LSTM Inference Acceleration for Stock Price Prediction

Yuyan Oscar Gao, Alvin Li, Muhamaiti Yesibao

Recap of the Problem Statement

A Recurrent Neural Network is a neural network that can recognize patterns in sequential data. Long Short-term memory (LSTM) is a special type.

The inference stage can be very time/resource-consuming.

We aimed accelerate it. Specifically, for stock price prediction.

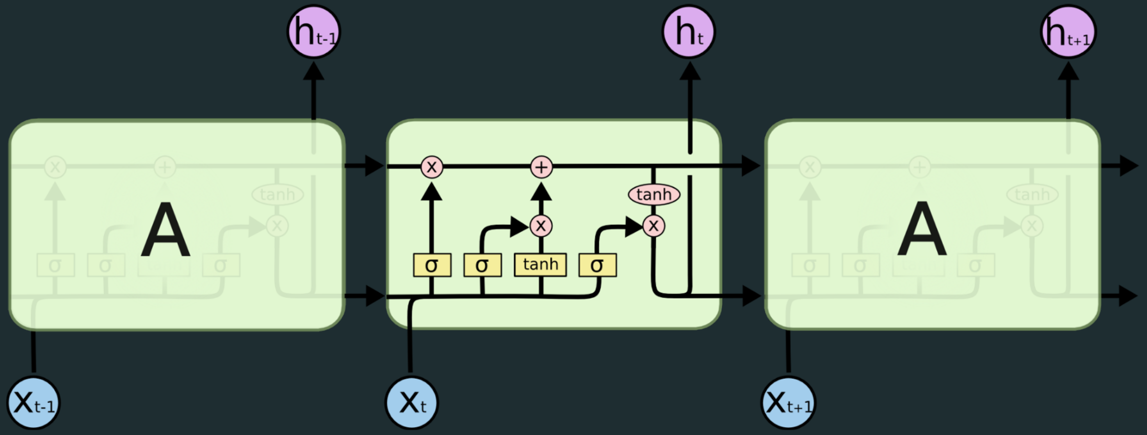


Figure 1. An example of LSTM network [1].

Summary of Work: Model Selection

1. Keras model [2]
    a. Uses a built-in sequential network structure
    b. Three layers: LSTM layer, dense layer, activation layer
2. PyTorch model (Our choice) [3]
    a. Two layers: an LSTM layer and a fully connected layer, which contains a dense function and an activation function

These models share similar accuracy/performance. However, the PyTorch model is more customizable and easier to translate.

Additional difference of our model: an additional set of bias weights

Summary of Work: Training

There is a trade-off between **complexity** (accuracy/size of model) and the **performance** (latency/feasibility of on-board acceleration). To maximize our output, we determined our structure of the model to be:

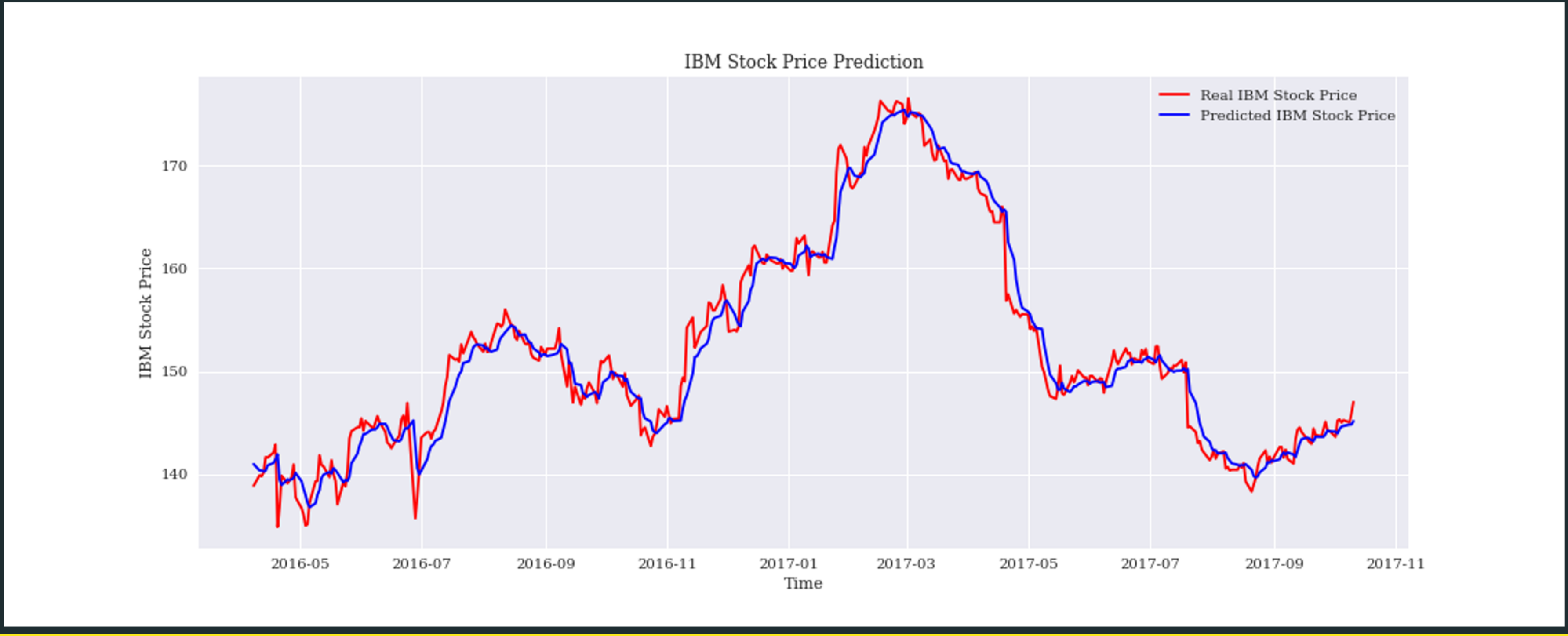
- Input shape: (1, 59), dimension:1, sequence length: 59
- Hidden dimension(number of LSTM units): 5
- Number of LSTM layer: 1
- Output shape: (1,1)

We used the IBM from the “Huge Stock Market Dataset ” [4].

Result shown in the next slide.

Result of Our Training

Test Score: 1.74 RMSE, Time: ~5.8ms per prediction



Summary of Work: Translation and Baseline

Golden C++ inference function from scratch using OOP. (github.com/oscargao98/LSTM_Inference_CPP)

Transformed code to make it synthesizable on our FPGA board.

Baseline model performance ~ 1.5ms

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**

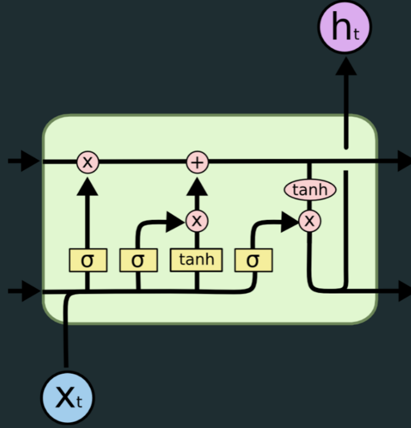
⊞ **Loop**

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

Summary of Work: Acceleration



Results are shown in next slide

Two major areas: Data input, Activation [5].

Techniques Attempted:

- Array Partitioning & Unrolling & Pipelining: Successful!
- Dataflow Optimization: Unsuccessful! – Nature of the inference
- Quantization: Successful! (<3% error)
- Function Approximation: Unsuccessful! – "Fast" tanh and exp functions' accuracy not acceptable

Summary of Work: Acceleration Results

Pipeline only

– 0.120 ms

Pipeline+Quantization

– 0.04864 ms

– 30x speedup from baseline

– 120x from original python

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

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

Challenges and Gains

- We met two major challenges:
  - During training: Choosing parameters that balance accuracy, performance, and simplicity.
  - Writing inference function in C++: Unpacking weights was the challenging. We will document this on Github for the community's future reference.
- We gained experience for a relevant acceleration problem:
  - Choosing the best framework - Pytorch in our case.
  - Approach acceleration problems differently. Ex. - #pragma DATAFLOW not effective.

Reference

1. “Understanding LSTM Networks -- colah’s blog.” https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed May 03, 2022).
2. “深層学習の１、上がる株みつかる | 何時もの話っ！.” https://memo.soarcloud.com/%e6%b7%b1%e5%b1%a4%e5%ad%a6%e7%bf%92%e3%81%8b%e3%82%89%e4%b8%8a%e3%81%8c%e3%82%8b%e6%a0%aa%e3%81%bf%e3%81%a4%e3%81%8b%e3%82%8b/ (accessed May 03, 2022).
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.