Reconfigurable Computing

Tutorial 1 - LSTM Network (Familiarlisation and Testbench)

Philip Leong

September 11, 2017

1 Introduction

The source files required to complete this tutorial are available from `https://github.com/phwl/hlslstm`.

The long short-term memory (LSTM) network has been used with great success in time-series prediction problems. In this series of tutorials, we will develop an FPGA implementation of an LSTM. This tutorial has the following goals:

- Practise translating published mathematical descriptions into hardware designs
- Gain experience in using high level synthesis
- Gain experience in design optimisation.

Our implementation will have identical functionality to BasicLSTMCell[1] which is used in Google's Tensorflow neural network package. This, in turn, is an implementation of the network published in reference [1].

Using the same notation as [1], let $h_t^l \in \mathbb{R}^{n_l}$ be a hidden state in layer l at timestep t, $T_{m,n} : \mathbb{R}^n \to \mathbb{R}^m$ be an affine transformation ($Wx + b$ for some $W \in \mathbb{R}$ and b), $\odot$ elementwise multiplication, sigm is the elementwise sigmoid function, tanh is the elementwise hyperbolic tangent, and h_t^0 be an input vector at timestep t. We use the activations h_t^L to predict y_t where L is the number of layers in the LSTM.

The neural network implements a state transition

$$LSTM : h_t^{l-1}, h_{t-1}^l, c_{t-1}^l \to h_t^l, c_t^l \tag{1}$$

where

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} = T_{(n_{l-1}+n_l),(4n_l)} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \tag{2}$$

$$\begin{aligned} c_t^l &= f \odot c_{t-1}^l + i \odot g && (3) \\ h_l^t &= o \odot \tanh(c_t^l) && (4) \end{aligned}$$

[1] `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/ops/rnn.py`

2 Laboratory Questions

In answering these questions, marks will be awarded not only for correctness but also understandability and elegance of the solution. Your answers to this laboratory should be in the form of a simple report. For each question below, provide a listing of the changes to the original code in your report.

2.1 (20%) Familiarisation with xsimple.py

Download the tutorial source files using the command

```
git clone git@github.com:phwl/hlslstm.git
```

You can check that everything is working by typing:

```
cd hlslstm/src
make test
```

You should see output similar to the below.

```
python xsimple.py >xsimple.out
g++ -p   -o simple simple.cpp -lm
grep pred xsimple.out > /tmp/xsimple.out
echo "testing simple ..."
testing simple ...
(./simple | grep pred | sdiff -w80 /tmp/xsimple.out -)
vprint(y_pred) -0.0055398695 -0.01926   vprint(y_pred) -0.0055398695 -0.01926
vprint(y_pred) -0.012691636 -0.027500   vprint(y_pred) -0.012691636 -0.027500
vprint(y_pred) -0.019167556 -0.028273   vprint(y_pred) -0.019167556 -0.028273
vprint(y_pred) -0.021765375 -0.026982 | vprint(y_pred) -0.021765375 -0.026983
rm -f /tmp/xsimple.out
```

The output is the result of an sdiff command which compares the y_pred output of a floating-point python implementation of the LSTM (on the left hand side) with the output of the gen.cpp C program (on the right hand side). If the lines of text are different (as in the case in the last line), a | appears between them. Note that the simple program prints a number of variables, but (make **test**) only compares the y_pred outputs. The initial weight values are random so each time the xsimple.py program generates simple, the outputs will be different.

Modify the program so it accepts vectors of length 32 and produces output predictions of length 32.

2.2 (40%) Testbench

Also modify the program so it has a main program (which we will call the *testbench*), that checks if the relative error of each individual output is less than 10%. The program should print "PASS" and the mean squared error (MSE) if correct. Otherwise it should print a diagnostic test message indicating which test pattern and output is in error (the expected output is provided in the l_y[] array defined in gen_io.h). Change the data type used for calculation from double to float. Rerun and see if it passes the testbench. Is the MSE the same?

2.3 (40%) Vivado HLS Project

Create a Vivado HLS project[2] and import the single precision version of your simple.cpp program. Generate a synthesis report for the project, with a target clock period of $10ns$. How many cycles does the execution take? What is the actual clock period?

[2] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug871-vivado-high-level-synthesis-tutorial.pdf

References

[1] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.