Pedestrian Motion Prediction using FPGA to Accelerate the Inference

Author:
Pedram Babakhani

Supervisors:
Dr. Josif Grabocka

SoSe 2019

Student Research Project
Master of Science in Data Analytics

Wirtschaftsinformatik und Maschinelles Lernen
Stiftung Universität Hildesheim
Universitatsplätz 1, 31141 Hildesheim

Abstract

Autonomous Driving Application can be divided into three phases, Recognition, Prediction, and Planning respectively. At the Recognition phase, an autonomous driver must be able to recognize objects around it. Prediction aims to predict object movements, such as pedestrian motion prediction. Finally, the autonomous driver must make a decision accordingly. In this project, we are going to focus on pedestrians motion prediction using LSTM architecture. The main idea is using FPGA (Field Programmable Gate Array) as a hardware platform to test the model in less amount of time significantly. Actually, we are facing with a real-time application where testing time is a critical issue. We introduce FPGA as the hardware platform to reduce time and power exponentially in the autonomous driving application. The results demonstrate the capability of FPGAs in comparison to other hardware platforms like GPUs or CPUs.

Contents

List of Figures **ii**

1 Introduction **1**

2 Related Work **2**

3 Methodology **6**

4 Experiments **12**

5 Results **23**

6 Conclusion **26**

7 Future Work **26**

References **27**

List of Figures

1 Deep Neural Network structure overview 3
2 LSTM cell and the unfolded LSTM cell 6
3 LSTM long-term dependencies problem 7
4 The repeating module in a standard RNN contains a single layer 7
5 The repeating module in an LSTM contains four interacting layers 8
6 LSTM Forget Gate 8
7 LSTM Candidate Memory 9
8 LSTM Input Gate 9
9 LSTM Output Gate 10
10 LSTM Network 11
11 Walking Cycle 11
12 Walking Cycle 2 12
13 TUD Crossing Tracking 12
14 Experimental setup 13
15 TE0712 including FPGA Artix-7 15
16 FPGA Architecture 16
17 LSTM Matrix Overview 17
18 LSTM Block Diagram 19
19 Activation Functions 20
20 Activation Function Approximation 20
21 Pipeline of LSTM 21
22 Training and Validation Loss graph 23
23 Training and Validation accuracy graph 23
24 Simulation Result 24
25 Execution time comparison 25
26 Frequency comparison 25

1 Introduction

Nowadays, a lot of attempts has done to build things autonomous. Autonomy, is the final goal for a lot of industries making intelligent devices. Perhaps the most popular examples of such a notion are autonomous robots [1] doing things autonomously. A good example of such robots are vacuum cleaners all of us have in our home. However, autonomous vacuum cleaners do exist everywhere but they are not the most important application of autonomy. One of the industries that mostly benefits from autonomy is vehicle industry. There exist numerous companies trying to build autonomous vehicles such as helicopters [2], airplanes [3] and cars [4], [5]. There exist a lot of large scale companies working on autonomous vehicles like Bosch car multimedia [6], BMW [7], Ford [8], Google [9], Volkswagen [10] etc. However, most of these companies has a patent car that can drive autonomously, but still you do not see such cars in roads. Thats because autonomous cars are very critical vehicles and must not have any errors. Having a bug can easily result in fatality.

Artificial intelligence (referred to as AI) in general and specifically machine learning (referred to as ML) are key factors to incorporate autonomy into vehicles. There has been a lot of attempts to build self driving cars in past decades using machine learning methods. AI and ML methods are used in two parts of the autonomous car [4]. First, they are used to interpret the captured data from sensors like cameras, laser, etc. to understand the scene objects. Second, they are utilized to make decisions based on the interpreted scene to control the steering wheel, break and throttle. In this research project we are going to work on the first part. We believe that this part plays very important role in this area because the better the agent can interpret and understand the scene object, the better decision will be made and thus, the more reliable the agent can be.

Problem Statement In order to train the AI agent, the agent needs input to make a decision and the input is nothing else but data captured by sensors. There are many scenarios where the agent has to make decision in, such as staying in the path and following traffic signs. Most of them are likely to be constant and do not vary a lot. We are interested in such sudden scenarios where the agent has to make decision immediately, for example pedestrians, cyclists and other moving objects crossing the street or moving in front of the car. In order to make decision based on these scenarios, the agent has to have some intuitions of predicting what the objects will do in near future. The problem is the agent has only stream of images as the input and has to predict a movement based only on images. There has been some researches in this motion detection area. We want to apply this idea and focus on the driving scenarios.

In driving scenarios, every second matters a lot and the scene can be drastically changed within seconds. Therefore, we are trying to accelerate the time for agent to interpret the scenes such that the model can be used in real time. In this project we focus only on pedestrians motion prediction which will be trained and implement on FPGA to be tested in less amount of time significantly. Actually we are facing with a real time application which time matters a lot especially in inference. So we introduce FPGA (Field Programmable Gate Array) as the hardware platform to reduce especially time and power in autonomous driving application.

2 Related Work

Autonomous Driving Application can be divided into three phases, Recognition, Prediction and Planning respectively. Recognition is the first phase, actually autonomous driver must recognize objects around such as pedestrian recognition. Prediction aims predict object movements, such as pedestrian motion prediction. Finally, autonomous driver must make a plan which is the last phase. Autonomous driving applications have wide range of aspect regarding the objects or roads, in this project we are going to focus only on pedestrians motion prediction which will be trained and implemented on FPGA to be tested in less amount of time significantly. Actually we are facing with a real time application where time matters a lot especially in inference. We introduce FPGA (Field Programmable Gate Array) as hardware platform to reduce especially time and power in autonomous driving application. FPGAs are quite well known in embedded system industry used in real autonomous driver cause actually we will end up with hardware in real application.

Seeing the rise of Autonomous Driving application, there has been competitions between hardwares such as FPGA and GPU to offer a hardware platform that can reduce the computation time significantly and efficiently. As Deep Learning has driven most of the advanced autonomous driving applications, it is regarded as the main comparison point[5].

Even though GPU vendors have aggressively positioned their hardware as the most efficient platform for this new era, FPGAs have shown a great improvement in both power consumption and performance in Deep Neural Networks (DNNs) applications, which offer high accuracies for important image classification tasks and are therefore becoming widely adopted [11]. As there are various trade-off to consider, we bring some benefits and barriers of using FPGAs and GPU, and consider the studies performed by the main players in this field; namely Xilinx, Intel, Microsoft and UCLA research labs.

DNNs are widely used as learning models because of their inference accuracies. They can be formulated as graphs similar to the one shown in Figure 1.

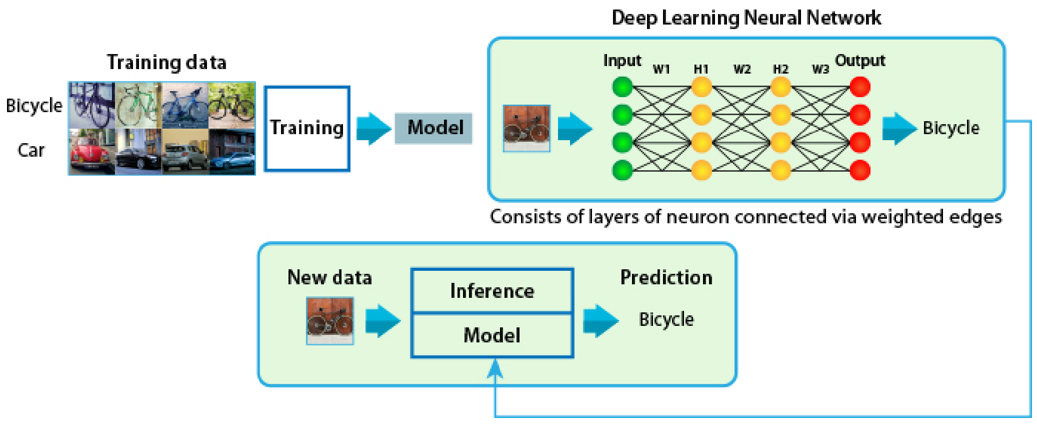


Figure 1: Deep Neural Network structure overview
source: https://www.aldec.com/en/company/blog/167--fpgas-vs-gpus-for-machine-learning-applications-which-one-is-better

The differences between FPGA and GPU can be classified into four main categories of hardware : Raw compute power, Efficiency and power, Flexibility and ease of use, and Functional Safety. There were many researches published by Xilinx [11], Intel [12], Microsoft [13] and UCLA [14] and the below content is derived from that source.[15]

1. **Raw Compute Power**: The researches done on Xilinx signifies that the Tesla P40 with Ultrascale+TM XCVU13P FPGA has almost the same compute power (40 INT8 TOP/s) and (38.3 INT8 TOP/s) respectively. FPGAs perform significantly higher in compute capability, when it comes to on-chip memory, which contributes in reducing the latency in deep learning applications and also high amount of on-chip cache memory used reduces the memory bottlenecks which is associated with external memory access. It also reduces the power and costs of a high memory bandwidth solution. In the application of Deep Neural Network the FPGAs support the full range of data types precisions, e.g., INT8, FTP32, binary and any other custom data type. Since the deep learning applications are evolving at a faster rate to catch up with this demand, GPU vendors must upgrade the existing architectures to

stay up-to-date. Therefore, the FPGAs come in handy because of its re-configurability and hence users can implement any custom data type into their design.[15]

2. **Efficiency and Power**: FPGAs are popular mainly because of their power efficiency. Several research project were done to test the performance of these hardware. Microsoft on an image classification project showed that Arria 10 FPGA performs almost 10 times better in power consumption. On the other research in general purpose compute efficiency by Xilinx, they claim that the Xilinx Virtex Ultrascale+ results almost four times better than NVidia Tesla V100. The research concludes that although the NVidia V100 provides a significant efficiency to the Xilinx FPGAs due to the efficient Tensor Cores for tensor operations for recent deep learning workloads, it is unpredictable for longitivity of NVidia's Tensor Cores remaining efficient for deep learning applications. For a huge number of end applications and its workloads the re-configurability of FPGAs provides much higher efficiency in addition to the software development stack of main vendors such as Xilinx (SDAccel) and Intel (FPGA SDK for OpenCL).[15]

3. **Flexibility and Ease-of-Use**: FPGAs can be programmed to add different steps or outputs altogether, allowing growth beyond existing GPU support without physically changing the way the GPUs are architected.[16] Given the existing system that has been running for a while and we want to add some new steps to the running program. If we use FPGA, we can reprogram it to insert the new steps easily, but if we run the program on GPU, we need some additional GPUs as well as programming to adjust the new steps to the existing program. The flexible architecture of FPGAs has shown great potential in sparse networks, which is one of the hot trends in current machine learning applications.[15] Another reason why FPGA is flexible is the compatibility for any-to-any I/O connection. In the aspect of easiness, GPUs are easier to use than the FPGA. However, there is an easy development platform for the users called Xilinx which provides tools such as SDoC, SDAccel and Vivado HLS. These tools have made the designing process of FPGA much easier for software engineers, as they can easily convert their C/C++ code to the HDL.[15]

4. **Functional Safety**: GPUs are originally designed for consumer graphics such as rendering high quality images for display, especially for gaming. In that particular case, functional safety is not a necessity because GPU only deals with the visual effect and nothing to be concerned as

harmful. On the other hand, the applications such as ADAS (Advanced Driver-Assistance Systems) requires functional safety because it is involving human being. Therefore, GPU itself is not really applicable for this field due to its unreliability. The hardware proposed in this project, which is FPGA, have been used for some applications requiring functional safety such as ADAS, medical devices and aerospace controls.[16] FPGAs have been designed in way to meet the safety requirement of wide range of applications including ADAS. In this respect, Xilinx Zynq®-7000 and Ultrascale+TM MPSoC devices are designed to support safety-critical applications such as ADAS.[15]

There are some researches conducted for accelerating the computation of the deep learning network. In [17], the authors worked on LSTM-RNNs for real-life speech recognition problem. They believe that the combination of LSTM and RNNs improves the prediction accuracy but the computation pattern and data access pattern more complex. Therefore, it is quite difficult for CPUs to accomplish remarkable performance on real-life problem where we need to infere the speech as quickly and precisely as possible. It means that a high-performance accelerator is hardly required. One of the alternatives goes to FPGA-based accelerator, if taking performance, energy-efficiency and flexibility into consideration. The authors focus on accelerating the on-line inference time of LSTM-RNNs and the result is FPGA-based accelerator for LSTM-RNNs that optimizes both computation performance and communication requirements outperforms previous approaches[18].

In this project, we implement a LSTM on FPGA and test the model and measure the time and power consumption then compare it to state of the art. We will see that FPGAs are capable to beat GPUs regarding the time and power. There has been successful researches in motion detection area. They employed Deep Learning architecture using Convolutional Neural Network (CNN) because CNN is well-known for its capability of dealing with images. In this project, we are going to employ Recurrent Neural Network (RNN) which is known for its capability to work with time series data. We use RNN because we are dealing with sequence of data that is correlated within its sequence. The movement of an object at current time is affected by the previous movement and so is the next movement. In addition, we are also using Long Short-term Memory (LSTM). LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. LSTMs help to retain history for short time which is helpful to train the model better. After training the model, we transfer that model to a hardware called FPGA for testing.

3 Methodology

There has been successful researches in motion detection area using CNN and RNN. In this project, we employ Recurrent Neural Network (RNN) which is known for its capability to work with time series data. It also deals with sequence of data that is correlated within its sequence. In addition, we also use Long Short-term Memory (LSTM).

Recurrent Neural Networks Recurrent Neural Networks (RNNs) are one type of neural networks that are capable of handling sequence data, such as sequence of words in sentences and series of observations over time. RNNs have shown some great successful applications, for example language processing, video processing, time-series prediction and human activity recognition. In general neural network, the inputs and the outputs are independent of each other. This may be inappropriate approach for dealing with sequential data, where the current output in sequential data is affected by the previous output along with current input to some extent. That is exactly how RNNs actually work. RNNs process the input sequentially. After getting the output for each timestamp, RNNs retain the output and pass it to the next input.

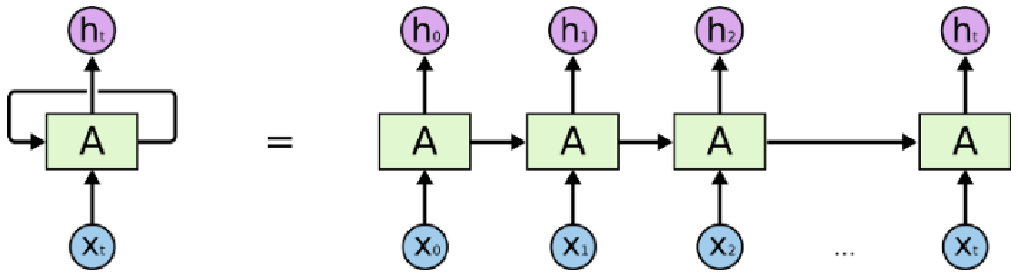


Figure 2: LSTM cell and the unfolded LSTM cell
source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Although RNNs are able to process data sequentially, they are limited to looking back only few steps. For example in word prediction, I am from Germany and I speak ____. The blank space would be filled by a word "German" but the RNNs find it difficult to predict it because the important information, which is Germany, lies few words before the blank space. The word "Germany" does not give much information compared to the word "speak" right before the blank space. This is the problem of long-term dependencies. Hence, LSTM comes to overcome this problem.

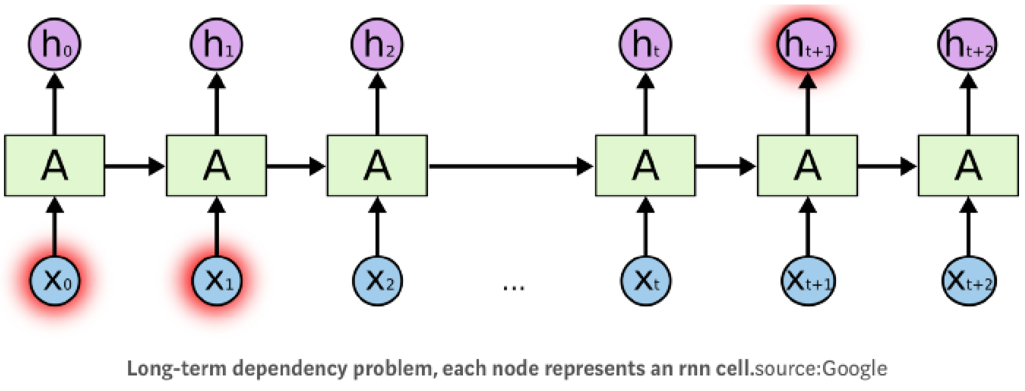


Figure 3: LSTM long-term dependencies problem
source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM) Long Short-Term Memory (LSTM) was first developed by Hochreiter Schmidhuber (1997) as a variant of Recurrent Neural Network (RNN). LSTM is a type of RNNs with capability of handling Long-Term dependencies. 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 [19]

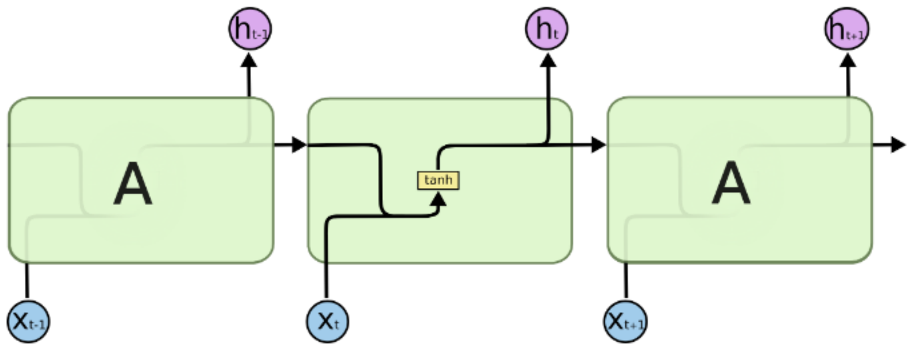


Figure 4: The repeating module in a standard RNN contains a single layer
source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs also have this chain like structure, but the repeating module has a different structure.[19] There are four different networks, which are called gates, interacting in a special way to process the input.

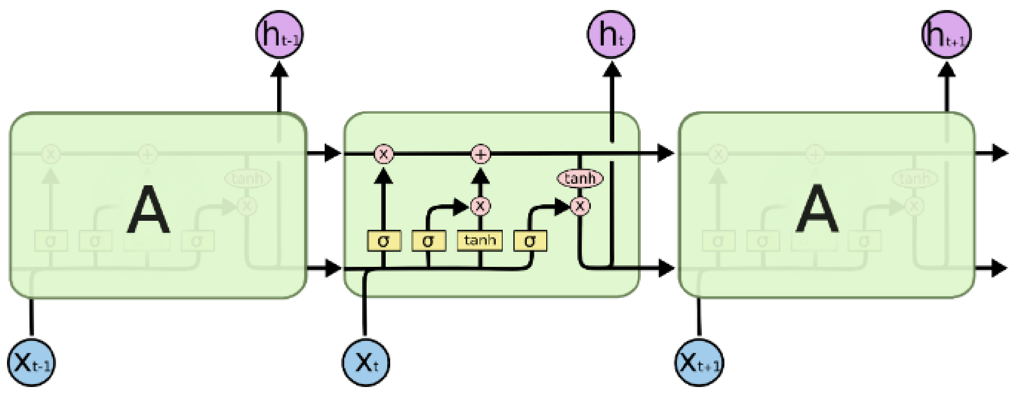
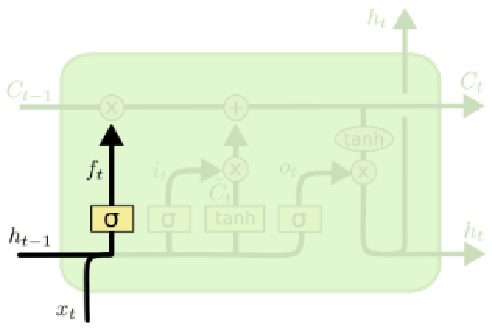


Figure 5: The repeating module in an LSTM contains four interacting layers
source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

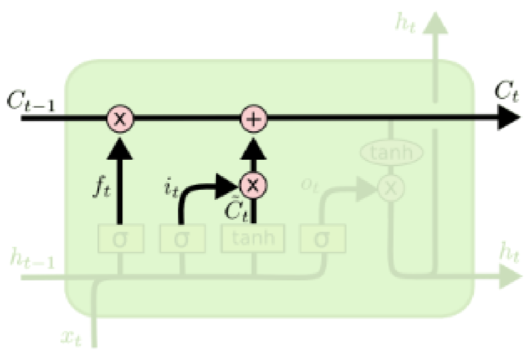
A LSTM cell consists of 4 gates :



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 6: LSTM Forget Gate
source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

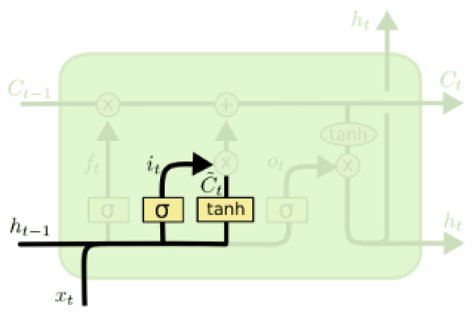
1. **Forget Gate** Forget gate helps the network to decide which information from previous step to be kept and deleted. This gate has its own parameters, which are weights W_f. This weights act as a switch to filter any undesired previous information. Formally, input and output from previous step are multiplied by the weights W_f, then the results are passed into the activation gate with *sigmoid* function.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 7: LSTM Candidate Memory
`source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

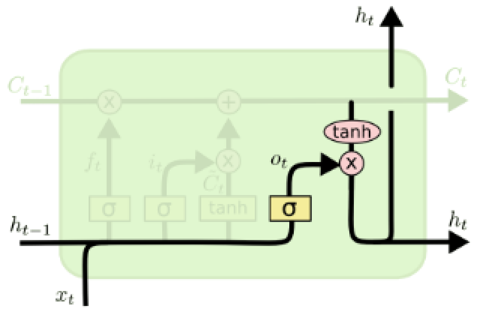
2. **Candidate memory** Candidate memory $\widetilde{C}_t$ decides what information could be proposed as the memory cell c_t. Memory cell is the cell that brings the information throughout the process. For every step, it contains different information depending on the input obtained from the surrounding gates. This gate processes the input and the output from previous step by multiplying with its own weights W_{hc}.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

Figure 8: LSTM Input Gate
`source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

3. **Input Gate** Input gate decides which information could be added to the candidate memory. The output from input gate together with the candidate memory are processed by element-wise product. The results are then processed with the output from forget gate to determine the current memory cell C_t.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 9: LSTM Output Gate
`source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

4. **Output Gate** Output gate decides what information are going to the output o_t. The memory cell C_t goes to *tanh* function and then together with the output o_t are processed with element-wise product to yield the actual output h_t. This output is then sent to the next cell as the short-term memory.

LSTM Formula

$$f_t = \sigma(\,W_{hf}.h_{t\text{-}1} + W_{xf}.x_t + b_f) \tag{1}$$

$$i_t = \sigma(\,W_{hi}.h_{t\text{-}1} + W_{xi}.x_t + b_i) \tag{2}$$

$$C'_t = tanh(\,W_{hC}.h_{t\text{-}1} + W_{xC}.x_t + b_C) \tag{3}$$

$$o_t = \sigma(\,W_{ho}.h_{t\text{-}1} + W_{xo}.x_t + b_o) \tag{4}$$

$$C_t = f_t * C_{t\text{-}1} + i_t * C'_t \tag{5}$$

$$h_t = o_t * tanh(C_t) \tag{6}$$

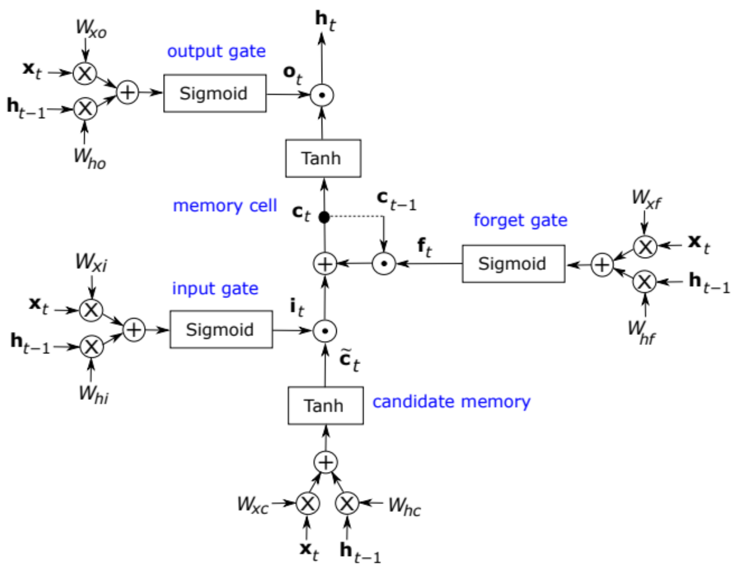


Figure 10: LSTM Network

Problem Setting In the problem setting, we consider a walking pedestrian as a sequence of frames where each frame represents the position of a leg in walk cycle. In the above figure we can observe the repeating sequence of a person after every 25 to 30 frame cycle. Then the problem filters down to just predicting the next frame(movement) given the sequence of previous movements. Therefore we implemented LSTM with 1 and 2 units.

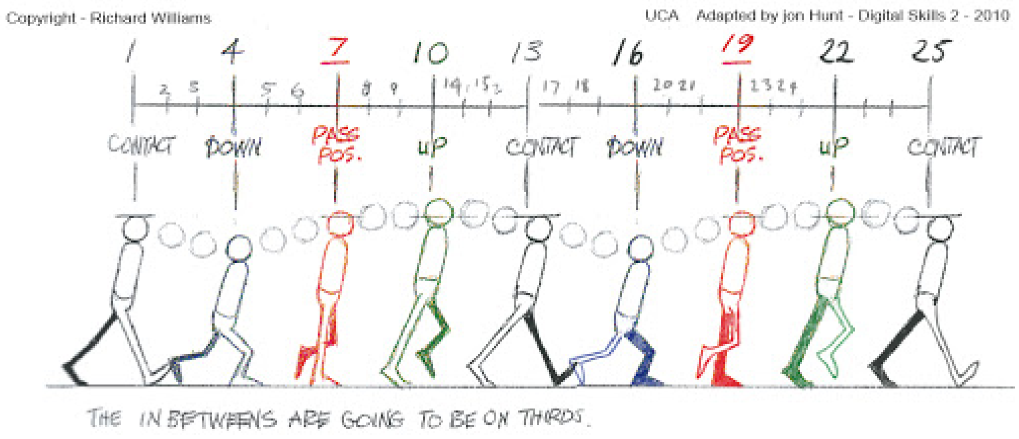

Copyright - Richard Williams
UCA Adapted by jon Hunt - Digital Skills 2 - 2010
1
4
7
10
13
16
19
22
25
CONTACT
DOWN
PASS POS.
UP
CONTACT
DOWN
PASS POS.
UP
CONTACT
THE IN BETWEENS ARE GOING TO BE ON THIRDS.

Figure 11: Walking Cycle
source: `https://www.schoolofmotion.com/blog/walk-cycle-inspiration`

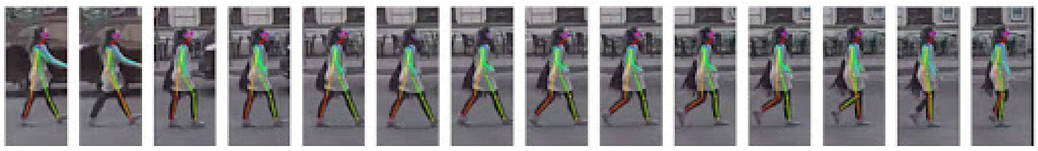


Figure 12: Walking Cycle 2
source: `https://www.researchgate.net/figure/Examples-of-2D-pose-estimation-by-skeleton-fitting-Top-pedestrian-in-side-view-walking_fig2_326681295`

After that, we implement a LSTM on FPGA and test the model and measure the time and power consumption then compare it to state of the art. We see that FPGAs are capable to beat GPUs regarding the time and power.

4 Experiments

In this experiment, we use dataset from TUD crossing tracking. This dataset contains videos of different people crossing a street that have been put into sequence of frames. There are 52 different samples being used to learn the model.

Preprocessing In the preprocessing step, we assign the label to each frame according to our key label set given in figure 11.

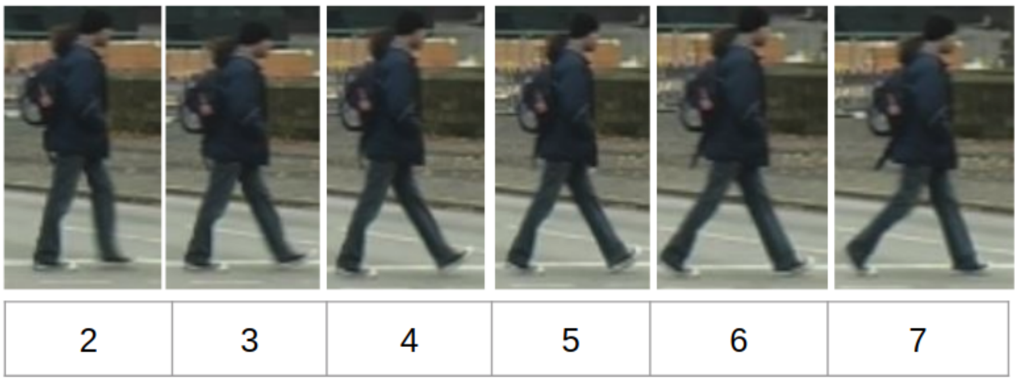


Figure 13: TUD Crossing Tracking
source: `http://www.vision.ee.ethz.ch/~rhayko/paper/eccv2012_riemenschneider_hough_region`

After that, we enlarge our dataset by building the augmented dataset. Given the full sequence of frames for each person. We build a dataset by clipping the sequence every 8 frames then set the next frame to be the output. Supposed we have the sequence with label 1,2,3,4,5,6,7,8,9 and 10 , we can build a dataset as follows:

Input	Output
1,2,3,4,5,6,7,8	9
2,3,4,5,6,7,8,9	10

At the end, we can have 592 instances to build the model. We split the dataset into training and testing dataset. The training dataset has 396 instances and testing dataset has 196 instances.

Learning the model In the learning process, we use supervised learning as we have the ground truth for each instance. We feed n(n=8) many sequences of frames into the LSTM, then output the next predicted frame given the input. According to the Figure 2, we can see those moments such as contacting with the ground, knees going down, passing position, and knees going up. The input for the LSTM are just sequences of the label but we normalize the label to be between 0 and 1 such that it has even distribution. For the output label, we use one-hot encoding with 30 labels, where each label has 30 columns. The label's column is set to be one and the rest of columns is 0. In this experiment, we also compare the performance of the model using 1 LSTM and 2 LSTM. Each model are trained for 300 epoch. At the end, we choose the one with better performance to be implemented in the hardware.

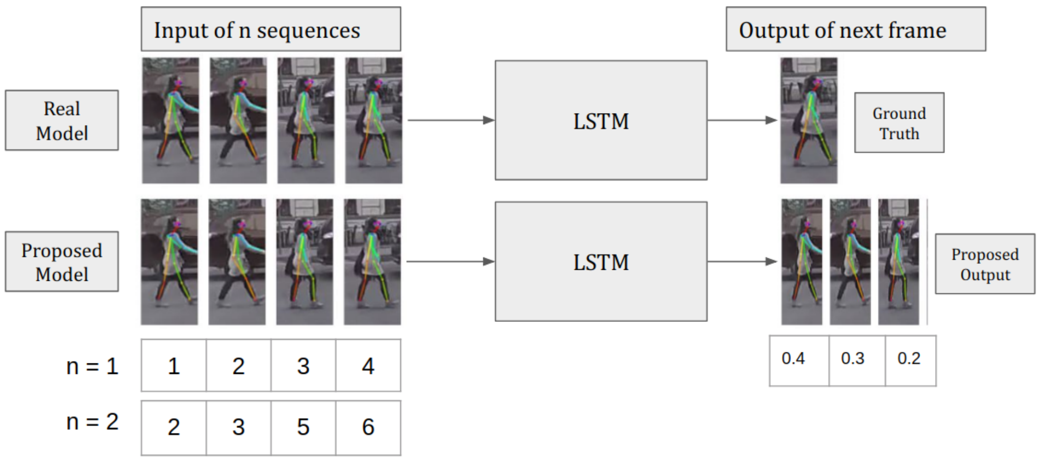


Figure 14: Experimental setup

Evaluation The evaluation metrics that we used are categorical_crossentropy for computing the loss and top-n categorical function for evaluating the accuracy with n=3. The model choose top 3 proposed frames and match them to the ground truth. Whenever the ground truth is contained in the proposed frames, it means that the model predicts the output well. We choose top-n categorical loss instead of just categorical loss as our evaluation metrics because we consider that different people has different walking sequences. Although we mentioned before that we found significant moments of walking cycle every 4 frames, it is a general case but it may vary from person to person, i.e. given the past walking labels as 1,2,3 and 4, it does not necessarily that the next label would be 5 but could be 6 or even 7 because the less remarkable difference in the frames between those significant moments.

Hardware Section After obtaining the model, we transferred the parameters into the hardware FPGA and tested on it. FPGAs are different regarding hardware architecture which can be number of Look Up Table (LUT) , number of I/Os, clock frequency, pin packages and etc. Evaluation Board can be varied based on the hardware components next to the FPGA such as DDR3, Flash Memory, LAN, Artix is one of a well known FPGA family provided by Xilinx. Every family is categorized among different series, every serie has some particular features, 7 series FPGA provides some features like Multi-boot and power efficiency which is not supported in earlier series. The Artix 7 FPGA has been chosen because of the low cost and enough time and power efficiency for the target application. Figure 15 shows the hardware module has been chosen.

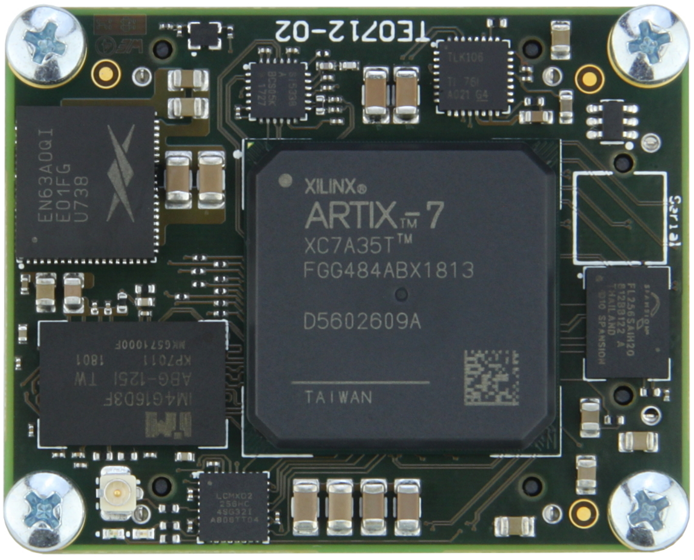


Figure 15: TE0712 including FPGA Artix-7

The TE0712-02 is an industrial-grade FPGA module integrating a Xilinx Artix-7 FPGA, a 10/100 Mbit Ethernet transceiver, 1 GByte of DDR3 SDRAM, 32 MByte Flash memory for configuration and operation, and powerful switch-mode power supplies for all on-board voltages. A large number of configurable I/O's is provided via rugged high-speed stacking strips.

FPGAs are capable to be program partially as soft processor unit. Soft processors means part of FPGA will be programmed to work as a micro controller which called Microblaze. Artix 7 supports Microblaze as soft processing system. On the other hand, hard processors refer to those processors which exist physically next to the FPGA on the evaluation board, On the other words, A physical Microcontroller which mostly it is ARM cortex 8 or 9, works parallel with FPGAs. As we mentioned FPGAs are capable for big matrix multiplication and arithmetic computation, Micro controller are capable to be used as a co-processor to control the flow of data.

More efficiency can be gained by splitting the tasks between FPGA and Controller. Controller will be processing unit system integrated Microblaze. Microblaze will be connected to all hardware components on the evaluation board. The connection is based on Master-Slave. Though the Microblaze is master of all hardware components, it is slave of FPGA. The Architecture of the controller will be like the following.

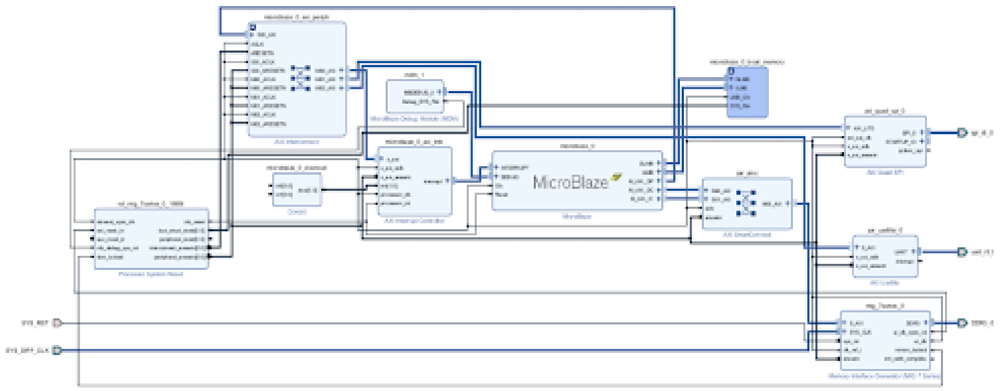


Figure 16: FPGA Architecture

Microblaze is connected to 7 series Mig which it refers to DDR3 SDRAM, Quad SPI refers to 4x channel connection with Flash memory, UART lite refers to interface to communicate with FPGA and Microblaze through JTAG connection. The rest of the components are generated by Xilinx Vivado which are necessary to be added to Microblaze.

The instruction memory and data memory will be DDR3 SDRAM (1GB) which is absolutely enough for weights and biases. Flash Memory will be used as a space memory for Bootloader to boot the Microblaze and FPGA both together. In this case FPGA will be exclusively used for matrix multiplications and activation functions.

Data Representation As we mentioned before, data memory will be DDR3 SDRAM. The DDR3 SDRAM structure is 32 bit * 32 M. All numbers we have in our application are represented in Floating Point format. Furthermore, floating point computation are usually high resources and time demanding. Floating Point numbers must be represented in Q (Fixed Point) format on FPGAs because there is no Floating Point Unit (FPU) on neither FPGAs nor Microblaze. In Q (Fixed Point) is a fixed point number format where the number of fractional bits and integer bits are specified. Q8:8, Q16:16 and Q32:32 represent number of bits for fraction and decimal. Furthermore, floating point computation are usually high resources and time demanding.

We choose Q16:16 data representation as 32 bits Fixed Point to store a number in a sector of DDR3 SDRAM. Q8:8 does not represent numbers as precise as Q16:16 and Q32:32, Additionally, It may lead to overflow. On the other hand Q32:32 will be too big for our application because there is no big numbers with tiny fraction. Negative values will be represented by Two's Complement.

In order to convert the numbers to Q16:16, values must be multiply by

2^{16}, then round up. The result will be represented as binary, the point is a virtual point which exist only theoretically.

For example: 5.625 can be written as :
5.625 * 2^{16} = 368640 = 0000000000000000011.1010000000000000
-5.625 = Two's_Complement(5.625) = 1111111111111111100.0110000000000000

Data Configuration All weight and biases will be stored in DDR3 SDRAM. Based on the number of hidden layers and past sequences, amount of needed memory will be calculated. Figure 17 shows overview of weights, biases, input and output, generally all matrices and their dimensions where D and H are number of inputs and outputs respectively. B is batch size which can be set 1.

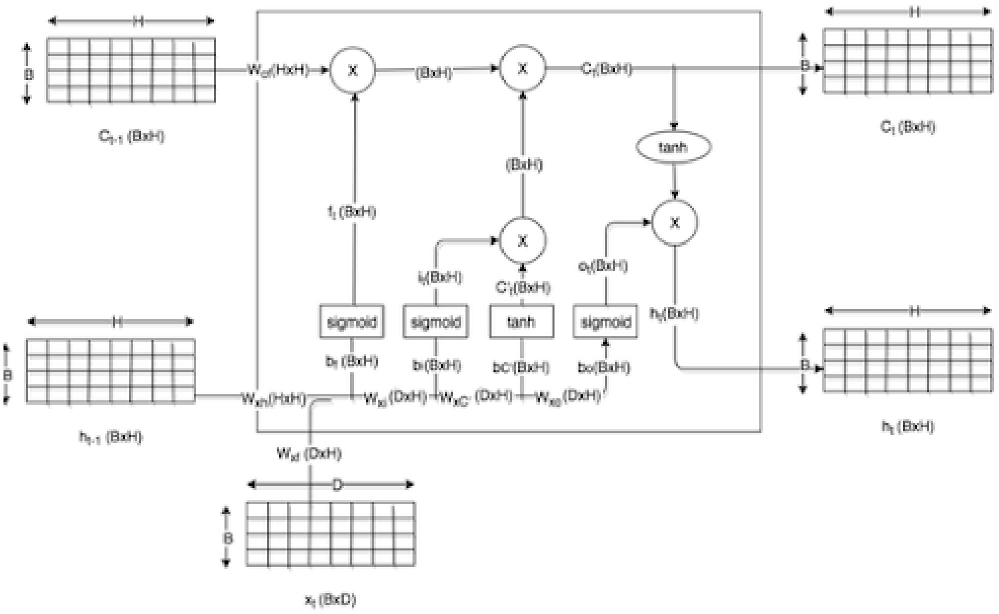


Figure 17: LSTM Matrix Overview

All dimensions are mentioned in the following way :

	$W_{xi} \in \mathbb{R}^D \times \mathbb{R}^H$
$i_t \in \mathbb{R}^B \times \mathbb{R}^H$	$W_{xf} \in \mathbb{R}^D \times \mathbb{R}^H$
$f_t \in \mathbb{R}^B \times \mathbb{R}^H$	$W_{xo} \in \mathbb{R}^D \times \mathbb{R}^H$
$c_t \in \mathbb{R}^B \times \mathbb{R}^H$	$W_{hi} \in \mathbb{R}^H \times \mathbb{R}^H$
$o_t \in \mathbb{R}^B \times \mathbb{R}^H$	$W_{hf} \in \mathbb{R}^H \times \mathbb{R}^H$
$h_t \in \mathbb{R}^B \times \mathbb{R}^H$	$W_{ho} \in \mathbb{R}^H \times \mathbb{R}^H$
$x_t \in \mathbb{R}^B \times \mathbb{R}^D$	$b_i \in \mathbb{R}^B \times \mathbb{R}^H$
$h_{t-1} \in \mathbb{R}^B \times \mathbb{R}^H$	$b_f \in \mathbb{R}^B \times \mathbb{R}^H$
$c_{t-1} \in \mathbb{R}^B \times \mathbb{R}^H$	$b_c \in \mathbb{R}^B \times \mathbb{R}^H$
	$b_o \in \mathbb{R}^B \times \mathbb{R}^H$

In our model, D is 8 and H is 32 so the whole amount of memory space we need will be calculated like the following:

- Input data : $x_t = N_D \times 1 = 8$
- Weights : W_{xi}, W_{xf}, W_{xc}, $W_{xo} \rightarrow N_D \times N_H \quad (8 \times 32 \times 4 = 1024)$
  $W_{hi}, W_{hf}, W_{hc}, W_{ho} \rightarrow N_H \times N_H \quad (32 \times 32 \times 4 = 4096)$
- Biases : b_i, b_f, b_c, $b_o \rightarrow N_H \times 1 \quad (32 \times 1 \times 4 = 128)$
- Required memory space = 32 bits $\times(8 + 1024 + 4096 + 128) = 164KB$

Amount memory we needed to store the weights and biases is 164 KB which is a small proportion of the entire DDR3 SDRAM.For small models which need memory less than 128KB, weights and biases can be stored in local memory of Microblaze which can be restored faster. The FPGA (Artix 7) will be connected to 12 RAMs which are synchronized by global clock of FPGA. All matrices value will be stored into DDR3 SDRAM. W_{xi}, W_{xf}, W_{xc}, W_{xo},W_{hi}, W_{hf}, W_{hc}, W_{ho} and b_i, b_f, b_c, b_o will be stored into nonvolatile memory to be restored at every reset. Figure 6 demonstrates connection of all weights and biases to FPGA unit. Though at every power on the board, Microblaze will start to feed the FPGA by weights and biases parallelly. Every entry of weights and biases will be transferred at one clock cycle. The maximum clock cycle we need to read the weights and biases is size of the biggest matrices which are W_{hi}, W_{hf}, W_{hc}, W_{ho} with size of 32 x 32. So we need around 1024 clock cycles after reset activation.

Hardware Implementation Based on LSTM formulas, computations can be divided into metrics multiplications, matrics additions and activation functions.

Matrix Multiplication has been implemented by generating 32 bits multiplier and adder to multiply the values and add them. 32 bits multiplier and adder has been used from IEEE.NUMERIC_STD library. Figure 18 shows the LSTM overview including the hardware functions,I/O and memories connection.

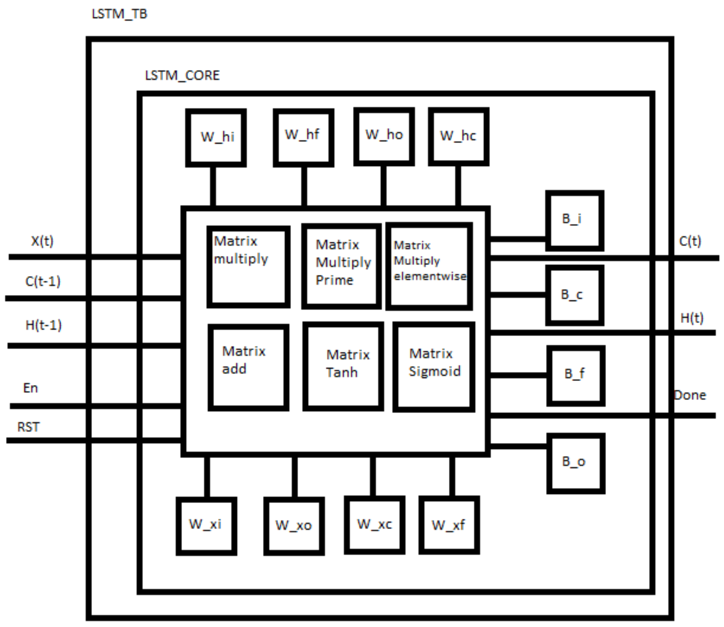
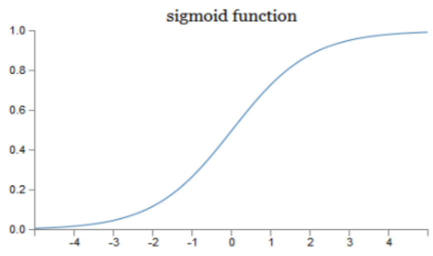
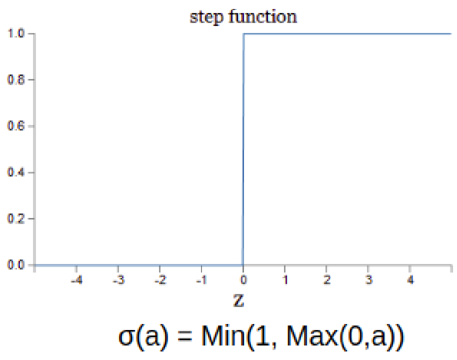


Figure 18: LSTM Block Diagram

Sigmoid functions can be implemented using look up table (LUT) or step function. Step function is not as accurate as look up table (LUT) method because the Sigmoid and Tanh will be approximated by step function. Figure 19 shows the sigmoid and step function respectively. As it has been shown, the step function is not precise approximation.



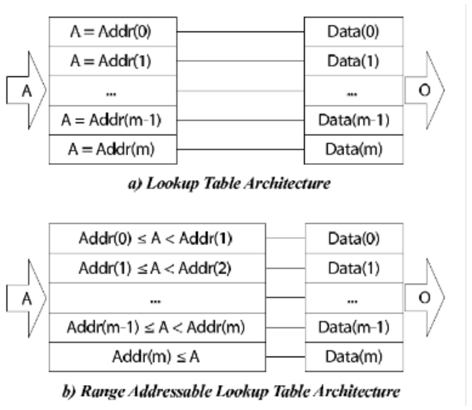
(a) Sigmoid Function



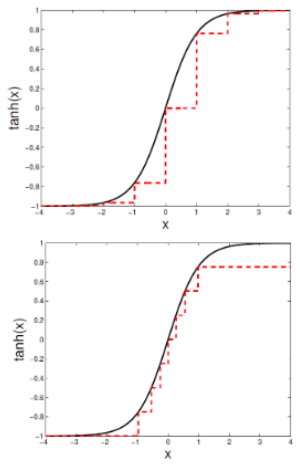
(b) Step Function

Figure 19: Activation Functions

The other method is LUT using Piecewise Linear Approximation. Piecewise linear schemes use a series of linear segments to approximate a function. The Tanh and Sigmoid function have been implemented using Piecewise Linear Approximation. Size of every segment is 0.1 in range of [-4 4] and the average of upper bound and lower bound value of segment will be assign to the Tanh and Sigmoid value for a particular segment. The calculated values are converted to Q16:16 format and will be stored in registers of FPGA. Look up tabel (LUT) architecture for Tanh is shown in Figure 20.



(a) Look Up Table Architecture



(b) Piecewise Linear Approximation

Figure 20: Activation Function Approximation

Pipeline Pipelining is an implementation technique where multiple instructions are overlapped in execution. The computer pipeline is divided in stages. Each stage completes a part of an instruction in parallel. Pipelining does not decrease the time for individual instruction execution. Instead, it increases instruction throughput. The throughput of the instruction pipeline is determined by how often an instruction exits the pipeline. Pipeline technique is implemented by embedding registers between stages, at every clock cycle computation results are written in the registers. At next clock cycle new results have been written to the registers and previous results are shifted to the next stage. The pipeline consists of 7 stages. Figure 21 shows the overview of the pipeline.

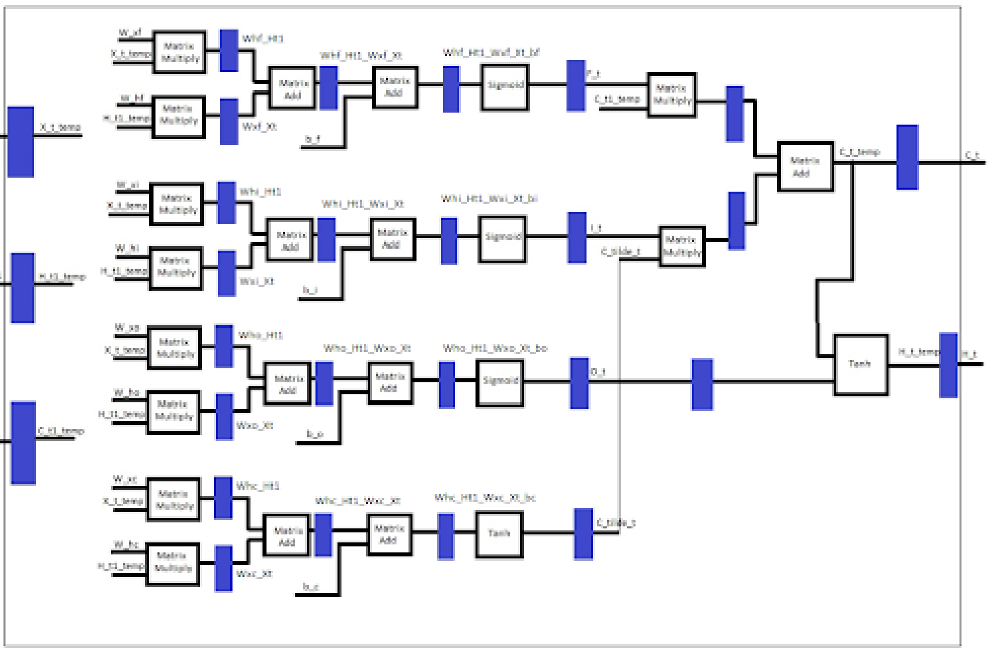


Figure 21: Pipeline of LSTM

The blue rectangles are registers which are synchronized by global clock. After all weights and biases are restored in internal memory of FPGA (1024 Clock Cycle), FPGA starts to compute.

The first stage of pipleline is keep the value of inputs and output of previous step. The second stage of pipeline following computations are done:

$(W_{hf} * h_{t\text{-}1})$ $(W_{hi} * h_{t\text{-}1})$ $(W_{ho} * h_{t\text{-}1})$ $(W_{hc} * h_{t\text{-}1})$

$(W_{xf} * h_{t\text{-}1})$ $(W_{xi} * h_{t\text{-}1})$ $(W_{xo} * h_{t\text{-}1})$ $(W_{xc} * h_{t\text{-}1})$

Third stage of pipleline :

$(W_{hf} * h_{t\text{-}1}) + (W_{xf} * h_{t\text{-}1})$

$(W_{hi} * h_{t\text{-}1}) + (W_{xi} * h_{t\text{-}1})$
$(W_{ho} * h_{t\text{-}1}) + (W_{xo} * h_{t\text{-}1})$
$(W_{hc} * h_{t\text{-}1}) + (W_{xc} * h_{t\text{-}1})$
The forth stage of pipeline is:
$((W_{hf} * h_{t\text{-}1}) + (W_{xf} * h_{t\text{-}1}) + b_f)$
$((W_{hi} * h_{t\text{-}1}) + (W_{xi} * h_{t\text{-}1}) + b_i)$
$((W_{ho} * h_{t\text{-}1}) + (W_{xo} * h_{t\text{-}1}) + b_o)$
$((W_{hc} * h_{t\text{-}1}) + (W_{xc} * h_{t\text{-}1}) + b_c)$
the fifth of pipeline is:
$\sigma((W_{hf} * h_{t\text{-}1}) + (W_{xf} * h_{t\text{-}1}) + b_f)$
$\sigma((W_{hi} * h_{t\text{-}1}) + (W_{xi} * h_{t\text{-}1}) + b_i)$
$\sigma((W_{ho} * h_{t\text{-}1}) + (W_{xo} * h_{t\text{-}1}) + b_o)$
$\tanh((W_{hc} * h_{t\text{-}1}) + (W_{xc} * h_{t\text{-}1}) + b_c)$
The sixth stage of pipeline is:
$f_t * C_{t\text{-}1} + i_t * C'_t$
The seventh stage or last stage is:
$o_t * \tanh(C_t)$

After restoring the values of weight and biases from RAMs to FPGA, It takes 7 clock cycles to fill the pipeline and actually after 7 clock cycle the first output C_t and H_t have been computed.

We can calculate execution time by counting the clock cycles like the following:

Weight and biases transferring takes 1024 clock cycle Clock stability usually takes 20 to 30 clock cycles. Computation for 196 sample takes (196+7)-1 clock cycle because the first 7 samples fill the pipeline. It can be formulated generally as:

$$(H * H) + ResetStabilityTime + (T - S) + 1) \tag{7}$$

where H is number of hidden layers, T is number of the testing sample and S is number of the pipeline stage. So in this model Total clock cycle is:

$$(32x32) + 25 + (196 + 7) - 1 = 1251 \tag{8}$$

The clock cycle depends on oscillator connected to FPGA and Microblaze. Clock frequency is 100 MHz so the clock cycle is 10 ns. Result of the first sample is computed within 1056 clock cycle, after this point computation of every sample takes 1 clock cycle. All in all the whole test phase for 196 samples takes 12.5 μs.

5 Results

In terms of accuracy and loss of the training model, the results show that using 2 LSTM has better performance than using only 1 LSTM. This happens because using 2 LSTM allows the model to retain memory better and also have more complex model without having to suffer overfitting the dataset.

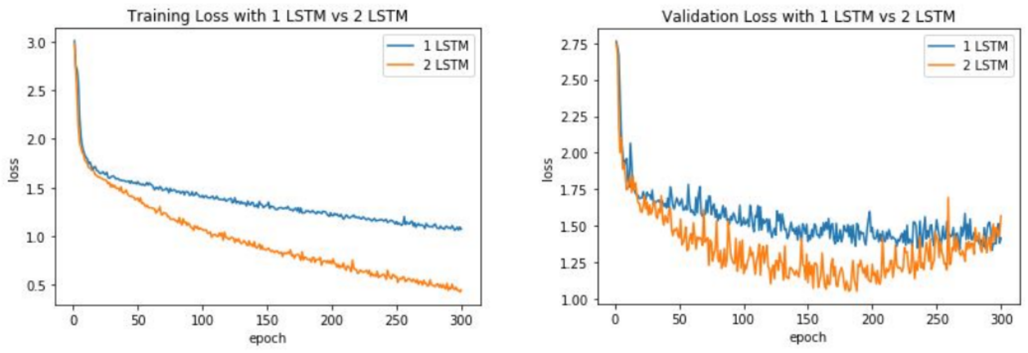


Figure 22: Training and Validation Loss graph

From the loss graph, it can be inferred that the performance of 2-LSTM model is much better than 1-LSTM model. The loss of 1-LSTM model converges to around 1.2, whereas the loss of 2-LSTM model can reach slightly lower than 0.5. The evidence can also be seen in the validation loss graph where 2-LSTM model can outperform 1-LSTM model.

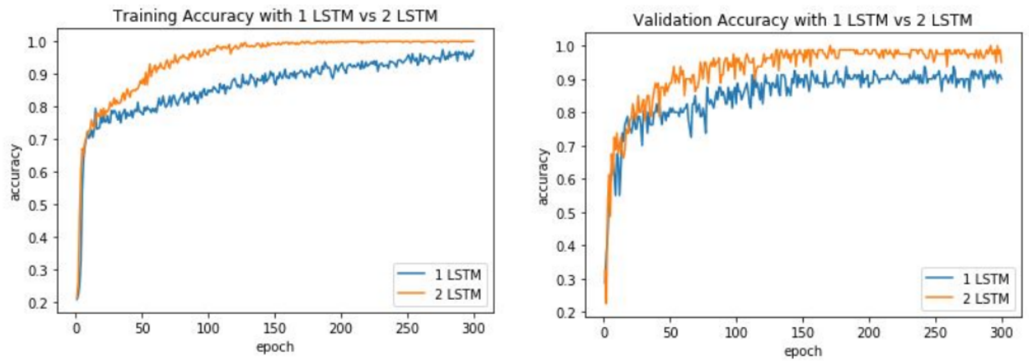
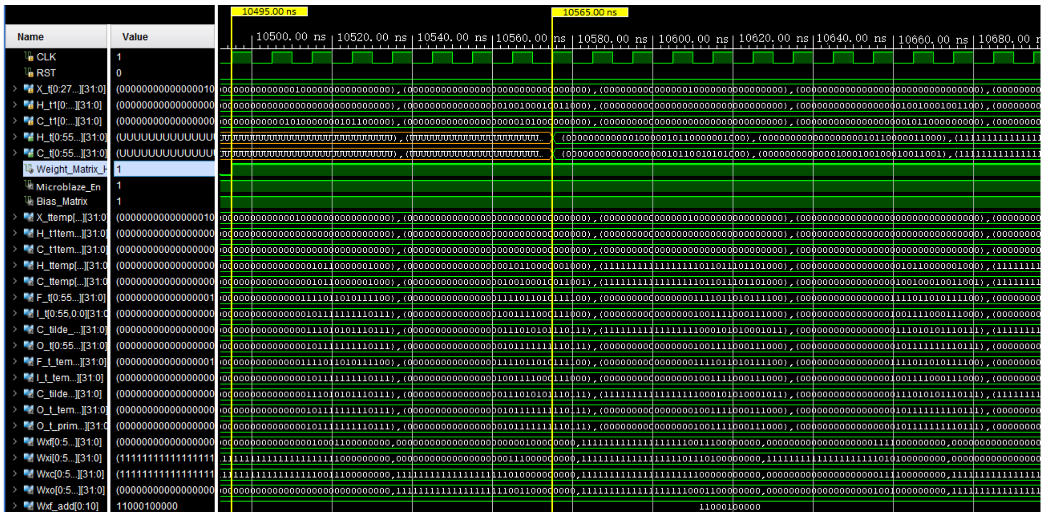


Figure 23: Training and Validation accuracy graph

In the accuracy graph, it can be seen that the performance of 2-LSTM model outperforms 1-LSTM model in both training and validation dataset. In the training dataset, the model with 2-LSTM can reach the accuracy of 98%, while the model with 1-LSTM can reach only 90%. Whereas in the validation dataset, the model with 2-LSTM can reach up to 96% of accuracy, while the model with 1-LSTM can achieve slightly below 90% of accuracy.

Figure 24 shows the experimental result using Vivado 2018.2 Simulation Environment.



Name	Value
CLK	1
RST	0
X_t[0:27...][31:0]	(00000000000000010
H_t1[0:...][31:0]	(00000000000000000
C_t1[0:...][31:0]	(00000000000000000
H_t[0:55...][31:0]	(UUUUUUUUUUUUUU
C_t[0:55...][31:0]	(UUUUUUUUUUUUUU
Weight_Matrix_H	1
Microblaze_En	1
Bias_Matrix	1
X_ttemp[...][31:0]	(00000000000000010
H_t1tem...][31:0]	(00000000000000000
C_t1tem...][31:0]	(00000000000000000
H_ttemp[...][31:0	(00000000000000000
C_ttemp[...][31:0	(00000000000000000
F_t[0:55...][31:0]	(00000000000000001
I_t[0:55,0:0][31:0	(00000000000000000
C_tilde_...][31:0]	(00000000000000000
O_t[0:55...][31:0]	(00000000000000000
F_t_tem...][31:0]	(00000000000000001
I_t_tem...][31:0]	(00000000000000000
C_tilde...][31:0]	(00000000000000000
O_t_tem...][31:0]	(00000000000000000
O_t_prim...][31:0	(00000000000000000
Wxf[0:5...][31:0]	(00000000000000000
Wxi[0:5...][31:0]	(11111111111111111
Wxc[0:5...][31:0]	(11111111111111111
Wxo[0:5...][31:0]	(00000000000000000
Wxf_add[0:10]	11000100000

Figure 24: Simulation Result

We also tried implementing the model on GPU as well as on CPU. The GPU used here is NVIDIA 940MX with clock cycle of 954 - 993 MHz, whereas the CPU specification is Intel Core i5 with clock cycle of 2.9 GHz. The results show that GPU took 88 μs to make an inference, while CPU needed 1019 μs to infer from the testing dataset.

The model was also tested on Google Colaboratory GPU to check its performance. The configuration of GPU was 1xTesla K80 having 2496 CUDA cores , 12GB GDDR5 VRAM whereas CPU was a single core hyper threaded Xeon Processors 2.3GHz i.e(1 core, 2 threads). The RAM was 12.6GB and Disk space available was 320GB. Figure 25 shows comparison of execution time of Artix 7 with other hardware platforms.

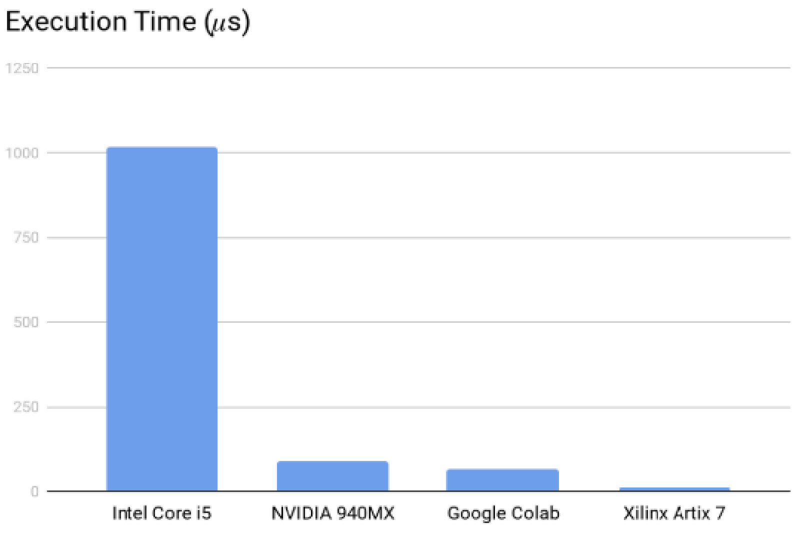


Figure 25: Execution time comparison

The relationship between power and frequency is inversely proportional to each other. On the other words, whenever frequency has been increased, power consumption is increased. Frequency and power supply voltage are two reliable parameters to estimate power consumption. Figure 26 shows frequency comparison of different platforms.

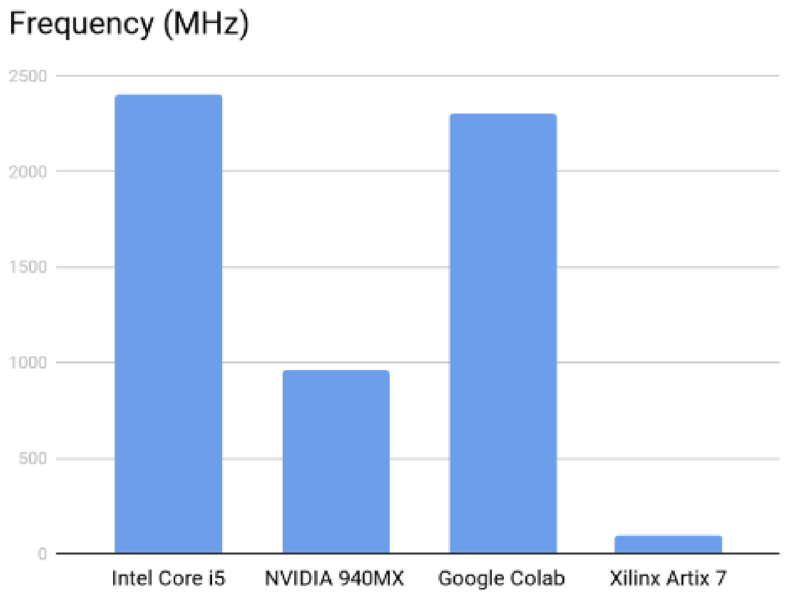


Figure 26: Frequency comparison

6 Conclusion

Recurrent Neural Networks (RNN) have gained popularity recently in many applications such as time series forecasting, Long Short Term Memory is one of the famous RNN architecture. This research has focused on pedestrian motion prediction which is a part of Autonomous Driving applications. We have considered sequence of frames which is corresponding to movements as a time series problem because based on our study the sequence of pedestrian motion has almost a unique pattern in every 25 frames so the movements can be classified in 25 classes and can be predicted using LSTM architecture. LSTM is able to predict the next frame of pedestrian motion. A model including one LSTM has been trained and tested with 90 % accuracy. Furthermore, Autonomous Driver must make a decision quickly and accurately. In fact, Autonomous driver is a real time application which time is a crucial issue. We have introduced FPGA as an hardware platform which is more efficient regarding time and power compared to other hardware platform such as CPU and GPU. We have implemented LSTM on TE0712 which is an evaluation board consisting Artix 7 FPGA family. Our evaluation of implemented LSTM using FPGA shows that it can offer superior performance regarding time and power consumption. All experimental results show the capability of FPGAs to reduce computation time in test phase compared to other hardware platforms such as CPUs and GPUs. All in all, Autonomous driving application must be implemented on embedded systems to be released as a real production, FPGAs are the most populated hardware platform for embedded systems.

7 Future Work

In the future work there are several directions. First of all, we can enrich our dataset with different pedestrian motions because in this project only the people with general walking cycle has been considered. This idea can be implemented on more powerful FPGA such as Aria 10 provided by Intel or Ultrascale+ MPSoC provided by Xilinx to gain more time efficiency due to high frequency supported by Aria 10 or Ultrascale+ MPSoC. The power consumption can be measured precisely using the circuit analysis tools, In this project we have estimated the power consumption using the clock frequency which is a considerable parameter in power consumption but it is not as accurate as measured power consumption. Gated Recurrent Unit (GRU) as a well known RNN architecture can be implemented and tested with same perspective and idea to gain more time and power efficiency.

References

[1] Nurvitadhi, Eriko, et al. *Can FPGAs beat GPUs in accelerating next-generation deep neural networks?*. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017.

[2] Cathal Murphy and Yao Fu, *Xilinx All Programmable Devices: A Superior Platform for Compute-Intensive Systems.*

[3] S. B. Stancliff, J. L. Laine, and M. C. Nechyba. `\Learning to fly: Design and construction of an autonomous airplane".` in 1999 Florida Conference on Recent Advances in Robotics, 1999.

[4] V. Nadkarni, *The Use of Reinforcement Learning in Autonomous Driving for Vehicular Path Planning.* Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017. 2017. [Online]. Available:
`https://www.slideshare.net/VisteonCorporation/the-use-ofreinforcement-learnin`
`[Accessed: 09-Mar2018]`

[5] S. Kardell and M. Kuosku, *Autonomous vehicle control via deep reinforcement learning*, Chalmers university of technology, 2017.

[6] *Robert Bosch Car Multimedia.* 2018. [Online]. Available:
`https://www.bosch.de/en/productsand-services/. [Accessed: 10-Mar-2018].`

[7] *The path to autonomous driving*, 2018. [Online]. Available:
`https://www.bmw.com/en/automotive-life/autonomous-driving.html. [Accessed: 10-Mar-2018].`

[8] *Looking future. Ford will have a fully autonomous vehicle in operation by 2021.* 2018. [Online]. Available:
`https://corporate.ford.com/innovation/autonomous-2021.html. [Accessed: 10-Mar-2018].`

[9] *WAYMO.* 2018. [Online]. Available:
`https://www.google.com/selfdrivingcar/. [Accessed: 10-Mar-2018].`

[10] S. O'Kane, *Former Google self-driving wiz will help Volkswagen and Hyundai build fully autonomous cars* .2018. [Online]. Available:

https://www.theverge.com/2018/1/4/16846526/aurora-chris-urmson-volkswagen-hyu
[Accessed: 10-Mar-2018]

[11] Nurvitadhi, Eriko, et al. *Can FPGAs beat GPUs in accelerating next-generation deep neural networks?* . Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017.

[12] C. Murphy and Y. Fu, *Xilinx All Programmable Devices: A Superior Platform for Compute-Intensive Systems*

[13] Ovtcharov, Kalin, et al. *Accelerating deep convolutional neural networks using specialized hardware* ,Microsoft Research Whitepaper 2.11 (2015).

[14] Cong, Jason, et al., *Understanding Performance Differences of FPGAs and GPUs* ,Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2018.

[15] https://www.aldec.com/en/company/blog/167--fpgas-vs-gpus-for-machine-learnin
-applications-which-one-is-better [Accessed: 20-Mar-2019]

[16] https://www.techbriefs.com/component/content/article/tb/
supplements/pit/features/29739[Accessed: 20-Mar-2019]

[17] Y. Guan, Z. Yuan, G. Sun, and J. Cong, *FPGA-based Accelerator for Long Short-Term Memory Recurrent Neural Networks* ,2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), Chiba, 2017, pp. 629-634.

[18] A. Chang, B. Martini, and E. Culurciello. *Recurrent Neural Networks Hardware Implementation on FPGA* ,arXiv preprint arXiv:1511.05552, 2015.

[19] http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[Accessed: 20-Mar-2019]