ECE 8893 Final report: LSTM Inference Acceleration for Stock Price Prediction

Yuyan Oscar Gao, Alvin Li, Muhamaiti Yesibao

I. Introduction

LSTM inference stage can be time and resource-consuming, so we propose accelerating it with an FPGA to optimize algorithms for stock price prediction. We propose taking an existing LSTM model and accelerating it using the techniques introduced in Dr. Callie Hao's "ECE8893 - FPGA Acceleration" course at the Georgia Institute of Technology and from online sources.

II. Problem Description

Recurrent neural networks (RNN) are neural networks that excel at dealing with timing-dependent sequential data. RNN outperforms other types of neural networks in applications where the data is sequential, such as natural language processing, text generation, and stock price prediction.

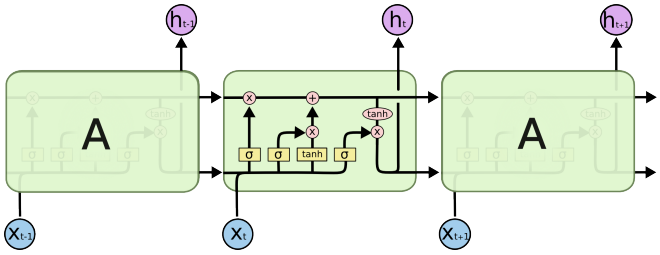

Fig. 1. An example of LSTM unit's architecture [1].

An example of LSTM architecture is shown in Fig 1. Each LSTM processing unit has two inputs: the cell state and the previous units' output, producing its output while updating the cell states. It is evident that with a trained model, the inference stage is time and resource-consuming, especially when dealing with a large data set in time-sensitive applications such as stock price prediction. In stock price prediction, thousands of stock prices are processed, and results are required in real-time to make buy-or-sell decisions. Therefore, we state that accelerating LSTM inference in stock price prediction is an important issue to solve.

III. Accomplished Work

The team's work is divided into two major tasks: 1)The creation and training of the LSTM model, and translation of the inference function to a synthesizable C++ code. 2)The acceleration of the inference function using what we have learned in Dr. Hao's ECE-8893 course. Specific steps are listed below.

A. Creation of model and translation of inference function

1) Model selection: For model selection, the team systematically reviewed and acquired an LSTM model tailored for stock price prediction. The model is developed by building on an existing open-source LSTM model and modifying it for the purpose of our project. Specifically, the team has tried two specific LSTM models: a Keras model and a PyTorch model.

- The Keras model [2] uses a sequential network, and it has three layers: a LSTM layer, a dense layer, and an activation layer.
- The PyTorch model [3] has two layers: a LSTM layer and a fully connected layer, which contains a dense function and an activation function. Two model has similar accuracy and the weights when testing the performance. However, the PyTorch is more customizable for the team to change the parameter to better fit the team's data. Therefore, the team decided to utilize the PyTorch model. In addition, the team added an addition of bias weights to increase the accuracy with relatively small amount of extra weights (DELETE THIS SENTENCE). Therefore, the final model we created has a LSTM layer and a fully connected layer for the output of the LSTM neurons to condense to a single value and better trailer the weights.

B. Training of the model

Based on the actual implementation and the testing of the model. The team discovered that there is a trade-off between complexity, which corresponds to the accuracy and the size of the model, and the performance, which corresponds to the latency and the feasibility of on-board acceleration. Therefore, to best achieve the team's purpose of the project. The team determined our structure of the model to be:

1) Input shape of (1, 59). The dimensionality of the input is 1 and the sequence length of the input is 59.
2) Hidden dimension, which is the number of LSTM units, of 5.
3) Number of LSTM layer of 1 with an additional set of bias weights.
4) Number of fully connected layer of 1 with both dense function and activation function.
5) Output shape of (1,1). A single value as the prediction of the next stock price.

For the dataset, we used IBM stock data from the "Huge Stock Market Dataset" [4].
The team trained this model with the stated dataset, the

accuracy is shown in Fig 2. This trained model was selected as the base of our inference acceleration. Fig 3 shows the performance of our baseline model to be about 1.5ms.

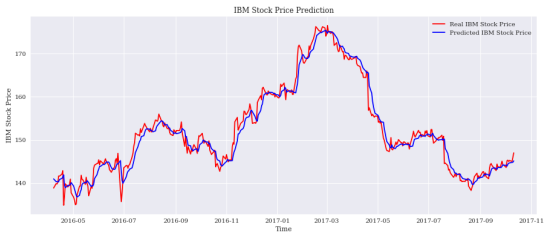

Fig. 2. Result of accuracy of our stock price prediction model.

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
149384	149384	1.494 ms	1.494 ms	149385	149385	no

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2857	-
FIFO	-	-	-	-	-
Instance	14	62	10817	6558	-
Memory	0	-	192	9	0
Multiplexer	-	-	-	2320	-
Register	-	-	4869	-	-
Total	14	62	15878	11744	0
Available	280	220	106400	53200	0
Utilization (%)	5	28	14	22	0

Fig. 3. Latency and utilization report of our baseline model

C. Acceleration of the inference function

After training the model, the team succeeded in translating the python-written model into Golden C++ inference function from scratch using Object Oriented Programming. In addition, the team successfully transformed code to make it synthesizable on the FPGA board.

For the acceleration of the inference function, we discovered that there are two major area of improvements: Data input and the Activation. [ciation]. To accelerate our inference function, we tried these different techniques in HLS:

- The first method is the Array Partitioning, Unrolling and Pipelining as we learned from the lecture.
- Dataflow Optimization, as we also learned from the lecture.
- Quantization, which was inspired by this[5].
- Function Approximation, which means that we tried using "Fast" tanh and exp functions to optimize the latency of activation functions.

After lots of experiments, we found that the Dataflow optimization is unsuccessful, the best result we got is no latency increases. We believe this is due to the nature of the LSTM inference of stock price prediction. We are simply outputting a single value for the result, hence the Dataflow trick cannot work in this scenario. In addition, the function approximation also did not work well. This is because the approximation error accumulated to a significant amount due to many number of calculations during inference, which makes the accuracy drop to an unacceptable level. Fortunately, the rest two methods works well, and we gained significant performance boost. We will discuss this in detail in the next section.

D. Results of acceleration

For the two successful method of acceleration, we are incrementing based on successful attempts. The first successful method is the Array Partitioning, Unrolling and Pipelining. The result of this attempt is shown in Fig 4. Based on this result, we additionally introduced Quantization method, which turned ap_fixed<32,10> operations into ap_fixed<8,2> operations. This attempt significantly reduced the latency and resource usage, since the bottleneck at this moment is the sigmoid and tanh functions. The result of final performance is shown in Fig 5.

E. Deliverable

The team has produced these deliverable:

1) A demonstration of the correctly translated C++ model inference, with test cases presented comparing our inference code to the python code used in the model.
2) Screenshots showing the techniques applied to accelerate the inference and the speedup achieved by using the aforementioned techniques.
3) A demonstration of on-board implementation of the dense layer similar to Lab 3.

IV. Challenges and Gains

During the project period, We met two major challenges

1) During training: Choosing parameters that balance accuracy, performance, and simplicity.
2) Writing inference function in C++: Unpacking weights was the challenging. A side note here is that We will document this on Github for the community's future reference.

Moreover, we gained experience for an acceleration problem. First, we understood the importance of choosing the right framework. In our case, choosing the PyTorch model and determining the structure and parameter of our model. In addition, we learned to approach acceleration problems differently. For example, the Dataflow Optimization did not work in our case.

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
12005	12005	0.120 ms	0.120 ms	12006	12006	no

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	26	-
FIFO	-	-	-	-	-
Instance	39	172	55612	36986	-
Memory	1	-	14336	99	0
Multiplexer	-	-	-	3152	-
Register	-	-	944	-	-
Total	40	172	70892	40263	0
Available	280	220	106400	53200	0
Utilization (%)	14	78	66	75	0

Fig. 4. Latency and utilization report with Array Partitioning, Unrolling and Pipelining

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
4866	4866	48.660 us	48.660 us	4867	4867	no

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	26	-
FIFO	-	-	-	-	-
Instance	2	29	7327	7572	-
Memory	0	-	3600	43	0
Multiplexer	-	-	-	2876	-
Register	-	-	788	-	-
Total	2	29	11715	10517	0
Available	280	220	106400	53200	0
Utilization (%)	~0	13	11	19	0

Fig. 5. Latency and utilization report of final inference function with Array Partitioning, Unrolling, Pipelining and Quantization

REFERENCES

[1] "Understanding LSTM Networks." https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed Mar. 16, 2022).

[2] " 深層学習の l 、上がる株みつかる l 何時もの話っ！. https://memo.soarcloud.com/深層学習から上がる株みつかる

[3] "Predicting Stock Price using LSTM model, PyTorch." https://kaggle.com/taronzakaryan/predicting-stock-price-using-lstm-model-pytorch (accessed May 03, 2022).

[4] "Huge Stock Market Dataset." https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs (accessed May 03, 2022).

[5] T. Mealey and T. M. Taha, "Accelerating Inference In Long Short-Term Memory Neural Networks," in NAECON 2018 - IEEE National Aerospace and Electronics Conference, Jul. 2018, pp. 382–390. doi: 10.1109/NAECON.2018.8556674.