Reconfigurable Computing

Tutorial 1 - LSTM Network

Philip Leong

September 7, 2017

1 Introduction

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 software. This, in turn, is an implementation of a regularisation scheme published in reference [1].

Using the same notation as [1], let $h^l_t \in \mathbb{R}^{n_l}$ be a hidden state in layer l at timestep t, $T_{m,n} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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^0_t be an input vector at timestep t. We use the activations h^L_t to predict y_t where L is the number of layers in the LSTM.

The neural network implements a state transition

$$LSTM : h^{l-1}_t, h^l_{t-1}, c^l_{t-1} \rightarrow h^l_t, c^l_t \tag{1}$$

where

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

$$c^l_t = f \odot c^l_{t-1} + i \odot g \tag{3}$$

$$h^t_l = o \odot \tanh(c^l_t) \tag{4}$$

[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.

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.

Change 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 simple.cpp program. Generate a synthesis report for the project. How many cycles does the execution take? What is the clock period?

References

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

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