LSTM Neural Network Implementation

Andreas Theophilou

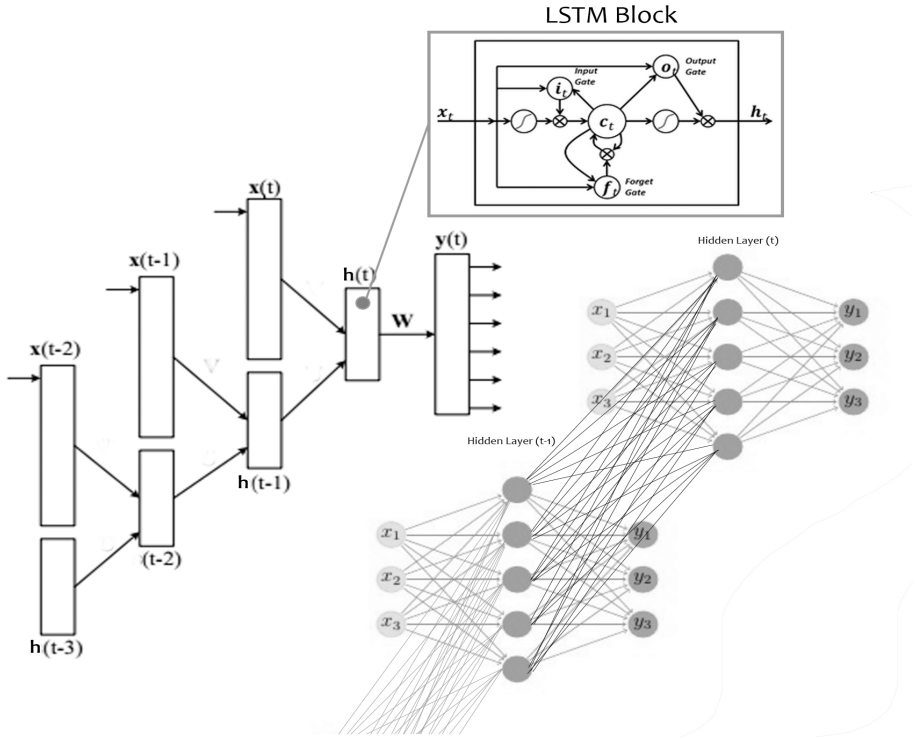
September 9, 2016

1 Introduction

This document serves as an overview of the architecture of LSTM networks in addition to providing the details behind the particular implementation within RNN_NOVA and assumes familiarity with the standard artificial neural network.
LSTMs (long short term memory) are extensions to the RNN (recurrent neural network) proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber.

The LSTM helps prevent the vanishing gradient problem in RNNs through gated inputs to a cell where (numerical) closure of the gates allows longer term dependencies of the overall network.
The benefits of the LSTM and an offspring GRU (gated recurrent unit) have thrived our ability to analyze and predict time dependent processes such as natural language.

The figure below shows the unrolling of a standard neural network with inputs (x) at time (t), a layer of lstmblocks at each (h) at time (t) and two variations, the left where the output is only performed after a certain quantity of time-steps and the right where the output and thereby error is performed at each time (t). Unrolling provides the ability to have many variations of network structures, for instance, the (x) inputs are not required at each time-step and a single layer of inputs when t=1 could be applied to a network with 30 time-steps, meaning the the network will predict the next 29 steps until t=30 based upon a single input vector when t=1.



Due to the unrolling process, back propagating the network must be done with respect to the timestep counter. The weight connections between the layers (x) (h) and (y) are shared between time-steps, when performing weight updates a combination of the weight updates at the (t) are made, due to the potential of exploding gradients when summing over (t) gradient clipping is performed.

2 Feed Forward

Input and Context Layers to LSTMBlock (hidden) outputs

Input Gate

The input gate transfers the input to the cell on the condition is it not numerically closed based on the the current inputs and the cell at t-1 (through the cell bridge).

$$iGate_j^t = \sigma(wxi_j^{(x)} xl_t + whi_j^{(h)} hl_{t-1} + wci_j^{(c)} cell_{t-1} + b_j^{(i)})$$

iGate Weight Connections

$$wxi_j^{(x)} xl_t = \begin{pmatrix} wxi_{11} & \cdots & wxi_{1j} \\ \vdots & \ddots & \vdots \\ wxi_{x1} & \cdots & wxi_{xj} \end{pmatrix} \begin{pmatrix} xl_1 \\ xl_2 \\ .. \\ .. \\ xl_j \end{pmatrix} : whi_j^{(h)} hl_{t-1} = \begin{pmatrix} whi_{11} & \cdots & whi_{1j} \\ \vdots & \ddots & \vdots \\ whi_{h1} & \cdots & whi_{hj} \end{pmatrix} \begin{pmatrix} hl_1^{(t-1)} \\ hl_2^{(t-1)} \\ .. \\ .. \\ hl_j^{(t-1)} \end{pmatrix}$$

$$wci_j^{(c)} cell_{t-1} = \begin{pmatrix} wci_{11} & \cdots & wci_{1i} \\ \vdots & \ddots & \vdots \\ wci_{j1} & \cdots & wci_{ji} \end{pmatrix} \begin{pmatrix} cell_1^{(t-1)} \\ cell_2^{(t-1)} \\ .. \\ .. \\ cell_j^{(t-1)} \end{pmatrix}$$

Cell Gate

$$g_j^{(t)} = \phi(wxc_j^{(x)} xl_t + whc_j^{(h)} hl_j^{(t-1)} + b_j^{(c)})$$
$$cell_j^t = fGate_j^{(t)} * cell_j^{(t-1)} + iGate_j^{(t)} g_j^{(t)}$$

Forget Gate

The forgot gate decides if the cell acknowledges it's previous values as the forget gate is multiplied by cell at t-1 when forwarding through the cell.

$$fGate_j^t = \sigma(wxf_j^{(x)} xl_t + whf_j^{(h)} hl_{t-1} + wcf_j^{(c)} cell_{t-1} + b_j^{(f)})$$

Output Gate

The output gate decides if the lstm-block is allowed to perform an output.

$$oGate_j^t = \sigma(wxo_j^{(x)} xl_t + who_j^{(h)} hl_{t-1} + wco_j^{(c)} cell_{t-1} + b_j^{(i)})$$

LSTM-Block Output

$$z_j^{(t)} = \phi(cell_j^{(t)})$$
$$hl_j^t = oGate_j^{(t)} z_j^{(t)}$$

where σ is the transfer function sigmoid and ϕ is the transfer function tanh

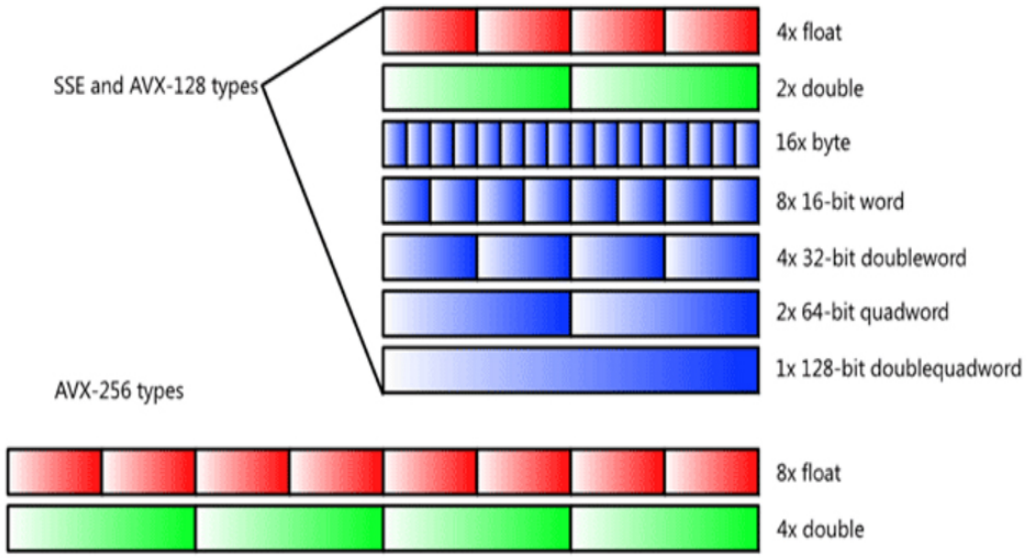
Hidden Layer to Output Layer

$$\begin{pmatrix} who_{11} & \cdots & who_{1i} \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ who_{j1} & \cdots & who_{ji} \end{pmatrix} \begin{pmatrix} hl_1 \\ hl_2 \\ .. \\ .. \\ hl_j \end{pmatrix} = \begin{pmatrix} ol_1 \\ ol_2 \\ .. \\ ol_i \end{pmatrix}$$

After forwarding to the output layer a softmax transfer function is applied for classification probabilities:

$$y_i = \varphi(ol)_i = \frac{e^{ol_i}}{\sum_{k=1}^{K} e^{ol_k}} = \frac{e^{who_i^T hl}}{\sum_{k=1}^{K} e^{who_k^T hl}}$$

Given that most computative operations are of matrix * vector and with mini batching matrix * matrix form, vectorizing loops can greatly improve the networks performance. For calculations performed on the cpu making use of Intel's SIMD (Single Instruction, Multiple Data) instruction set extension can allow speeds to be multiple times faster at the loss of floating point precision.



When vectorizing with simd instructions the cpu utilizes multiple registries at once to perform parallel computations to a certain bit size. SSE (Streaming SIMD Extensions) allows for 128bits operations in parallel which is 2 double-precision floating point operations, 4 single-precision floating point operations, 8 half-precision floating-point and 16 8bit integers all in parallel.
An addition to SSE, AVX (Advanced Vector Extensions) allows for 256bit parallel computations with AVX 2.0 allowing 512bit although 4 double operations with AVX is not twice the speed of 2 double-precision operations with SSE.
Ultimately with these intruction sets cpus are becoming more parallel and although parallelism in GPUs (CUDA) hugely outperforms parallelism in cpus, wider distribution of cpus (mobiles) makes potential speed increases from cpu SIMD operations very useful.

3 Back Propagation Through Time

3.1 via Classification

Multi-Class Cross Entropy

$$L(t,y) = -[\sum_{i=1}^{M}\sum_{c=1}^{C} 1_{(t_i=c)} * ln(y_{ic})] = -[\sum_{i=1}^{M}\sum_{c=1}^{C} 1_{(t_i=c)} * ln(\frac{e^{who_i^T hl}}{\sum_{k=1}^{K} e^{who_k^T hl}})]$$

The derivative with respect to ol_i

$$\frac{\partial L}{\partial ol} = -\sum_{i=1}^{M}[\sum_{c=1}^{C} \frac{t_c}{y_c}\frac{\partial y_c}{\partial ol_i}]$$

Derivative of the softmax activation function at y_c with respect to ol_i

$$\text{if } c = i : \frac{\partial y_c}{\partial ol_i} = \frac{\partial y_i}{\partial ol_i} = \frac{\partial \frac{e^{ol_i}}{\sum_k e^{ol_k}}}{\partial ol_i} = \frac{(\frac{\partial}{\partial ol_i} e^{ol_i})(\sum_k e^{ol_k}) - e^{ol_i}(\frac{\partial}{\partial ol_i}(\sum_k e^{ol_k}))}{(\sum_k e^{ol_k})^2}$$

$$= \frac{e^{ol_i}(\sum_k e^{ol_k}) - e^{ol_i} e^{ol_i}}{(\sum_k e^{ol_k})^2} = (\frac{e^{ol_i}}{\sum_k e^{ol_k}}) - (\frac{e^{ol_i}}{\sum_k e^{ol_k}})^2 = \varphi(ol)_i - \varphi(ol)_i^2$$

$$= y_i(1 - y_i)$$

$$\text{if } c \neq i : \frac{\partial y_c}{\partial ol_i} = \frac{(\frac{\partial}{\partial ol_i} e^{ol_c})(\sum_k e^{ol_k}) - e^{ol_c}(\frac{\partial}{\partial ol_i}(\sum_k e^{ol_k}))}{(\sum_k e^{ol_k})^2}$$

$$= \frac{(0) - e^{ol_c}(e^{ol_i})}{(\sum_k e^{ol_k})^2} = -(\frac{e^{ol_i}}{\sum_k e^{ol_k}})(\frac{e^{ol_c}}{\sum_k e^{ol_k}}) = -\varphi(ol)_i \varphi(ol)_c$$

$$= -y_i y_c$$

therefore

$$\sum_{c=1}^{C} \frac{t_c}{y_c}\frac{\partial y_c}{\partial ol_i} = \frac{t_i}{y_i}\frac{\partial y_i}{\partial ol_i} + \sum_{c\neq i}^{C} \frac{t_c}{y_c}\frac{\partial y_c}{\partial ol_i} = \frac{t_i}{y_i}(y_i(1-y_i)) + \sum_{c\neq i}^{C} \frac{t_c}{y_c}(-y_i y_c)$$

$$= t_i - t_i y_i + \sum_{c\neq i}^{C}(-t_c y_i) = t_i + \sum_{c=1}^{C}(-t_c y_i) = t_i - y_i \sum_{c=1}^{C}(t_c) = t_i - y_i$$

derivative of loss function with respect to the whole output layer

$$\frac{\partial L}{\partial ol} = -\sum_{i=1}^{M}[t_i - y_i] = \sum_{i=1}^{M}[y_i - t_i] = \sum_{i=1}^{M}[\delta_i]$$

$$\therefore \frac{\partial L}{\partial ol_i} = y_i - t_i = \delta_i$$

3.2 Weight Updates

$$\frac{\partial L}{\partial who_{ji}} = \frac{\partial L}{\partial ol_i}\frac{\partial ol_i}{\partial who_{ji}} = \delta_i \frac{\partial ol_i}{\partial who_{ji}} = \delta_i h_j$$

Adagrad:

$$\Delta whoCache_{ji} \leftarrow whoCache_{ji} + (\delta_i h_j)^2$$

$$\Delta who_{ji} \leftarrow who_{ji} - \eta * \frac{(\delta_i h_j)}{\sqrt{\Delta whoCache_{ji} + 10^{-8}}}$$

Hidden Layer Gradients

$$\frac{\partial L}{\partial hl_j} = \frac{\partial L}{\partial ol}\frac{\partial ol}{\partial hl_j} = \sum_{i=1}^{M}[y_i - t_i]\frac{\partial ol}{\partial hl_j} = \sum_{i=1}^{M}\delta_i who_{ji}$$

LSTM Derivatives

LSTM internal Backward Pass

$$\frac{\partial hl_j}{\partial cell_j} = (\sum_{i=1}^{M}\delta_i who_{ji}) * oGate_j * (1 - z_j z_j) * iGate_j * (1 - g_j g_j)$$

$$\frac{\partial hl_j}{\partial iGate_j} = (\sum_{i=1}^{M}\delta_i who_{ji}) * oGate_j * (1 - z_j z_j) * g_j * (\sigma(iGate_j)(1 - \sigma(iGate_j))$$

$$\frac{\partial hl_j}{\partial oGate_j} = (\sum_{i=1}^{M}\delta_i who_{ji}) * \sigma(oGate_j) * z$$

$$\frac{\partial hl_j}{\partial fGate_j} = (\sum_{i=1}^{M}\delta_i who_{ji}) * oGate_j * (1 - z_j z_j) * (\sigma(fGate_j)(1 - \sigma(fGate_j)) * cell_j^{(t-1)}$$

Internal Cell Bridge Weight Updates:
Adagrad:

$$\Delta wiCache_j^{(c)} \leftarrow wiCache_j^{(c)} + (\nabla iGate_j * cell_j^{(t-1)})^2 : wi_j^{(c)} \leftarrow wi_j^{(c)} - \eta * \frac{(\nabla iGate_j * cell_j^{(t-1)})}{\sqrt{\Delta wiCache_j^{(c)} + 10^{-8}}}$$

$$\Delta wfCache_j^{(c)} \leftarrow wfCache_j^{(c)} + (\nabla fGate_j * cell_j^{(t-1)})^2 : wf_j^{(c)} \leftarrow wf_j^{(c)} - \eta * \frac{(\nabla fGate_j * cell_j^{(t-1)})}{\sqrt{\Delta wfCache_j^{(c)} + 10^{-8}}}$$

$$\Delta woCache_j^{(c)} \leftarrow woCache_j^{(c)} + (\nabla oGate_j * cell_j^{(t-1)})^2 : wo_j^{(c)} \leftarrow wo_j^{(c)} - \eta * \frac{(\nabla oGate_j * cell_j^{(t-1)})}{\sqrt{\Delta woCache_j^{(c)} + 10^{-8}}}$$

Updating Input and Context Weights
...