

# Final Project - Sentiment Analysis

Project due: 2024. 12. 20. 11:59 PM

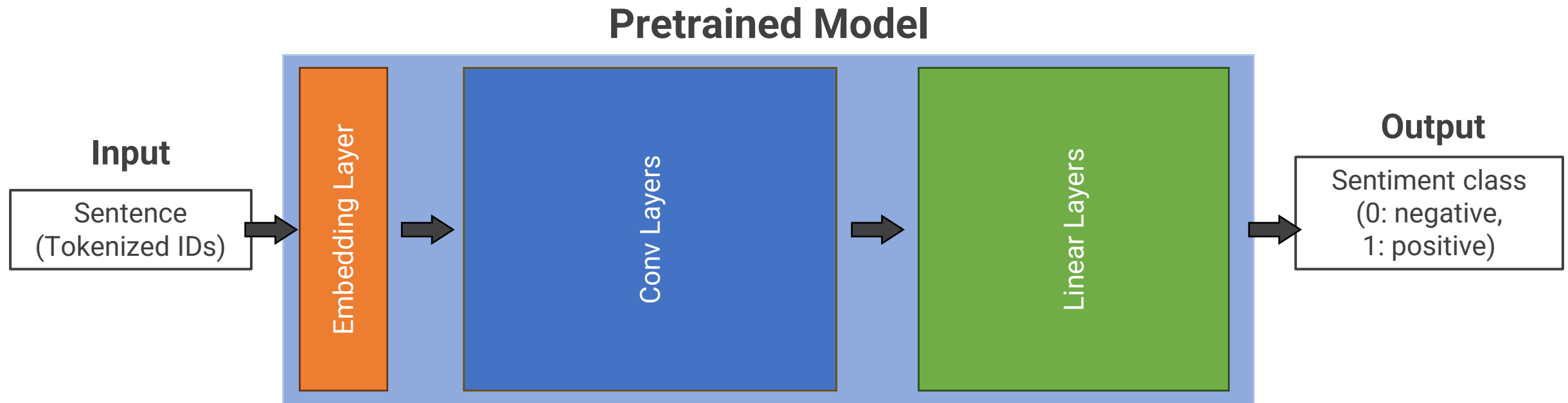
Last updated: 2024. 12. 17. 19:00 PM

# Project Goal

- 순차 코드로 구현되어 있는 딥 러닝 추론 프로그램을 병렬화/최적화
  - 총 4개의 계산 노드를 사용 (NVIDIA RTX TITAN 총 16장)
  - Pthread, OpenMP, MPI, CUDA 사용 가능
  - 외부 라이브러리 사용 불가능

# Target Model

- Sentiment Analysis Model
  - Determining whether a given text sentence a positive or negative sentiment
  - CNN based model

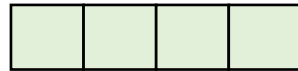


# Background

# Background - Tensor

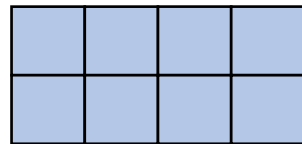
- 딥 러닝에서 데이터를 다루는 단위
  - 뼈대 코드에 구현되어 있는 연산들은 tensor를 입출력으로 가짐
  - 정의: `include/tensor.h`, 구현: `src/tensor.cu`

1D Tensor  
(e.g., Vector)



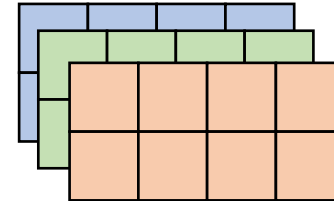
Shape = {4}

2D Tensor  
(e.g., Matrix)



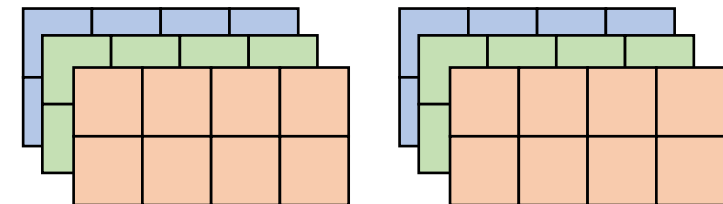
Shape = {2, 4}

3D Tensor  
(e.g., RGB image)



Shape = {3, 2, 4}

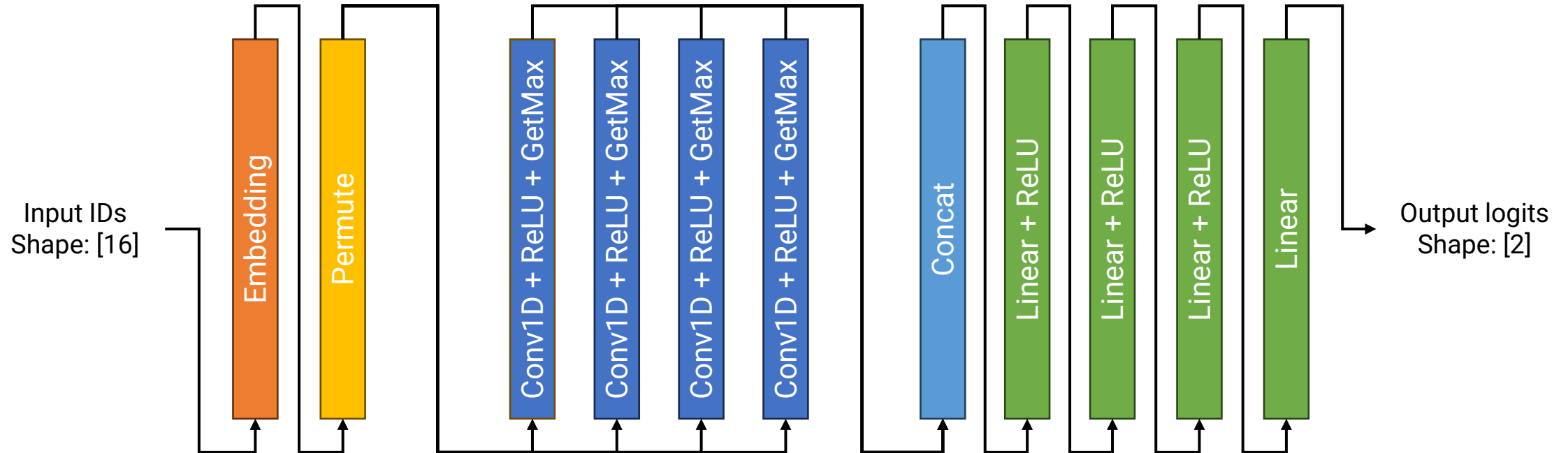
4D Tensor  
(e.g., CNN filter)



Shape = {2, 3, 2, 4}

# Background - Model

- CNN based Sentiment Analysis Model
  - 정의: include/model.h, 구현: src/model.cu



# Background - Layers

- A set of nodes or neurons that processes and transforms input data
  - 정의: include/layer.h, 구현: src/layer.cu
  - 종류
    1. Embedding
    2. Permute
    3. Conv1D
    4. ReLU
    5. GetMax
    6. Concat
    7. Linear

## Background – Embedding Layer

- Lookup table에서 토큰 값을 Index로 해서 Embedding vector를 반환
- 입출력
  - [in1] in: [s]
  - [in2] w: [NUM\_VOCAB, H]
  - [out] out: [s, H]

Index	Hidden Dim				
	0	1	2	3	4
0	1.3	2.4	5.5	2.3	1.4
1	65	34	3.3	4.6	1.4
...					
122	2.5	6.4	2.3	4.4	6.5
123	3.5	3.5	2.4	2.1	7.8
124	6.7	8.6	2.5	4.1	9.4
125	1.2	8.6	6.4	6.7	2.4
...					4.3
21634	3.5	3.5	2.4	2.1	7.8

# of Vocab

123 (Token)

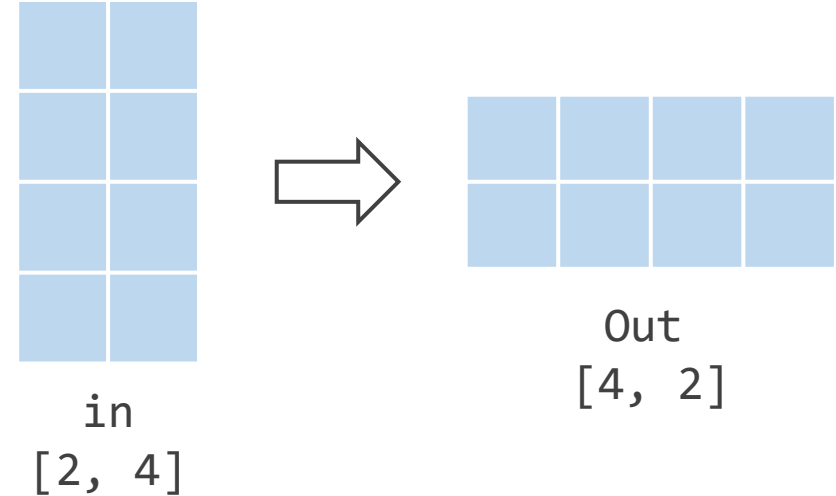
Embedding vector

<Lookup table>



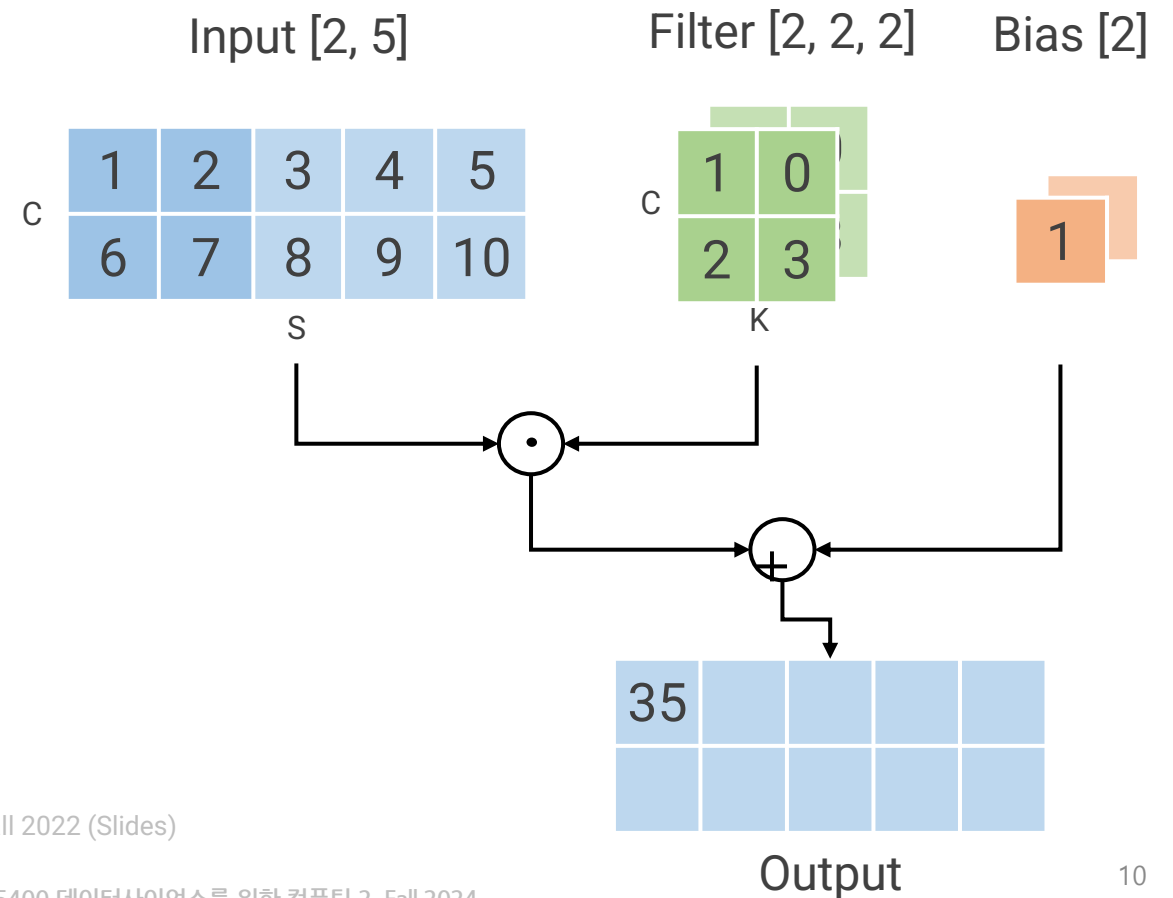
## Background – Permute Layer

- 시퀀스(sequence) 차원과 임베딩(embedding) 차원의 순서를 변환
- 입출력
  - [in]    in:  $[M, N]$
  - [out] out:  $[N, M]$



# Background - Conv1D Layer

- 1차원 입력 데이터를 따라 1차원 filter를 이동하면서 합성곱
- 입출력
  - [in1] in: [C, s]
  - [in2] w: [OC, C, K]
  - [in3] b: [OC]
  - [out] out: [OC, os]

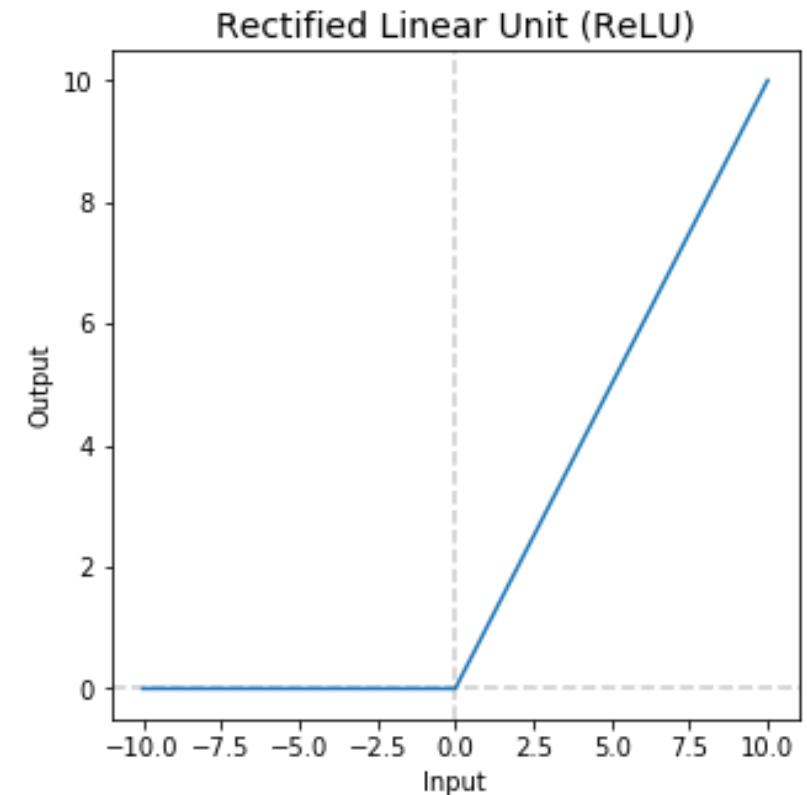


## Background – GetMax Layer

- 시퀀스(sequence) 차원을 기준으로 최대값을 구하는 연산
  - $out = \max(in, dim = -1)$
- 입출력
  - [in]    in: [C, s]
  - [out] out: [C]

## Background – ReLU Layer

- ReLU(Rectified Linear Unit) 활성화 함수를 적용하는 연산
  - $out = \max(0, in)$
- 입출력
  - [in & out] inout: [N]



(출처 : <https://towardsdatascience.com/deep-study-of-a-not-very-deep-neural-network-part-2-activation-functions-fd9bd8d406fc>)

## Background – Linear Layer

- 입력에 대한 선형 변환(Linear transformation)을 적용하는 연산
  - $out = in \times w^T + b$
- 입출력
  - [in1] in: [N]
  - [in2] w: [M, N]
  - [in3] b: [M]
  - [out] out: [M]

## Background - Layers (cont'd)

- 각 연산들에 대한 자세한 설명은 파이토치(PyTorch) 공식 문서 참고
  1. Embedding: <https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html>
  2. Permute: <https://pytorch.org/docs/stable/generated/torch.permute.html>
  3. Conv1D: <https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html>
  4. ReLU: <https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html>
  5. GetMax: <https://pytorch.org/docs/main/generated/torch.max.html>
  6. Concat: <https://pytorch.org/docs/stable/generated/torch.concat.html>
  7. Linear: <https://pytorch.org/docs/stable/generated/torch.nn.Linear.html>

## Background - Layers (cont'd)

- 각 연산들의 이론적 배경을 이해할 필요는 없음
  - 딥 러닝에 대해 전혀 몰라도 프로젝트를 진행하는 데에 문제 없음
  - 연산들의 특징 및 연산 간 dependence 등을 이해하는 것이 중요
  - 뼈대 코드를 보고 각 연산이 어떤 일을 하는지 이해할 것

# Skeleton Code & How to Run



# Skeleton Code

- `/shpc/skeleton/final-project/` 를 복사하여 진행
- `final-project`
  - | - Makefile
  - | - run.sh
  - | - data
    - |     `- inputs.bin, answers.bin
  - | - include
    - |     `- tensor.h, layer.h, model.h
  - `- src
    - `- tensor.cu, layer.cu, model.cu, main.cu

# How to Run (1)

- make 커맨드로 컴파일
  - \$ make

```
shpcta@ellogin3:~/final-project$ make
mkdir -p obj
mpic++ -std=c++14 -O3 -Wall -march=native -mavx2 -mfma -mno-avx512f -fopenmp -I/usr/local/cuda/
include -Iinclude -c -o obj/main.o src/main.cpp
/usr/local/cuda/bin/nvcc -Xcompiler=-std=c++14 -Xcompiler=-O3 -Xcompiler=-Wall -Xcompiler=-
march=native -Xcompiler=-mavx2 -Xcompiler=-mfma -Xcompiler=-mno-avx512f -Xcompiler=-fopenmp
-Xcompiler=-I/usr/local/cuda/include -Xcompiler=-Iinclude -c -o obj/model.o src/model.cu
/usr/local/cuda/bin/nvcc -Xcompiler=-std=c++14 -Xcompiler=-O3 -Xcompiler=-Wall -Xcompiler=-
march=native -Xcompiler=-mavx2 -Xcompiler=-mfma -Xcompiler=-mno-avx512f -Xcompiler=-fopenmp
-Xcompiler=-I/usr/local/cuda/include -Xcompiler=-Iinclude -c -o obj/tensor.o src/tensor.cu
/usr/local/cuda/bin/nvcc -Xcompiler=-std=c++14 -Xcompiler=-O3 -Xcompiler=-Wall -Xcompiler=-
march=native -Xcompiler=-mavx2 -Xcompiler=-mfma -Xcompiler=-mno-avx512f -Xcompiler=-fopenmp
-Xcompiler=-I/usr/local/cuda/include -Xcompiler=-Iinclude -c -o obj/layer.o src/layer.cu
cc -std=c++14 -O3 -Wall -march=native -mavx2 -mfma -mno-avx512f -fopenmp -I/usr/local/cuda/incl
ude -Iinclude -o main obj/main.o obj/model.o obj/tensor.o obj/layer.o -pthread -L/usr/local/cud
a/lib64 -lstdc++ -lcudart -lm -lmpi -lmpi_cxx
```

## How to Run (2)

- run.sh 스크립트로 실행
  - 실행 스크립트를 이해하고 옵션들을 확인할 것
  - \$ ./run.sh -h

```
shpcta@login3:~/final-project$ ./run.sh -h
salloc: Pending job allocation 867844
salloc: job 867844 queued and waiting for resources
salloc: job 867844 has been allocated resources
salloc: Granted job allocation 867844
Usage: ./main [-i 'pth'] [-p 'pth'] [-o 'pth'] [-a 'pth'] [-n 'num_sentences'] [-v] [-w] [-h]
Options:
  -i: Input binary path (default: ./data/inputs.bin)
  -p: Model parameter path (default: /home/s0/shpc_data/params.bin)
  -o: Output binary path (default: ./data/outputs.bin)
  -a: Answer binary path (default: ./data/answers.bin)
  -n: Number of input sentences (default: 1)
  -v: Enable validation (default: OFF)
  -w: Enable warm-up (default: OFF)
  -h: Print manual and options (default: OFF)
```

## How to Run (3)

- 실행 예시
  - \$ ./run.sh -n 4 -w -v

```
shpcta@ellogin3:~/final-project$ ./run.sh -n 4 -w -v
salloc: Pending job allocation 867846
salloc: job 867846 queued and waiting for resources
salloc: job 867846 has been allocated resources
salloc: Granted job allocation 867846

=====
Model: Sentiment Analysis
-----

Validation: ON
Warm-up: ON
Number of sentences: 4
Input binary path: ./data/inputs.bin
Model parameter path: /home/s0/shpc_data/params.bin
Answer binary path: ./data/answers.bin
Output binary path: ./data/outputs.bin
=====

Initializing inputs and parameters...Done!
Warming up...Done!
Predicting sentiment...Done!
Elapsed time: 5.372378 (sec)
Throughput: 0.744549 (sentences/sec)
Finalizing...Done!
Saving outputs to ./data/outputs.bin...Done!
Validating...PASSED!
```

## How to Run (4)

- 뼈대 코드에 제공된 파이썬 프로그램을 통해 실행 결과 확인
  - `$ python3 tools/bin2text.py <input_path> <output_path>`

```
shpcta@login3:~/final-project$ python3 tools/bin2text.py
Usage: python tools/bin2text.py <input_path> <output_path>
E.g, python tools/bin2text.py data/inputs.bin data/outputs.bin
<input_path> is the path to the inputs.bin file
<output_path> is the path to the outputs.bin file
shpcta@login3:~/final-project$ python3 tools/bin2text.py data/inputs.bin data/outputs.bin
Sentence #1
Input Sentence: i love sci-fi and am willing to put up with a lot . sci-fi <unk> are
Tokenized IDs: [12, 119, 1036, 6, 218, 1744, 8, 277, 70, 20, 5, 178, 3, 1036, 0, 30]
Predicted Sentiment: 1 (positive)
Probability: 0.7057

Sentence #2
Input Sentence: worth the entertainment value of a rental , especially if you like action movies . this
Tokenized IDs: [308, 2, 721, 1099, 7, 5, 2148, 4, 261, 52, 26, 45, 228, 98, 3, 14]
Predicted Sentiment: 1 (positive)
Probability: 0.7004

Sentence #3
Input Sentence: its a totally average film with a few <unk> action sequences that make the plot seem
Tokenized IDs: [100, 5, 482, 809, 23, 20, 5, 174, 0, 228, 898, 15, 106, 2, 114, 321]
Predicted Sentiment: 1 (positive)
Probability: 0.5280

Sentence #4
Input Sentence: star rating ***** saturday night **** friday night *** friday morning ** sunday night * monday
Tokenized IDs: [341, 706, 5696, 1911, 306, 1829, 2204, 306, 2965, 2204, 1902, 3478, 2090, 306, 2551, 7681]
Predicted Sentiment: 0 (negative)
Probability: 0.7420
```

# Project Rules & Restrictions

# 프로젝트 규칙 및 주의사항 (1)

- 수정 불가능한 파일
  - inputs.bin, answers.bin: 입력과 정답 바이너리 파일
  - params.bin: 학습된 모델 파라미터가 저장된 바이너리 파일
  - model.h, main.cpp, Makefile: 그 외 수정 불가능한 파일
- 수정 가능한 파일 (제출 파일)
  - tensor.h, tensor.cu
  - layer.h, layer.cu
  - model.cu.
  - run.sh: 적절하게 프로그램 실행 옵션을 추가 및 수정하여 사용할 것
    - 프로젝트 제출 이후 성능 재현 및 평가는 ./run.sh -v 명령으로 실행할 것이므로 각자 최고 성능이 나오는 옵션(-n, -w 옵션 등)으로 ./run.sh 파일을 수정한 후 제출할 것

## 프로젝트 규칙 및 주의사항 (2)

- 프로그램 로직 혹은 모델 구조를 변경하는 수정은 불허
- 가능한 수정의 예시
  - 메모리 레이아웃 변경, 루프 순서 변경, 패딩 데이터/연산 추가, 커널 병합(kernel fusion) 등
- 불가능한 수정의 예시
  - 시간 측정 부분(predict\_sentiment 함수) 외에서 모델 추론 연산 수행
  - 동일한 출력을 만들어내는 다른 모델/알고리즘 사용 등
- **애매한 것은 조교에게 문의할 것**



## 프로젝트 규칙 및 주의사항 (3)

- CUDA 외의 라이브러리 사용 불가능
  - 사용 불가능한 라이브러리 예시: cuBLAS, cuDNN, cBLAS, MAGMA, BLIS, PyTorch, Tensorflow 등등
  - GPU 사용 시 CUDA를 권장하나, OpenCL을 꼭 사용하고 싶다면 조교에게 먼저 문의
- Tensor Core 사용 가능 (입력: FP16, 출력: FP16 or FP32)
  - 현재 뼈대 코드는 모두 FP32 (float) 데이터 타입으로 동작하며, Validation 또한 FP32로 진행하므로 데이터 타입과 타입 변환(casting)에 유의
  - 부동소수점(Floating point) 오차로 Validation에 실패하지 않도록 주의
- **애매한 것은 조교에게 문의할 것**

# Grading & Submission

# 프로젝트 평가방법

- **성능 (80%)**

- 성능 기준: Throughput (sentences/sec)
- 하나의 입력 문장의 길이는 16개의 토큰으로 고정 (변경 불가)
- 처리할 입력 문장의 개수는 본인이 원하는 값으로 설정 가능 (최대: 16384)
- 1등부터 4등까지는 만점, 4등 대비 2배씩 성능이 감소할 때마다 10% 감점
- Validation 실패 시 0점 처리

- **레포트 (20%)**

- 파일명: report.pdf
- 5 페이지 이내로 본인이 적용한 최적화 내용을 간략히 작성
- 본인이 측정한 1) 성능(Throughput) 및 2) 성능 화면 캡처 이미지 첨부할 것

# 프로젝트 제출

- 마감 시간: 2024. 12. 18. 11:59 PM
  - 마감 시간을 초과하여 제출한 것에 대해서는 실격 처리
  - 실습 서버, 시스템 문제 등 천재지변에 준하는 경우 예외 처리
- shpc-submit 유틸리티를 사용하여 제출
- 제출 파일 (총 7개)
  - tensor.h, tensor.cu, layer.h, layer.cu, model.cu, run.sh
  - report.pdf
  - 프로젝트 제출 이후 성능 재현 및 평가는 오로지 ./run.sh -v 명령으로 실행되므로 입력 개수(-n)를 ./run.sh에 설정해둘 것
    - [X] ./run.sh -n 32 -w -v
    - [O] ./run.sh -v

# Comments

## Comments (1)

- 각 연산의 특징, 연산 간 dependence 등을 파악하는 것이 중요
- 뼈대 코드를 먼저 잘 이해한 뒤에 시작하는 것을 추천
- 항상 근거를 가지고 최적화를 해야 함
  - 최적화에 앞서 어디가 문제인지 찾아내는 것이 기본
  - [X] '교수님이 말씀하시길, kernel fusion 이 좋다더라'
  - [O] 'Kernel launch 횟수가 너무 많으니, kernel fusion 을 통해 kernel launch overhead 를 줄이자'
  - [X] 'Convolution 연산이 제일 복잡해 보이니까 최적화 하는 게 좋지 않을까?'
  - [O] '실행 시간을 측정해보니 Convolution 연산이 가장 오래 걸리니까 이것부터 최적화 하자'

## Comments (2)

- 해볼만한 것들
  - 여러 개의 입력을 묶음으로 처리 (batching)
  - Peak FLOPS 대비 얼마나 빠른지 계산
  - 커널 최적화
    - Matmul, Convolution 연산 최적화
    - Reduction
    - Kernel fusion
  - 프로파일링
    - 수업시간에 배운 Nsight Systems, Compute 사용

# Updates



# Updates

## [24.12.12]

- './Makefile' 수정 ('-arch=sm\_70' for TC)

## [24.12.14]

- 'main.cpp' 수정 (124번째 줄, pinned memory pointer를 새로 덮어쓰는 문제 수정)

## [24.12.16]

- './Makefile' 수정 (-arch=sm\_75 -gencode=arch=compute\_75,code=sm\_75 추가)