Project#2 Final Report

Mohd. Arij (MT18208)

Shivam Aggarwal (2016195)

Shivani Mamodia (MT18223)

1. Application and Recurrent Neural Network (RNN) Architecture Overview

Recurrent Neural Networks are the most popular and effective class of networks to use when there is dependence among the data points. They allow to exploit the sequential nature of information and helps which helps to scale the quality of prediction models. However, this advantage comes at the cost of computational complexity. Recent architectures LSTM, GRU involves many dot products just to compute single hidden state. In our project, we attempted to accelerate the inference of LSTM (RNN) architecture by implementing it on FPGA ZC706.

In this project, we trained a LSTM network which converts the English text to German text based on its vocabulary. For this we choose an encoder-decoder architecture variant. The encoder is a LSTM layer that takes the input of latent dimension of 64. It then encodes the input sequence into a fixed length. The final hidden stage of encoder is sent as input to the decoder. The outputs of the encoder are discarded at every time stamp. The Decoder's first input is TAB key (Start of sequence). It generated output, hidden state and current stage. The output of every time stamp is served as input of next time stamp. The Output of decoder is passed through Dense layer and then to Softmax as activation function. This output is used as input to next time stamp. Also, the output of the decoder at every time stamp is the translation in German of every character in English.

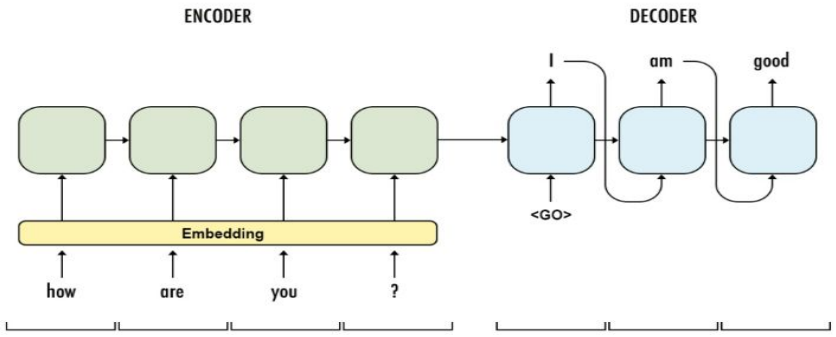


Fig: Word level

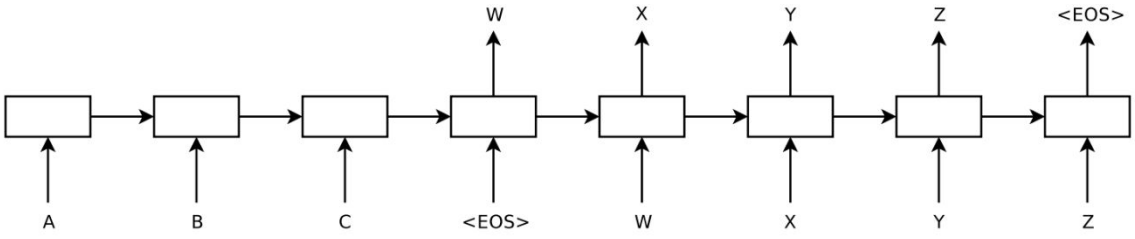
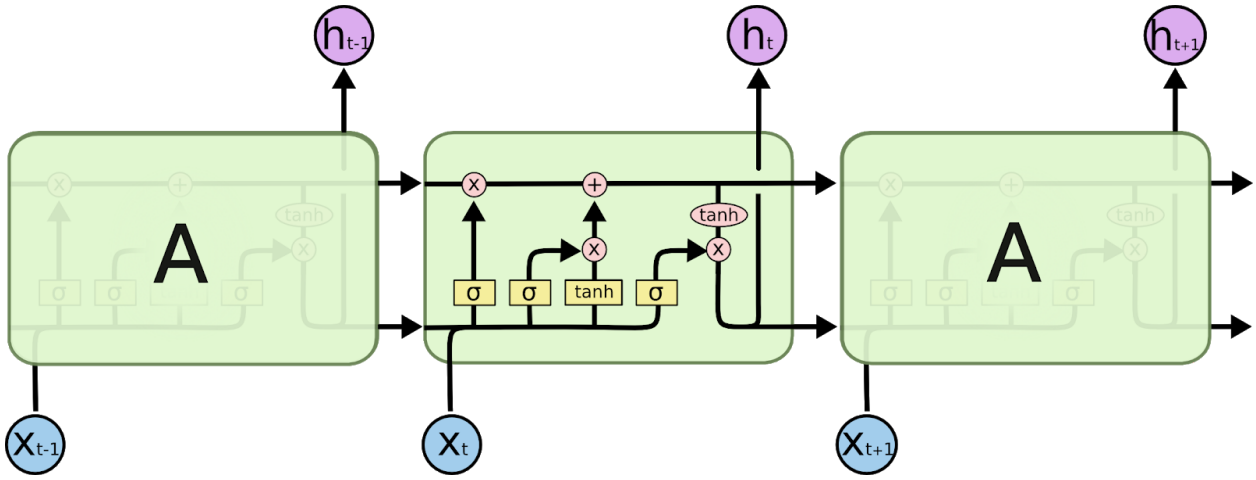


Fig: Character level

MODEL ARCHITECTURE

1. **Embedding layer:** The most logical representation would then be to represent each word by an integer and since vocabulary of your dataset is finite, you would have a finite set of integers representing the words. For the purposes of matrix algebra we represent these integers as an array indices of a 1–0 array where each word is represented by a 1 at an array index of its designated integer. All other indices are 0s. This type of character level representation is called one-hot vector representation of the input words.

2. **Encoder (LSTM):** Long Short Term Memory RNN layer is used as an implementation of Encoder in this architecture. LSTM is the network which is capable of understanding long term dependencies. LSTMs are explicitly designed to avoid long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



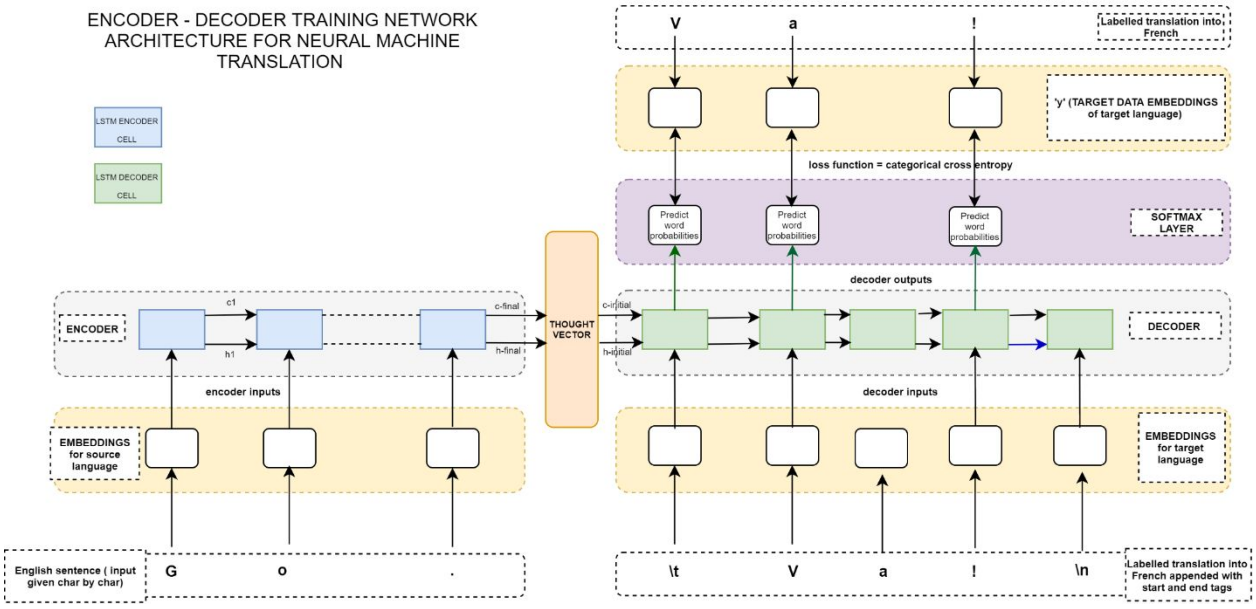
The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

3. **Decoder (LSTM):** The same LSTM implementation is done on the decoder side with the addition of Dense Layer and also the activation layer of softmax.

4. **Fully Connected**: Fully connected layer is responsible for learning the correct mapping from the features it receives from the RNN layer to correct label.

5. **Softmax:** Softmax was used to convert the logits to the probabilities. These probabilities were used to calculate the cross entropy loss which is used to score the current network state.



6. Training of Model and Key Results

Dataset source: **http://www.manythings.org/anki/**

There are over 700,000 English sentences. Over half of these (400,000+) have audio files. It is an extensive dataset with over 30+ different bilingual datasets.

For our task of Neural Machine Translation, we started with an implementation of English to French sentences in keras.

Dataset description:

Number of input samples: 10k

Number of input samples after cleaning data: ~5k

Number of unique input tokens: 70

Number of unique output tokens: 93
Max sequence length for inputs: 16
Max sequence length for outputs: 59

Latent dim	Train Accuracy	Val Accuracy
256	91.89	80.54
16	86.91	76.32

We switched to English to German translation, since it was a much cleaner dataset and we can achieve much better accuracy on the dataset.

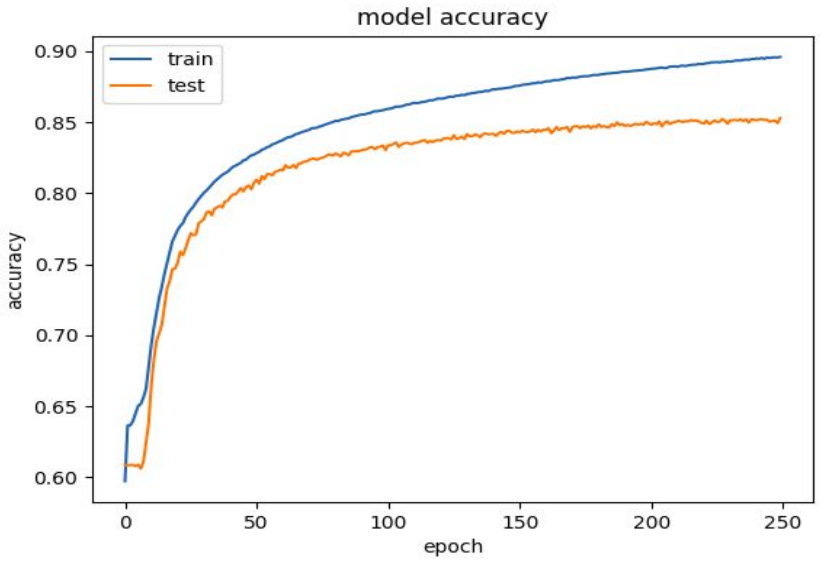
Model size: 653.1kB

English to German Translation:

Trained for around 250 epochs

Number of clean samples: 7156
Number of unique input tokens: 71
Number of unique output tokens: 84
Max sequence length for inputs: 16
Max sequence length for outputs: 45
Train on 5724 samples, validate on 1432 samples

Latent dim	Train Accuracy	Val Accuracy
256	91.14	85.27
128	91.51	85.63
64	**90.49**	**85.57**
32	86.8	84.08



Model Information

Layer	Output Shape	parameters	Connected to
Input_1	(None, None, 71)	0	--
Input_2	(None, None, 84)	0	--
LSTM_1	(None, 64)	34816	input_1
LSTM_2	(None, None, 64)	38144	input_2
Dense_1	(None, None, 84)	5460	Lstm_2[0][0]

Total params: 78,420
Trainable params: 78,420
Non-trainable params: 0

Sample results:

Input sentence: Eat up.
Expected translated sentence: Iss auf.
Decoded sentence: Iss auf.

Input sentence: He ran.
Expected translated sentence: Er rannte.
Decoded sentence: Er rannte.

Input sentence: Hello!
Expected translated sentence:Hallo!
Decoded sentence: Hallo!

Input sentence: Wait!
Expected translated sentence: Warte!
Decoded sentence: Warte!

Input sentence: Help!
Expected translated sentence: Hilfe!
Decoded sentence: Hilfe!

Input sentence: I won!
Expected translated sentence:Ich hab gewonnen!

Decoded sentence: Ich weinte es versuchen.

Input sentence: I know.
Expected translated sentence: Ich weiß.
Decoded sentence: Ich wach sie.

Key Results: OpenCL implementation

Metric	**Reference implementation**	**Your implementation**
Training accuracy* (or error)	NA	90.49
Testing accuracy* (or error)	NA	85.57

OpenCL implementation:

Using standard Sigmoid function:

Average time taken by program on device : 0.042016 sec

Accuracy:0.732317

Using sigmoid approximation:

Time taken by program on device : 0.038085 sec

Accuracy:0.641302

7. OpenCL Implementation of Trained Model

(3.a) Implementation details

Pseudo Code:

```
def LSTMlayer(weight,x_t,h_tm1,c_tm1):
    '''
    c_tm1 = np.array([0,0]).reshape(1,2)
    h_tm1 = np.array([0,0]).reshape(1,2)
    x_t   = np.array([1]).reshape(1,1)

    warr.shape = (nfeature,hunits*4)
    uarr.shape = (hunits,hunits*4)
```

```
barr.shape = (hunits*4,)
'''

warr,uarr, barr = weight
s_t = (x_t.dot(warr) + h_tm1.dot(uarr) + barr)
hunit = uarr.shape[0]
i  = sigmoid(s_t[:,:hunit])
f  = sigmoid(s_t[:,1*hunit:2*hunit])
_c = np.tanh(s_t[:,2*hunit:3*hunit])
o  = sigmoid(s_t[:,3*hunit:])
c_t = i*_c + f*c_tm1: Element-wise multiplication
h_t = o*np.tanh(c_t): Element-wise multiplication
return (h_t,c_t)
```

(3.b) Functional results

Time taken by program on GPU device (NVIDIA GeForce GTX 1050M) : **0.040592 sec**

The output of the respective layers is verified by the keras implementation. The kernal implementation is verified using GPU as well as FPGA also. The machine translation of the English word eg. *Eat up* is converted in to German language *Iss Auf*. This functional verification of the result is done on both GPU and FPGA. The emulation was **successfully** done for the machine translation.

(3.c) Performance results

```
187     queue.enqueueWriteBuffer(buffer_warr, CL_TRUE, 0, 71*256*sizeof(float), weig
188     queue.enqueueWriteBuffer(buffer_uarr, CL_TRUE, 0, 64*256*sizeof(float), weig
189     queue.enqueueWriteBuffer(buffer_barr, CL_TRUE, 0, 1*256*sizeof(float), weigh
190
191     // /* Create the kernel */
192
193     cl::Kernel kernel2(program, "LSTM_decoder");
194
195     char* input_string = "Eat up.        ";
196     int max_encoder_time = 16;
197
198     for (int i=0; i<max_encoder_time; i++) {
199       //char LSTM_inp = input_string[i];
200
201       float* LSTM_input = (float*) malloc(sizeof(float)*71);
202       encode_input(input_string[i], LSTM_input);
203
204       queue.enqueueWriteBuffer(buffer_xt, CL_TRUE, 0, 1*71*sizeof(float), LSTM_i
```

Problems | Console | Guidance | Properties | SDx Log | SDx Terminal

TCF Debug Process Terminal - P2312

```
-0.704140
I
s
s

a
u
f
```

```
Kernel Summary
Kernel Name   Type  Target              OpenCL Library  Compute Units
------------  ----  ------------------  --------------  -------------
LSTM_encoder  clc   fpga0:OCL_REGION_0  LSTM_encoder    1


--------------------------------------------------------------------------------
OpenCL Binary:     LSTM_encoder
Kernels mapped to: clc_region

Timing Information (MHz)
Compute Unit    Kernel Name   Module Name   Target Frequency  Estimated Frequency
--------------  ------------  ------------  ----------------  -------------------
LSTM_encoder_1  LSTM_encoder  sigmoid       100               137.608368
LSTM_encoder_1  LSTM_encoder  LSTM_encoder  100               112.082497

Latency Information (clock cycles)
Compute Unit    Kernel Name   Module Name   Start Interval  Best Case  Avg Case  Worst Case
--------------  ------------  ------------  --------------  ---------  --------  ----------
LSTM_encoder_1  LSTM_encoder  sigmoid       1               27         27        27
LSTM_encoder_1  LSTM_encoder  LSTM_encoder  146867 ~ 0      146866     undef     undef

Area Information
Compute Unit    Kernel Name   Module Name   FF     LUT     DSP  BRAM
--------------  ------------  ------------  -----  ------  ---  ----
LSTM_encoder_1  LSTM_encoder  sigmoid       6778   11361   42   4
LSTM_encoder_1  LSTM_encoder  LSTM_encoder  44872  110540  98   48
--------------------------------------------------------------------------------

```

```
Kernel Summary
Kernel Name   Type  Target              OpenCL Library  Compute Units
------------  ----  ------------------  --------------  -------------
LSTM_decoder  clc   fpga0:OCL_REGION_0  LSTM_decoder    1


--------------------------------------------------------------------------------
OpenCL Binary:     LSTM_decoder
Kernels mapped to: clc_region

Timing Information (MHz)
Compute Unit    Kernel Name   Module Name   Target Frequency  Estimated Frequency
--------------  ------------  ------------  ----------------  -------------------
LSTM_decoder_1  LSTM_decoder  sigmoid       100               137.608368
LSTM_decoder_1  LSTM_decoder  LSTM_decoder  100               121.403419

Latency Information (clock cycles)
Compute Unit    Kernel Name   Module Name   Start Interval  Best Case  Avg Case  Worst Case
--------------  ------------  ------------  --------------  ---------  --------  ----------
LSTM_decoder_1  LSTM_decoder  sigmoid       1               27         27        27
LSTM_decoder_1  LSTM_decoder  LSTM_decoder  186352 ~ 0      186351     undef     undef

Area Information
Compute Unit    Kernel Name   Module Name   FF      LUT     DSP  BRAM
--------------  ------------  ------------  ------  ------  ---  ----
LSTM_decoder_1  LSTM_decoder  sigmoid       6778    11361   42   4
LSTM_decoder_1  LSTM_decoder  LSTM_decoder  158547  319618  768  105
--------------------------------------------------------------------------------

```

4. Extra work (for bonus points eligibility)

1. **Sigmoid Approximation**: We approximated the sigmoid function using some equations by creating the bins of small intervals according the paper. Results are correct upto 3 decimal places for all the input values of x.

TABLE I
THE INTERVAL/FUNCTION FORM

Interval	Function Form
x (-1,1)	y=0.2383x+0.5000
x [-2,-1]	y=0.0467* x^2 +0.1239*x+0.2969
x [-3,-2)	y=0.0298* x^2 +0.2202*x+0.4400
x [-4,-3)	y=0.0135* x^2 +0.1239*x+0.2969
x [-5,-4)	y=0.0054* x^2 +0.0597*x+0.1703
x [-5.03,-5)	y=0.0066
x [-5.2,-5.03)	y=0.0060
x [-5.41,-5.2)	y=0.0050
x [-5.66,-5.41)	y=0.0040
x [-6,-5.66)	y=0.0030
x [-6.53, -6)	y=0.0020
x [-7.6,-6.53)	y=0.0010
x [-∞,-7.6)	y→0
x [1,2)	y=-0.0467* x^2 +0.2896*x+0.4882
x [2,3)	y=-0.0298* x^2 +0.2202*x+0.5600
x [3,4)	y=-0.0135* x^2 +0.1239*x+0.7030
x [4,5)	y=-0.0054* x^2 +0.0597*x+0.8297
x [5,5.0218)	y=0.9930
x [5.0218,5.1890)	y=0.9940
x [5.1890,5.3890)	y=0.9950
x [5.3890,5.6380)	y=0.9960
x [5.6380,5.9700)	y=0.9970
x [5.9700,6.4700)	y=0.9980
x [6.4700,7.5500)	y=0.9990
x [7.5500,+∞)	y→1

2.**tanh approximation**:

tanh= 2*sigmoid(2*x) - 1;

The tanh function is used by using the same approximation of the sigmoid function.

The result of the implementation of sigmoid function shows that the absolute errors between the simulation value based FPGA and theoretical value based MATLAB are no more than 0.0033, which shows the non-linear approximation.

The implementation of these functions by using such approximation leads to the reduction in the runtime as well as area. This is an optimization technique which is being used to implement these functions.

5. References: -

[1] Sutskever, I., O. Vinyals, and Q. V. Le. "Sequence to sequence learning with neural networks." *Advances in NIPS*(2014).

[2] https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/recurrent_neural_networks/machine-translation-using-rnn.html

[3] https://github.com/gionanide/Neural_Machine_Translation

[4] Z. Xie, "A non-linear approximation of the sigmoid function based on FPGA," *2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI)*, Nanjing, 2012, pp. 221-223.

[5] https://dl.acm.org/citation.cfm?id=3287717

[6] https://machinelearningmastery.com/define-encoder-decoder-sequence-sequence-model-neural-machine-translation-keras/