_j \in h_t \\ 0 \end{cases}$$

$$\frac{\partial (W_o h_t)_i}{\partial h_t} = \sum_j (W_o)_{ij} \cdot \frac{\partial a_j}{\partial h_t}$$

$$\frac{\partial W_o h_t}{\partial h_t} = \sum_i \sum_j (W_o)_{ij} \cdot \frac{\partial a_j}{\partial h_t} \otimes e_i = \sum_i \sum_j \delta_{ju} (W_o)_{ij} \cdot \frac{\partial a_j}{\partial h_t} \otimes e_i = \sum_i \sum_u (W_o)_{iu} \cdot \frac{\partial a_u}{\partial h_t} \otimes e_i = \sum_i \sum_u (W_o)_{iu} \cdot e_u \otimes e_i$$
$$= ((W_o)_{.U})^\tau, \quad \text{the } {}_{.U} \text{ means use all rows and only columns in the quantity U}$$

The following gradients will use the sum-representation.

$$(\tanh')_j = \frac{\partial \tanh(\arg)_j}{\partial arg_j} = 1 - \tanh(\arg)_j^2$$

$$\tanh' = \frac{\partial \tanh(\arg)}{\partial arg} = \operatorname{diag}\left(\frac{\partial}{\partial arg} \odot \tanh\right), \qquad \text{derivation analogue to (3.2)}$$

$$\frac{\partial \tanh(C_{t+1})}{\partial h_t} = \frac{\partial C_{t+1}}{\partial h_t} \tanh'$$

$$\frac{\partial f_{t+1}}{\partial h_t} = W_f^\tau f'_{t+1} = W_f^\tau \sigma', \quad \frac{\partial i_{t+1}}{\partial h_t} = W_i^\tau i'_{t+1} = W_i^\tau \sigma', \qquad \text{derivation analogue to (3.3)}$$

$$\frac{\partial \tilde{C}_{t+1}}{\partial h_t} = W_c^\tau \tilde{C}'_{t+1} = W_c^\tau \tanh', \qquad \text{derivation analogue to (3.2), but substitute } \sigma \text{ with tanh}$$

$$C_{t+1} = f_{t+1} \odot C_t + i_{t+1} \odot \tilde{C}_{t+1}$$

f_{t+1}, i_{t+1} and $\tilde{C}_{t+1}$ do depend on h_t. Using product rule and linearity for the following gradient

$$\frac{\partial C_{t+1}}{\partial h_t} = \frac{\partial f_{t+1}}{\partial h_t} \operatorname{diag}(C_t) + \frac{\partial i_{t+1}}{\partial h_t} \operatorname{diag}(\tilde{C}_{t+1}) + \frac{\partial \tilde{C}_{t+1}}{\partial h_t} \operatorname{diag}(i_{t+1}).$$

Now there are all building blocks to finally compute the gradient $\frac{E_T}{h_t}$

$$h_{t+1} = o_{t+1} \odot \tanh(C_{t+1})$$

$$\begin{aligned} \frac{\partial h_{t+1}}{\partial h_t} &= \frac{\partial o_{t+1}}{\partial h_t} \operatorname{diag}(\tanh(C_{t+1})) + \frac{\partial \tanh(C_{t+1})}{\partial h_t} \operatorname{diag}(o_{t+1}) \\ &= \frac{\partial o_{t+1}}{\partial h_t} \operatorname{diag}(\tanh(C_{t+1})) + \frac{\partial C_{t+1}}{\partial h_t} \frac{\partial \tanh(C_{t+1})}{\partial C_{t+1}} \operatorname{diag}(o_{t+1}) \\ &= \frac{\partial o_{t+1}}{\partial h_t} \operatorname{diag}(\tanh(C_{t+1})) + \frac{\partial C_{t+1}}{\partial h_t} \tanh' \operatorname{diag}(o_{t+1}) \end{aligned}$$

$$\frac{\partial E_T}{\partial h_t} = \frac{\partial E_T}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} = \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial E_T}{\partial h_{t+1}} = \delta_t^h$$

where $\frac{\partial E_T}{\partial h_t}$ starts with $\frac{\partial E_T}{\partial h_T}$.

3.2 Final Gradients of all Weight Matrices

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

$$
\frac{\partial E_T}{\partial o_t} = \frac{\partial E_T}{\partial h_t}\frac{\partial h_T}{\partial o_t} = \frac{\partial h_T}{\partial o_t}\frac{\partial E_T}{\partial h_t} = \frac{\partial o_t}{\partial o_t}\mathrm{diag}(\tanh(C_t))\delta_t^h = \mathbb{1}\mathrm{diag}(\tanh(C_t))\delta_t^h = \mathrm{diag}(\tanh(C_t))\delta_t^h = \tanh(C_t) \odot \delta_t^h = \delta_t^o
$$

$$
\begin{aligned}
\frac{\partial o_t}{\partial C_t} &= W_o^\tau o_t' = W_o^\tau \sigma' \\
\frac{\partial h_t}{\partial C_t} &= \frac{\partial o_t}{\partial C_t}\mathrm{diag}(\tanh(C_t)) + \frac{\partial \tanh(C_t)}{\partial C_t}\mathrm{diag}(o_t) \\
&= \frac{\partial o_t}{\partial C_t}\mathrm{diag}(\tanh(C_t)) + \tanh(C_t)'\mathrm{diag}(o_t) \\
\frac{\partial C_t}{\partial i_t} &= \frac{\partial}{\partial i_t}(f_t \odot C_{t-1} + i_t \odot \tilde{C}_t) = \frac{\partial i_t}{\partial i_t}\mathrm{diag}(\tilde{C}_t) = \mathbb{1}\mathrm{diag}(\tilde{C}_t) = \mathrm{diag}(\tilde{C}_t) \\
\frac{\partial C_t}{\partial f_t} &= \frac{\partial}{\partial f_t}(f_t \odot C_{t-1} + i_t \odot \tilde{C}_t) = \frac{\partial f_t}{\partial f_t}\mathrm{diag}(C_{t-1}) = \mathbb{1}\mathrm{diag}(C_{t-1}) = \mathrm{diag}(C_{t-1}) \\
\frac{\partial C_t}{\partial \tilde{C}_t} &= \frac{\partial}{\partial \tilde{C}_t}(f_t \odot C_{t-1} + i_t \odot \tilde{C}_t) = \frac{\partial \tilde{C}_t}{\partial \tilde{C}_t}\mathrm{diag}(i_t) = \mathbb{1}\mathrm{diag}(i_t) = \mathrm{diag}(i_t)
\end{aligned}
$$

$$
\frac{\partial E_T}{\partial C_t} = \frac{\partial E_T}{\partial h_t}\frac{\partial h_t}{\partial C_t} = \frac{\partial h_t}{\partial C_t}\frac{\partial E_T}{\partial h_t} = \left(\frac{\partial o_t}{\partial C_t}\mathrm{diag}(\tanh(C_t)) + \tanh(C_t)'\mathrm{diag}(o_t)\right)\delta_t^h
$$

$$
p, q \in \{1, \ldots, n\}
$$

$$
\frac{\partial}{\partial W_i}(W_i[x_t, h_{t-1}, C_{t-1}] + b_i) = \sum_q \sum_p [x_t, h_{t-1}, C_{t-1}]_p \cdot e_q \otimes e_p \otimes e_q, \quad \text{see derivation (1.2) in section 1}
$$

$$
\begin{aligned}
\frac{\partial i_t}{\partial W_i} &= \frac{\partial}{\partial W_i}(W_i[x_t, h_{t-1}, C_{t-1}] + b_i)\frac{\partial i_t}{\partial (W_i[x_t, h_{t-1}, C_{t-1}] + b_i)} \\
&= \left(\sum_q \sum_p [x_t, h_{t-1}, C_{t-1}]_p \cdot e_q \otimes e_p \otimes e_q\right) i_t' \\
&= \left(\sum_q \sum_p [x_t, h_{t-1}, C_{t-1}]_p \cdot e_q \otimes e_p \otimes e_q\right)((\sigma')_q e_q \otimes e_q) \\
&= \sum_q \sum_p ((\sigma')_q e_q) \otimes ([x_t, h_{t-1}, C_{t-1}]_p e_p) \otimes e_q \\
\frac{\partial f_t}{\partial W_f} &= \sum_q \sum_p ((\sigma')_q e_q) \otimes ([x_t, h_{t-1}, C_{t-1}]_p e_p) \otimes e_q, \quad \text{derivation analogue to (3.5)} \\
\frac{\partial \tilde{C}_t}{\partial W_c} &= \sum_q \sum_p ((\tanh')_q e_q) \otimes ([x_t, h_{t-1}]_p e_p) \otimes e_q, \quad \text{derivation analogue to (3.5)} \\
\frac{\partial o_t}{\partial W_o} &= \sum_q \sum_p ((\sigma')_q e_q) \otimes ([x_t, h_{t-1}, C_t]_p e_p) \otimes e_q, \quad \text{derivation analogue to (3.5)}
\end{aligned}
\tag{3.4}
$$

With the above gradients, the calculation of the final gradients can be done

$$\begin{aligned}\frac{\partial E_T}{\partial C_t}\frac{\partial C_t}{\partial i_t} &= \frac{\partial C_t}{\partial i_t}\frac{\partial E_T}{\partial C_t} = \delta_t^i\\ \frac{\partial E_T}{\partial C_t}\frac{\partial C_t}{\partial f_t} &= \frac{\partial C_t}{\partial f_t}\frac{\partial E_T}{\partial C_t} = \delta_t^f\\ \frac{\partial E_T}{\partial C_t}\frac{\partial C_t}{\partial \tilde{C}_t} &= \frac{\partial C_t}{\partial \tilde{C}_t}\frac{\partial E_T}{\partial C_t} = \delta_t^c\\ \frac{\partial E_T}{\partial o_t} &= \delta_t^o\end{aligned}$$

$$\begin{aligned}\frac{\partial E_T}{\partial (W_i)_t} &= \frac{\partial E_T}{\partial C_t}\frac{\partial C_t}{\partial i_t}\frac{\partial i_t}{\partial W_i} = \frac{\partial i_t}{\partial W_i}\frac{\partial C_t}{\partial i_t}\frac{\partial E_T}{\partial C_t} = \frac{\partial i_t}{\partial W_i}\delta_t^i\\ &= \left(\sum_q\sum_p \left((\sigma')_q \cdot e_q\right) \otimes \left([x_t, h_{t-1}, C_{t-1}]_p e_p\right) \otimes e_q\right)\delta_t^i\\ &= \sum_q\sum_p \left((\sigma')_q \cdot e_q\right) \otimes \left([x_t, h_{t-1}, C_{t-1}]_p \cdot \left(\delta_t^i\right)_q \cdot e_p\right)\\ &= \sum_q\sum_p \left((\sigma')_q \cdot \left(\delta_t^i\right)_q \cdot e_q\right) \otimes \left([x_t, h_{t-1}, C_{t-1}]_p \cdot e_p\right)\\ &= \left((\sigma' \mathbb{1}^n) \odot \delta_t^i\right)([x_t, h_{t-1}, C_{t-1}])^\tau\end{aligned} \tag{3.5}$$

where $\mathbb{1}^n$ is a column vector of dimension n containing ones.

$$\begin{aligned}\frac{\partial E_T}{\partial (W_f)_t} &= \frac{\partial E_T}{\partial C_t}\frac{\partial C_t}{\partial f_t}\frac{\partial f_t}{\partial W_f} = \frac{\partial f_t}{\partial W_f}\frac{\partial C_t}{\partial f_t}\frac{\partial E_T}{\partial C_t} = \frac{\partial f_t}{\partial W_f}\delta_t^f\\ &= \left((\sigma' \mathbb{1}^n) \odot \delta_t^f\right)([x_t, h_{t-1}, C_{t-1}])^\tau, \quad \text{derivation analogue to (3.5)}\\ \frac{\partial E_T}{\partial (W_c)_t} &= \frac{\partial E_T}{\partial C_t}\frac{\partial C_t}{\partial \tilde{C}_t}\frac{\partial \tilde{C}_t}{\partial W_c} = \frac{\partial \tilde{C}_t}{\partial W_c}\frac{\partial C_t}{\partial \tilde{C}_t}\frac{\partial E_T}{\partial C_t} = \frac{\partial \tilde{C}_t}{\partial W_c}\delta_t^c\\ &= \left((\tanh' \mathbb{1}^n) \odot \delta_t^c\right)([x_t, h_{t-1}])^\tau, \quad \text{derivation analogue to (3.5)}\\ \frac{\partial E_T}{\partial (W_o)_t} &= \frac{\partial E_t}{\partial o_t}\frac{\partial o_t}{\partial W_o} = \frac{\partial o_t}{\partial W_o}\frac{\partial E_t}{\partial o_t} = \frac{\partial o_t}{\partial W_o}\delta_t^o\\ &= \left((\sigma' \mathbb{1}^n) \odot \delta_t^o\right)([x_t, h_{t-1}, C_t])^\tau, \quad \text{derivation analogue to (3.5)}\end{aligned}$$

$$\frac{\partial i_t}{\partial b_i} = \frac{\partial}{\partial b_i}(W_i[x_t, h_{t-1}, C_{t-1}] + b_i)\frac{\partial i_t}{\partial (W_i[x_t, h_{t-1}, C_{t-1}] + b_i)} = \frac{\partial b_i}{\partial b_i}\sigma' = \mathbb{1}\sigma' = \sigma'$$

$$\begin{aligned}\frac{\partial f_t}{\partial (b_f)_t} &= \sigma', \quad \text{derivation analogue to (??)}\\ \frac{\partial C_t}{\partial (b_c)_t} &= \tanh', \quad \text{derivation analogue to (??)}\\ \frac{\partial o_t}{\partial (b_o)_t} &= \sigma', \quad \text{derivation analogue to (??)}\end{aligned}$$

$$\begin{aligned}\frac{\partial E_T}{\partial (b_i)_t} &= \frac{\partial E_T}{\partial C_t}\frac{\partial C_t}{\partial i_t}\frac{\partial i_t}{\partial b_i} = \frac{\partial i_t}{\partial b_i}\frac{\partial C_t}{\partial i_t}\frac{\partial E_T}{\partial C_t} = \frac{\partial i_t}{\partial b_i}\delta_t^i = \sigma'\delta_t^i\\ \frac{\partial E_T}{\partial (b_f)_t} &= \frac{\partial E_T}{\partial C_t}\frac{\partial C_t}{\partial f_t}\frac{\partial f_t}{\partial b_f} = \frac{\partial f_t}{\partial b_f}\frac{\partial C_t}{\partial f_t}\frac{\partial E_T}{\partial C_t} = \sigma'\delta_t^f\\ \frac{\partial E_T}{\partial (b_c)_t} &= \frac{\partial E_T}{\partial C_t}\frac{\partial C_t}{\partial \tilde{C}_t}\frac{\partial \tilde{C}_t}{\partial b_c} = \frac{\partial \tilde{C}_t}{\partial b_c}\frac{\partial C_t}{\partial \tilde{C}_t}\frac{\partial E_T}{\partial C_t} = \tanh'\delta_t^c\\ \frac{\partial E_T}{\partial (b_o)_t} &= \frac{\partial E_t}{\partial o_t}\frac{\partial o_t}{\partial b_o} = \frac{\partial o_t}{\partial b_o}\frac{\partial E_t}{\partial o_t} = \sigma'\delta_t^o\end{aligned}$$

To get the final gradients, the errors have to be accumulated over time. Now τ is the last timestep

$$\frac{\partial E}{\partial W_i} = \sum_{T=0}^{\tau}\sum_{\tilde{t}=0}^{T}\frac{\partial E_T}{\partial (W_i)_{\tilde{t}}}, \quad \frac{\partial E}{\partial b_i} = \sum_{T=0}^{\tau}\sum_{\tilde{t}=0}^{T}\frac{\partial E_T}{\partial (b_i)_{\tilde{t}}}$$
$$\frac{\partial E}{\partial W_f} = \sum_{T=0}^{\tau}\sum_{\tilde{t}=0}^{T}\frac{\partial E_T}{\partial (W_f)_{\tilde{t}}}, \quad \frac{\partial E}{\partial b_f} = \sum_{T=0}^{\tau}\sum_{\tilde{t}=0}^{T}\frac{\partial E_T}{\partial (b_f)_{\tilde{t}}}$$
$$\frac{\partial E}{\partial W_c} = \sum_{T=0}^{\tau}\sum_{\tilde{t}=0}^{T}\frac{\partial E_T}{\partial (W_c)_{\tilde{t}}}, \quad \frac{\partial E}{\partial b_c} = \sum_{T=0}^{\tau}\sum_{\tilde{t}=0}^{T}\frac{\partial E_T}{\partial (b_c)_{\tilde{t}}}$$
$$\frac{\partial E}{\partial W_o} = \sum_{T=0}^{\tau}\sum_{\tilde{t}=0}^{T}\frac{\partial E_T}{\partial (W_o)_{\tilde{t}}}, \quad \frac{\partial E}{\partial b_o} = \sum_{T=0}^{\tau}\sum_{\tilde{t}=0}^{T}\frac{\partial E_T}{\partial (b_o)_{\tilde{t}}}$$

Now the network can be updated depending on an optimization technique, e.g. the RMSprop optimization method [1] can be used. This method is often used to train Recurrent Neural Networks

$$g_t = \frac{\partial E}{\partial W_i}$$
$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$
$$W_i = W_i - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

The updates can be stochastic, mini-batches or batches. This training process is called the Backpropagation Trough Time algorithm.

It can be computationally expensive and even currupt the gradients, if we sum up all gradients at each timestep up to timestep 0. However, it is possible to just cut some part of the gradients and even get good results. This is called Truncated Backpropagation. The idea is to perform BPTT each k1 timesteps and compute the gradient for k2 timesteps [4].

Bibliography

[1] Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, 2012.

[2] Sepp Hochreiter and Juergen Schmidhuber. Long short-term memory, 1997.

[3] Christopher Olah. Understanding lstm networks, 2015.

[4] Ilya Sutskever. Training recurrent neural networks, 2013.