CSC14116 - Applied Parallel Programming

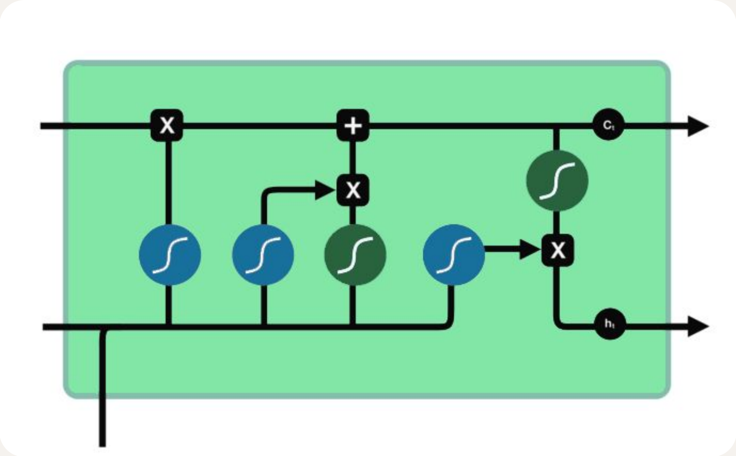
Parallel LSTM Training for Sequence Prediction from Sequential Data

19120454 - Bùi Quang Bảo

01

Introduction

About the project



Introduction

Analyze and parallel the training process of a LSTM model.

➜ Improve training speed + handle larger datasets

Task: Sequence prediction from sequential data (Seq2Seq)

Implement a raw LSTM model using Python with Numpy library, analyze, parallelize and measure the efficiency.

02

Background

About RNN and LSTM

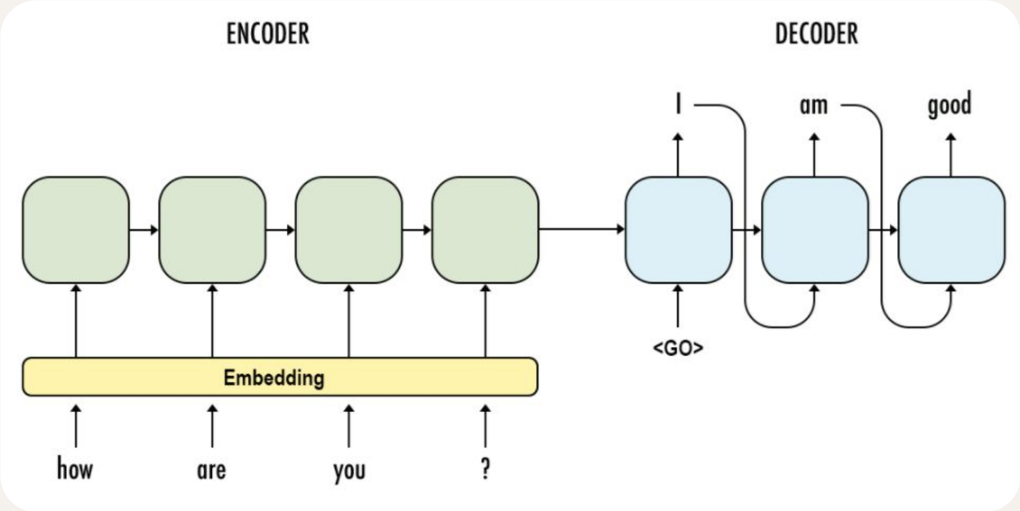
Sequence to Sequence (Seq2Seq)

Input: Sequence (List of tokens)

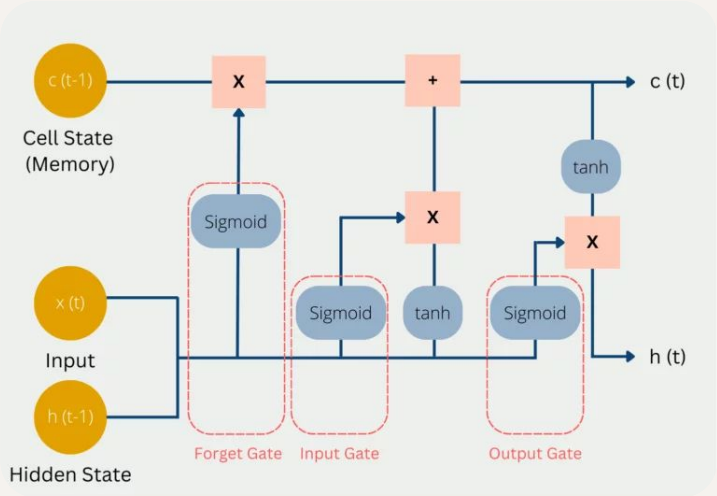
Output: Sequence (List of tokens)

Applications: Chatbots, Machine Translation, Text Summarization, ...

Solutions for Seq2Seq: RNN, CNN, Transformers, ...



Example about Seq2Seq



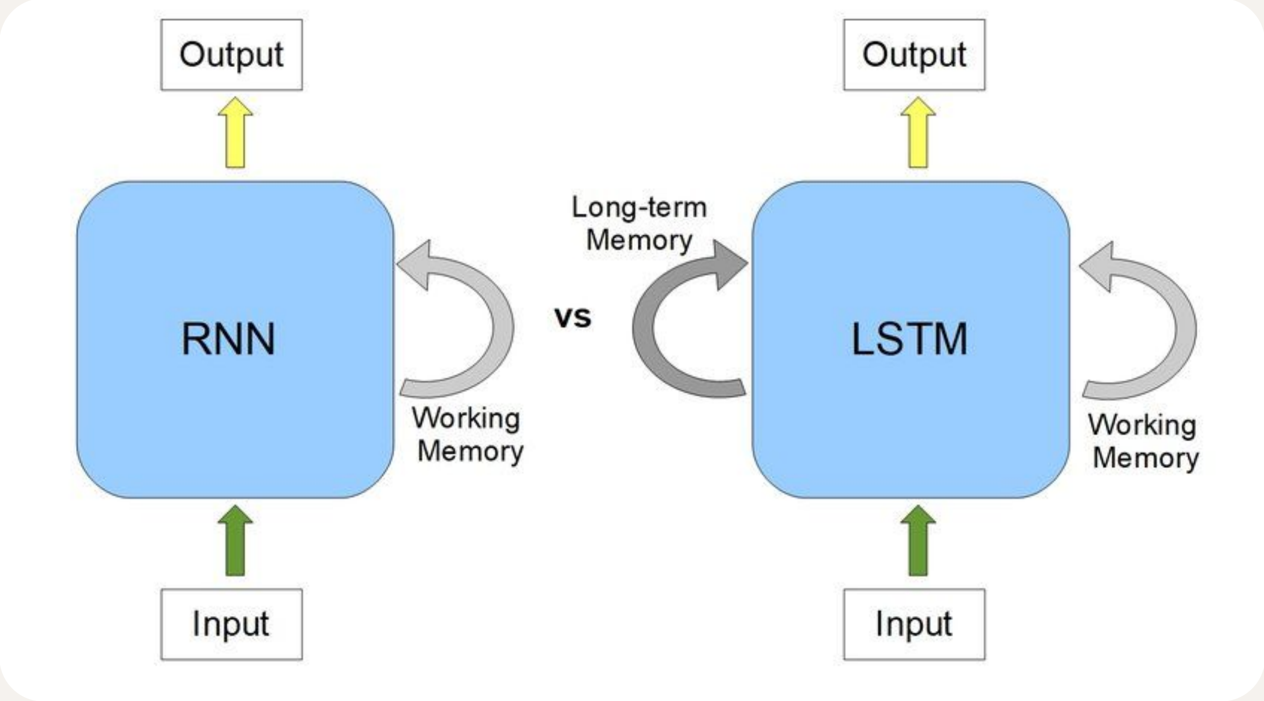
LSTM

- Long Short-Term Memory, is a type of recurrent neural network (RNN).

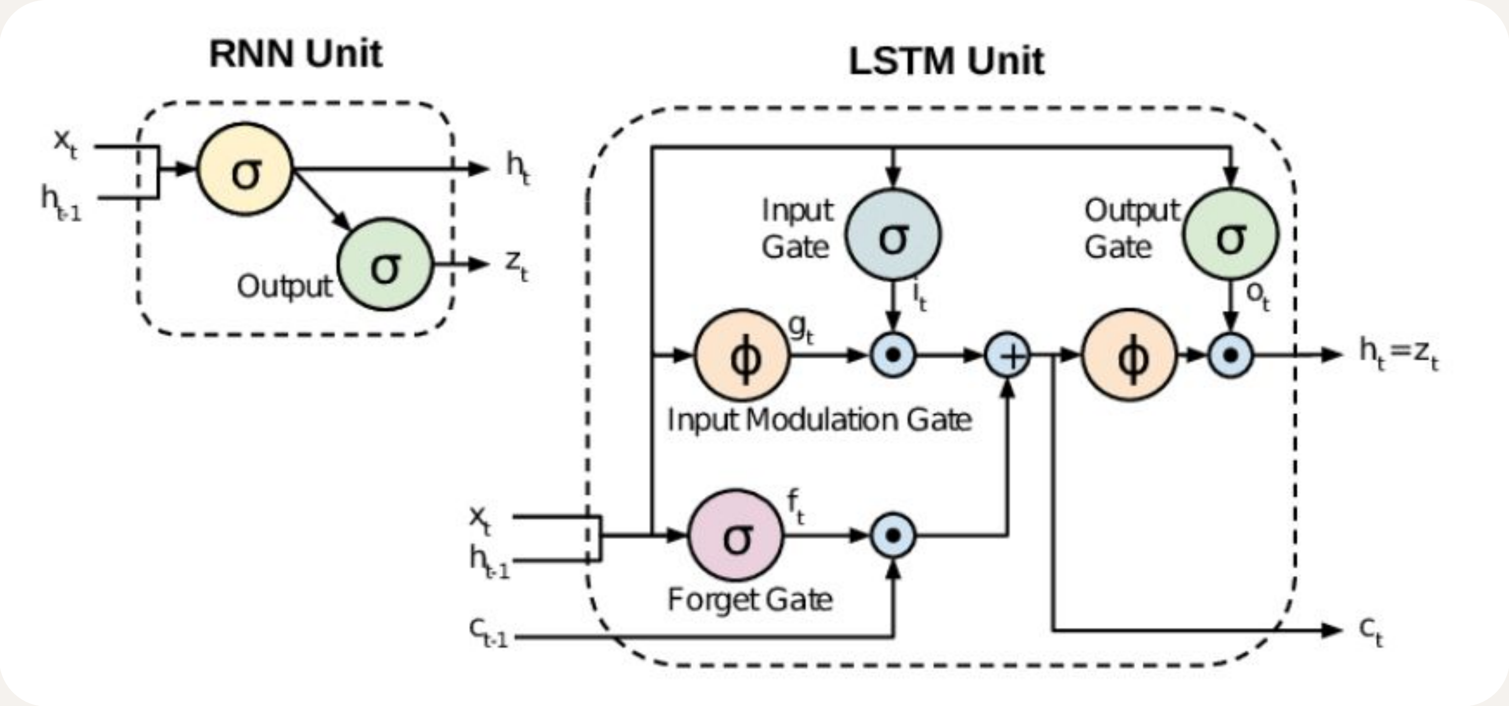
- LSTM networks have the ability to selectively ***remember*** or ***forget*** information over long periods of time.

- The architecture of an LSTM network includes: memory cells, input and output gates, and forget gates,...

RNN and LSTM

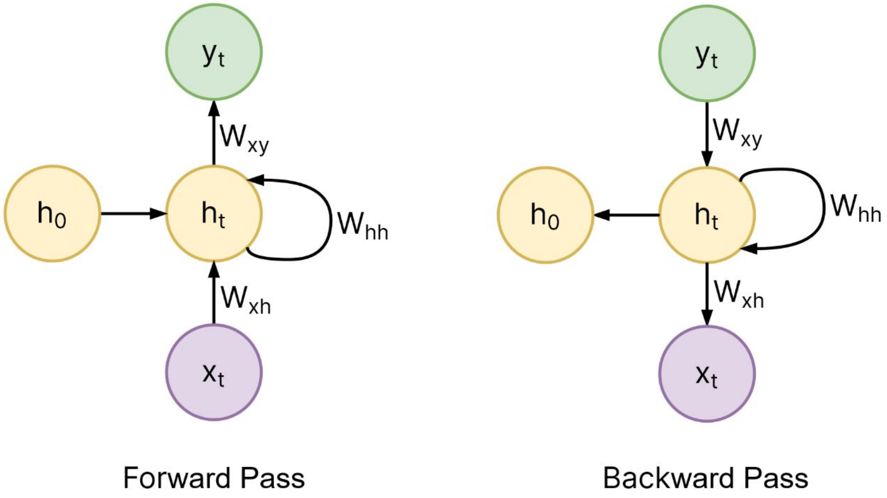

Output
RNN
Working
Memory
Input
vs
Long-term
Memory
Output
LSTM
Working
Memory
Input

RNN and LSTM

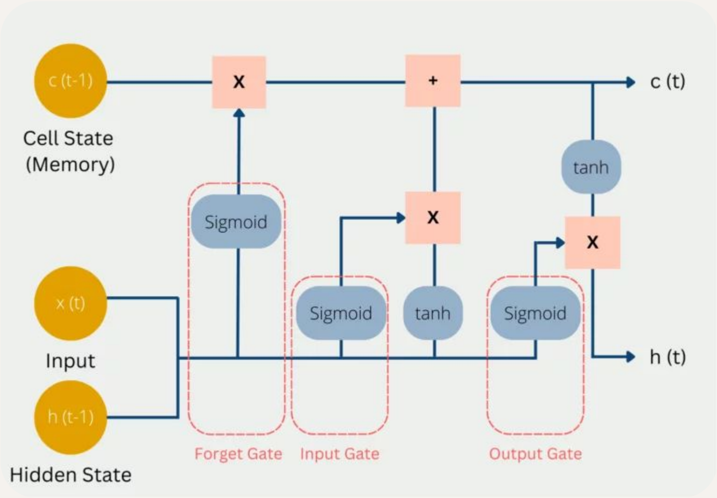


RNN Unit
x_t
h_{t-1}
σ
h_t
σ
z_t
Output
LSTM Unit
Input Gate
σ
Output Gate
σ
i_t
o_t
g_t
ϕ
ϕ
$h_t=z_t$
Input Modulation Gate
x_t
h_{t-1}
σ
f_t
Forget Gate
c_{t-1}
c_t

LSTM Architecture



LSTM Forward & Backward Propagation



Parallel Potential

- The compute-intensive process is the **training phase** of LSTM model.

- Some processes: data encoding, forward pass, backward pass, loss calculating, activation functions calculating, parameters optimizing,... looped for all samples for all epochs.

- The training phase of a LSTM model will benefit from parallelism.

03

Challenges

Challenges for parallelizing LSTM model

Challenges

1. LSTM model training does not contain totally-independent loops.

- Epochs cannot be parallelized, because after each epoch, the parameters must be updated by the optimization functions (Gradient Descent or Adam).

- The process of “looking” at validation data and training data is not totally independent and it is based on how we implement the optimization functions.

2. It’s not easy to demo and visualize in comparison with other models.

04

Resources

Hardware resources

Hardware Resources

Google Colab

Kaggle

05

Goals

Goals and Deliverables

Goals

100%

Plan to achieve

The final parallelized version will run x5 faster.

125%

Hope to achieve 1

The final parallelized version will run x10 faster.

150%

Hope to achieve 2

Training time do not scale up as quickly as the size of the dataset

75%

In case slowly

The final parallelized version will run faster.

06

Schedule

Timeline to do the project

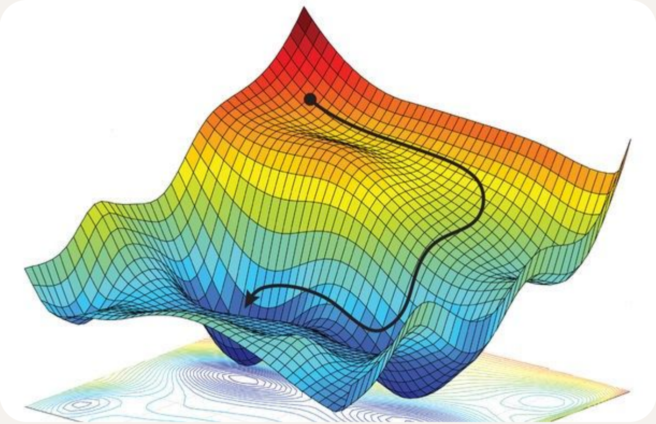
Schedule

Time	Assignee	To do
Phase 1 20/3 - 26/3	19120454 - Bùi Quang Bảo	Understand the LSTM model and analyze what and how to parallel.
Phase 2 27/3 - 9/4	19120454 - Bùi Quang Bảo	Finish the sequential implementation of the LSTM model.
Phase 3 10/4 - 30/4	19120454 - Bùi Quang Bảo	Parallel LSTM model training. Update the sequential implementation if necessary. Present the project at weekly reports or mid-term seminars, if any.
Phase 4 1/5 - 7/5	19120454 - Bùi Quang Bảo	Write the report about the project and the results.
Phase 5 Unknown	19120454 - Bùi Quang Bảo	Demo and present the project at the final seminal.
Phase 6 Unknown	19120454 - Bùi Quang Bảo	Update the project after the final seminar, if required.

07

Implementation

Implementation Strategy

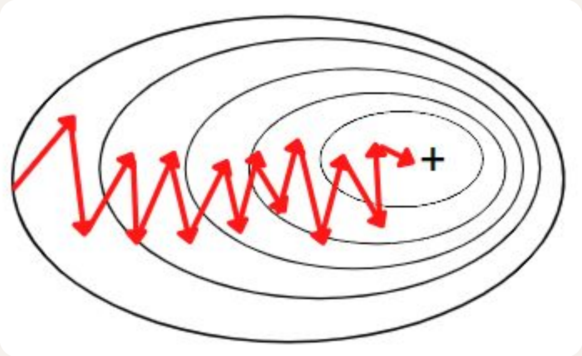


Gradient Descent

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks.

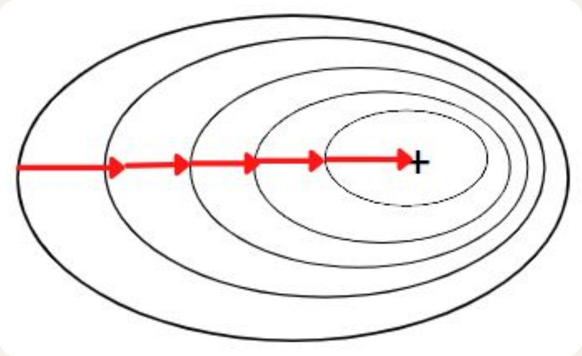
➜ Update LSTM network’s parameters

Optimization Strategy



Stochastic Gradient Descent

Rough gradient
Fast

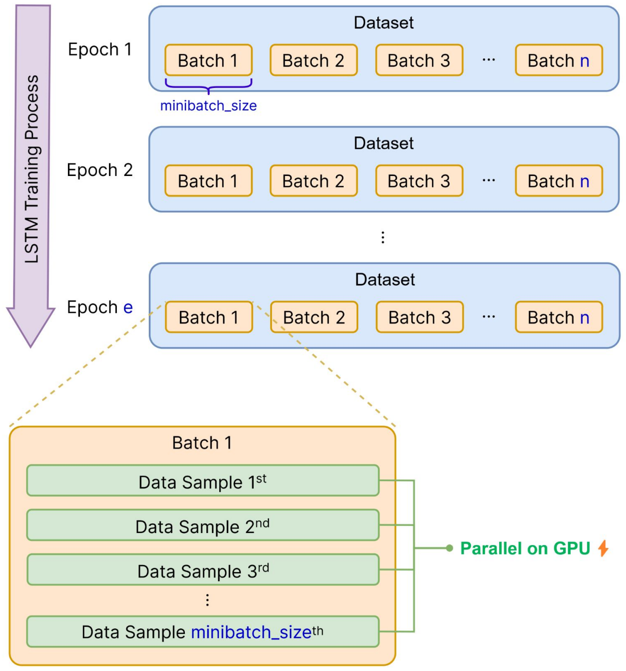


Batch Gradient Descent

Good gradient
Slow

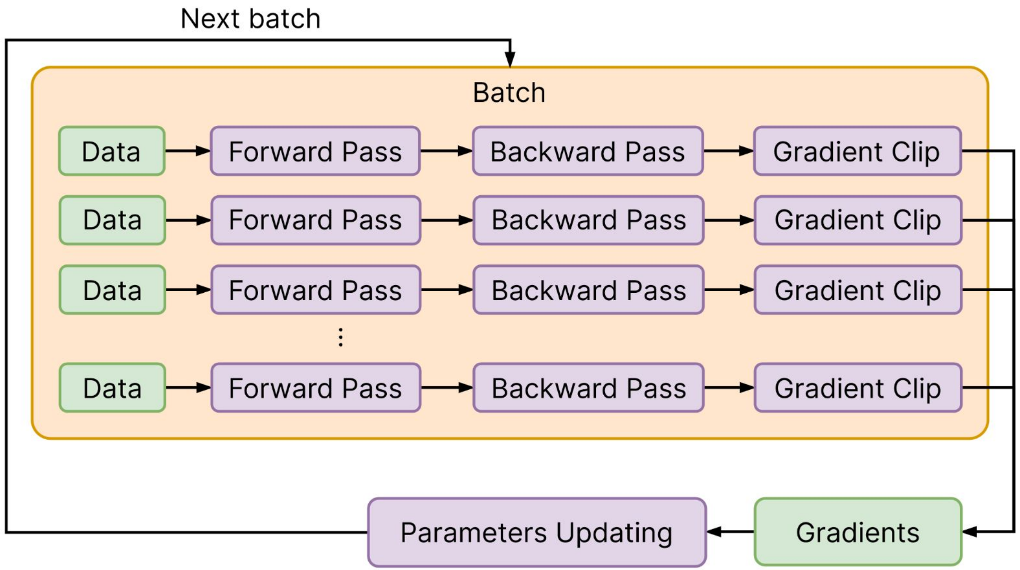
➜ Benefit from parallelism

Parallel Strategy



LSTM Training Parallel Implementation Strategy

Parallel Strategy



LSTM Training Parallel Implementation Strategy

08

Demo

Demo on real dataset

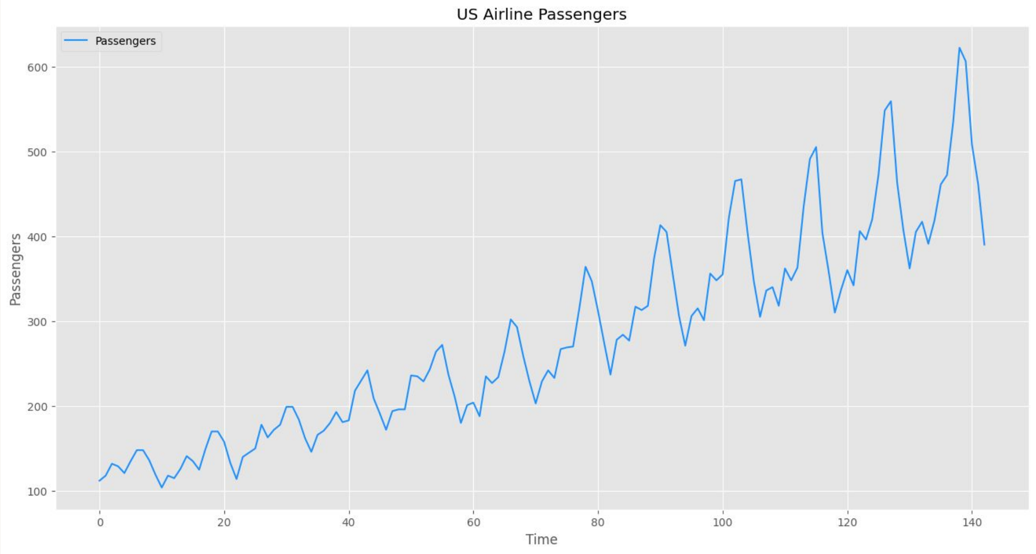
Dataset: US Airline Passengers

This dataset provides monthly totals of a US airline passengers from 1949 to 1960.

Original Source: *Box, G. E. P., Jenkins, G. M. and Reinsel, G. C. (1976) Time Series Analysis, Forecasting and Control. Third Edition. Holden-Day. Series G.*

Task to solve

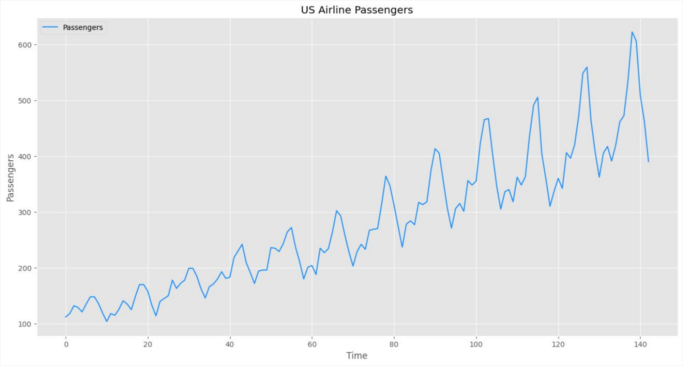
Predict/forecast the number of passengers of a US airline.

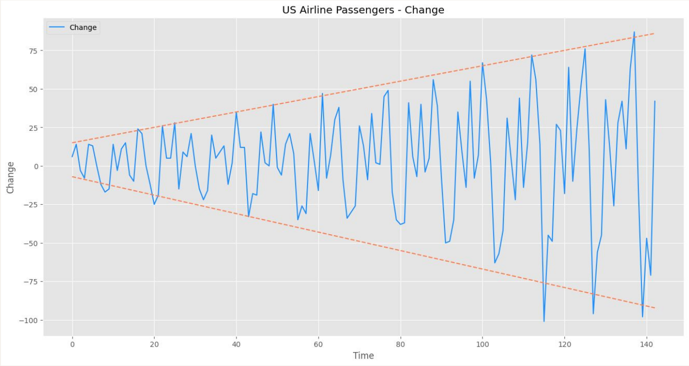


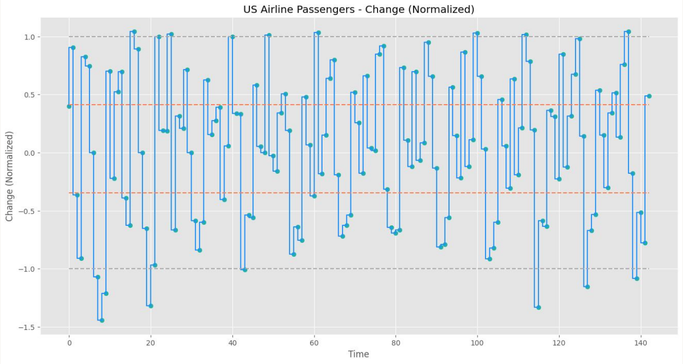
This is a time-series prediction task and totally fit the purpose of this project

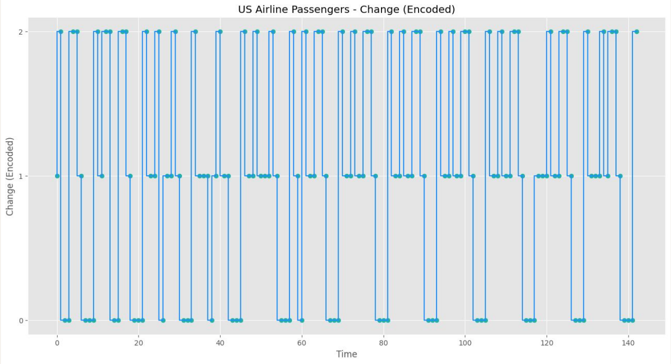
➜ We will build a LSTM model to solve this task.

Preprocessing

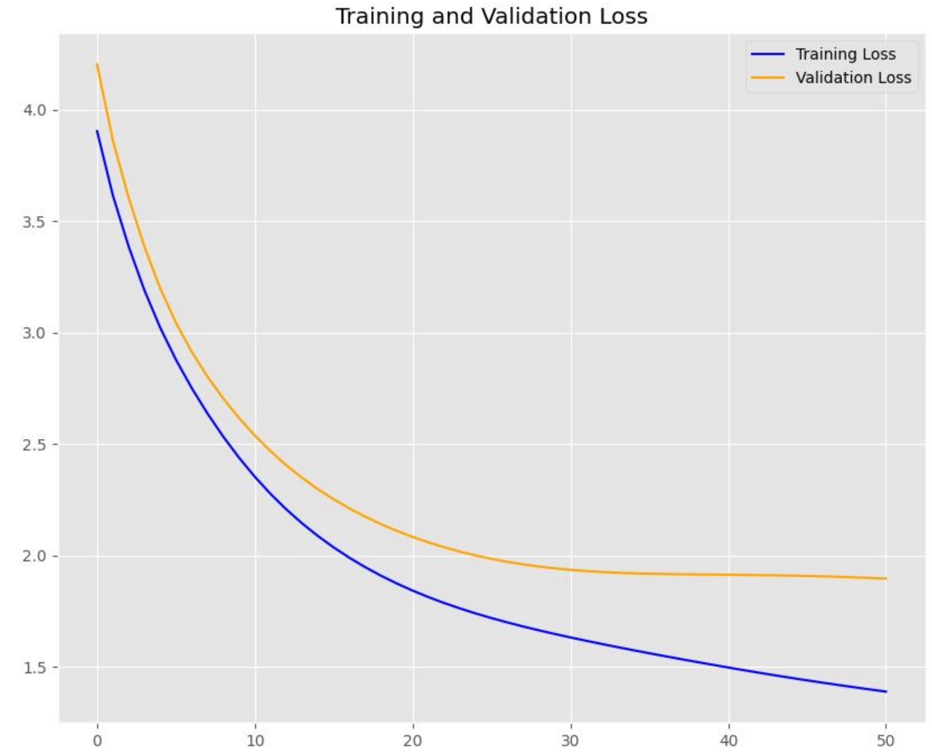






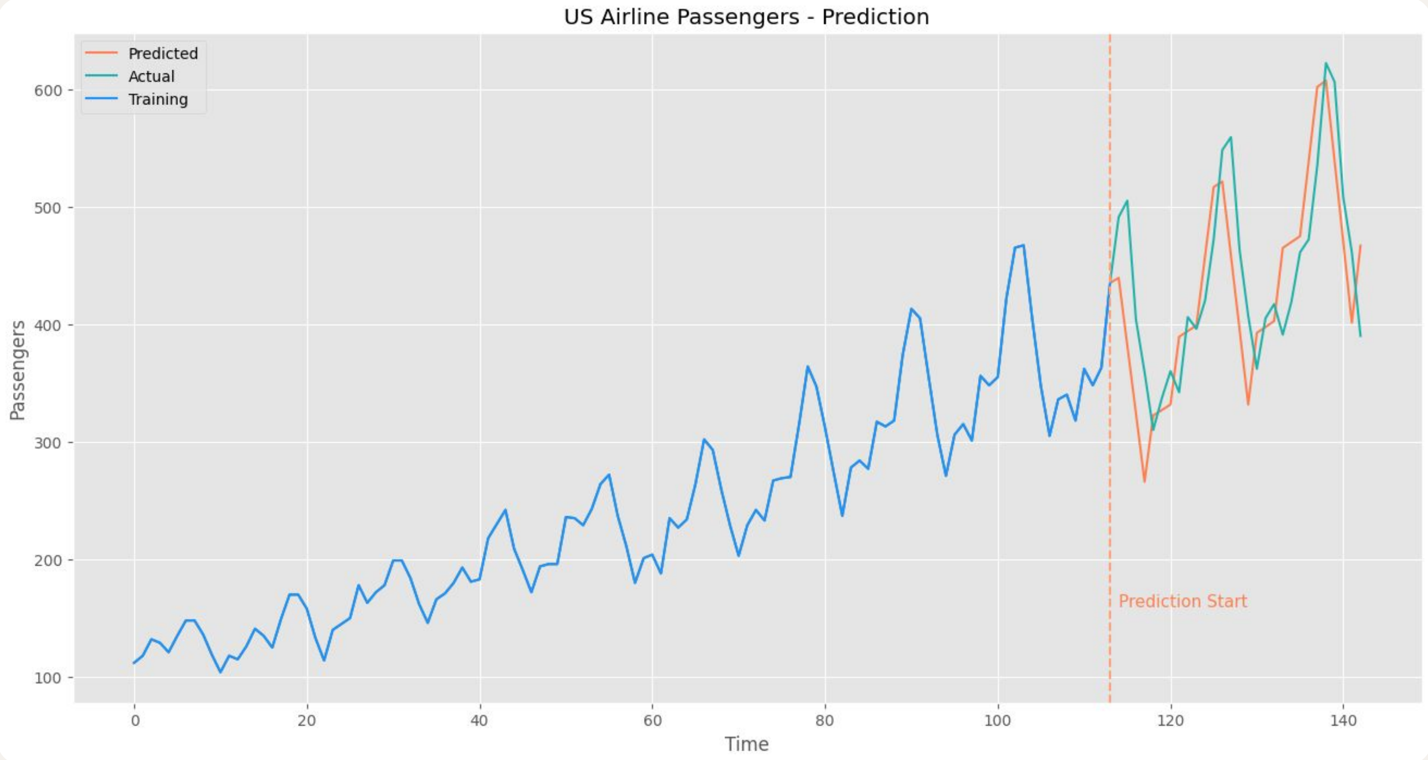


Result



Training and Validation Loss
Training Loss
Validation Loss
4.0
3.5
3.0
2.5
2.0
1.5
0
10
20
30
40
50

Result


US Airline Passengers - Prediction
Predicted
Actual
Training
Prediction Start
Passengers
Time
600
500
400
300
200
100
0
20
40
60
80
100
120
140

09

Versions Comparison

Parallelization Efficiency

Sequential Version

This is sequential implementation of LSTM, all the processes run on CPU.

Sequential Version:

- Each Epoch runs on: CPU
- Each Mini-batch runs on: CPU ❌
- Each Data sample runs on: CPU ❌
  - Thread positioning: No ❌
  - Optimize data transferring: No ❌

Numpy Version

This is sequential implementation of LSTM, use Numpy, all the processes run on CPU.

Numpy Version:

Each Epoch runs on: CPU

Each Mini-batch runs on: CPU ❌

Each Data sample runs on: CPU ❌

Thread positioning: No ❌

Optimize data transferring: No ❌

Parallel Version 1

This is the first parallel implementation of LSTM to run on GPU.

This version is not actually "parallel" yet, it's just a quick convert from sequential version to test the ability to run on GPU using numba.

Parallel V1:

- Each Epoch runs on: CPU
- Each Mini-batch runs on: GPU ✔️
- Each Data sample runs on: GPU ✔️
  - Thread positioning: No ❌
  - Optimize data transferring: No ❌

Parallel Version 2

This is the second parallel implementation of LSTM to run on GPU.

In this version, for each mini-batch we will invoke kernel once, and each data sample in the mini-batch will run on a thread on GPU.

Parallel V2:

Each Epoch runs on: CPU

Each Mini-batch runs on: GPU ✔️

Each Data sample runs on: GPU ✔️

Thread positioning: Yes ✔️

Optimize data transferring: No ❌

Parallel Version 3

This is the third parallel implementation of LSTM to run on GPU.

In this version, for each mini-batch we will invoke kernel once, and each data sample in the mini-batch will run on a thread on GPU. The improvement compared to the version 2 is that in this version, we ***avoid the unnecessary transfer*** for read-only data arrays.

By default, Numba ***automatically*** transfer NumPy arrays to the device, it can only do so conservatively by always transferring device memory back to the host when a kernel finishes. So, we decide to ***manually control*** the transfer behavior for these read-only data arrays:

- Data: *minibatch_set*
- Previous parameters information: *U, V, W, B, b_out*

Parallel Version 3

This is the third parallel implementation of LSTM to run on GPU.

In this version, for each mini-batch we will invoke kernel once, and each data sample in the mini-batch will run on a thread on GPU. The improvement compared to the version 2 is that in this version, we ***avoid the unnecessary transfer*** for read-only data arrays.

Parallel V3:

Each Epoch runs on: CPU

Each Mini-batch runs on: GPU ✔️

Each Data sample runs on: GPU ✔️

Thread positioning: Yes ✔️

Optimize data transferring: Yes ✔️

Versions

All implementation versions

	Mini-batch	Data Sample	Thread positioning	Data transferring optimization
Sequential	CPU	CPU		
Numpy	CPU	CPU		
Parallel V1	CPU+GPU	CPU+GPU	No	No
Parallel V2	CPU+GPU	CPU+GPU	Yes	No
Parallel V3	CPU+GPU	CPU+GPU	Yes	Yes

Results

5 epochs, mini-batch gradient descent

	Wall time	Efficiency (vs. Sequential)	Efficiency (vs. Numpy)	Evaluate
Sequential	4min 53s	*100%*		
Numpy	2min 10s	225%	*100%*	
Parallel V1	1min 20s	366%	162%	
Parallel V2	18.8 s	1558%	691%	
Parallel V3	17.2 s	**1703%**	**756%**	🥇 Best Version

10

Conclusion

Conclusion about the project

Conclusion

We successfully completed the project and reached the goal we stated in the project's proposal (100%)

	Wall time	Efficiency (vs. Sequential)	Efficiency (vs. Numpy)
Sequential	4min 53s	*100%*	
Numpy	2min 10s		*100%*
Parallel V3	17.2 s	**1703%**	**756%**

Thanks for listening!