

# Final Project - namegen

M1522.006700 확장형 고성능 컴퓨팅

M3239.005400 데이터사이언스를 위한 컴퓨팅 2

# Project Goal

- 순차 코드로 구현되어 있는 딥 러닝 추론 프로그램을 병렬화/최적화
  - 4 개 계산 노드의 모든 자원 사용
  - Pthread, OpenMP, MPI, CUDA 사용 가능
  - 외부 라이브러리 사용 불가능
- 영어 이름 생성 모델
  - Sequence model with GRU layers

N: # of names to generate  
rng\_seed: seed for randomness

**Input**



Pretrained Model



Karlen  
Elisah  
Stephen  
Christiano  
...

**Output**

# Example Run

- Build

```
● kjp4155@login0:~/skeleton$ ls
main.cpp Makefile model.bin namegen.cu namegen.h output.txt run.sh util.cpp util.h
● kjp4155@login0:~/skeleton$ make
g++ -std=c++14 -O3 -Wall -march=native -mavx2 -mfma -mno-avx512f -fopenmp -I/usr/local/cuda/include -c -o main.o main.cpp
g++ -std=c++14 -O3 -Wall -march=native -mavx2 -mfma -mno-avx512f -fopenmp -I/usr/local/cuda/include -c -o util.o util.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 -c -o namegen.o namegen.cu
cc -std=c++14 -O3 -Wall -march=native -mavx2 -mfma -mno-avx512f -fopenmp -I/usr/local/cuda/include -o main main.o util.o namegen.o -pthread -L/usr/local/cuda/lib64 -lmpi_cxx -lmpi -lstdc++ -lcudart -lm
● kjp4155@login0:~/skeleton$ ls
main      main.o      model.bin    namegen.h   output.txt   util.cpp   util.o
main.cpp  Makefile    namegen.cu   namegen.o   run.sh       util.h
○ kjp4155@login0:~/skeleton$
```

# Example Run

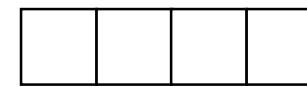
- Run

```
● kjp4155@login0:~/skeleton$ ./run.sh model.bin output.txt 20 4155
Generating 20 names...Done!
First 8 results are: Karlen, Elisah, Devonda, Stephen, Christiano, Mikelle, Madaline, Benuel
Writing to output.txt ...Done!
Elapsed time: 1.566710 seconds
Throughput: 12.766 names/sec
● kjp4155@login0:~/skeleton$ cat output.txt
Karlen
Elisah
Devonda
Stephen
Christiano
Mikelle
Madaline
Benuel
Crespin
Kolette
Librada
```

# Background - Tensor

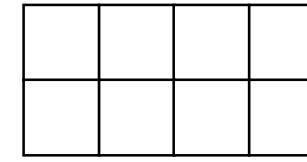
- 딥 러닝에서 데이터를 다루는 단위
  - 뼈대 코드에 구현되어 있는 연산들은 tensor 를 입출력으로 가짐

1D Tensor  
(e.g., Vector)



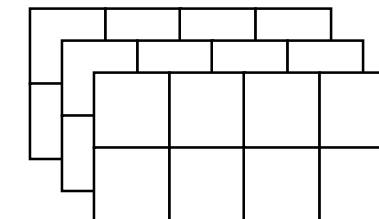
Shape = {4}

2D Tensor  
(e.g., Matrix)



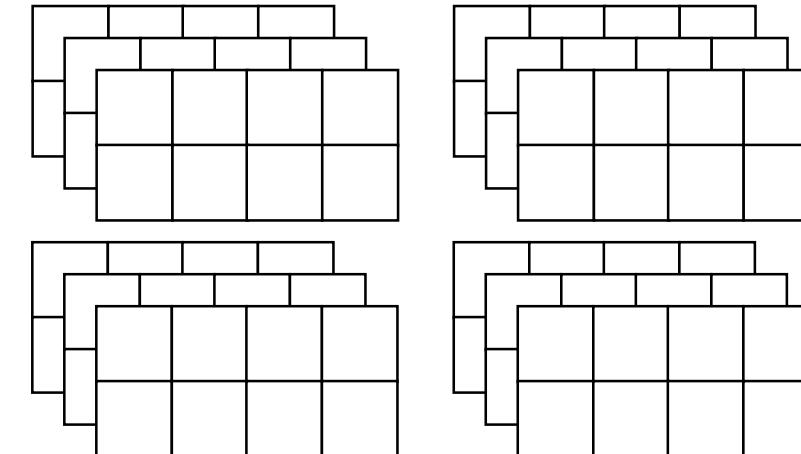
Shape = {2, 4}

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



Shape = {3, 2, 4}

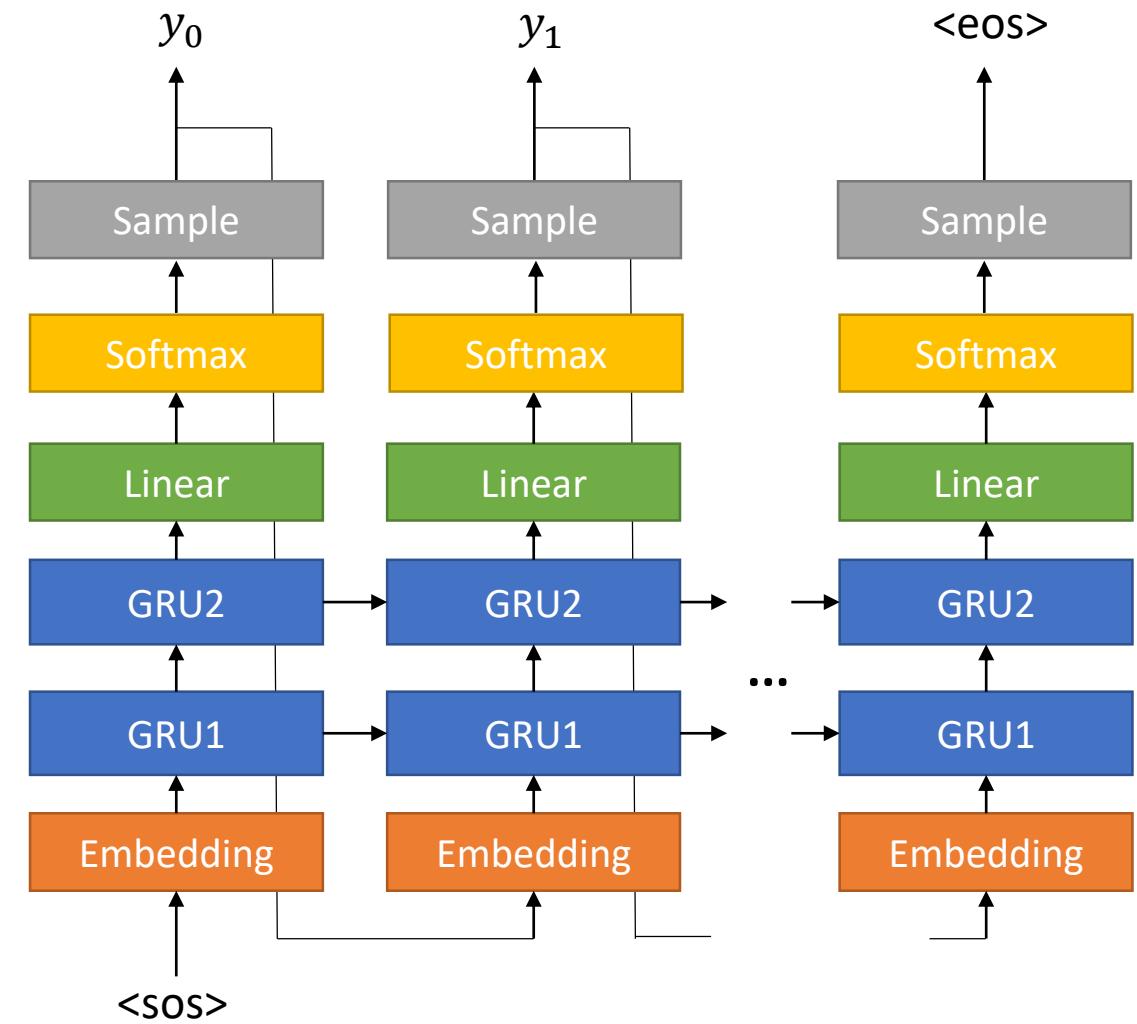
4D Tensor  
(e.g., multiple images, convolution weight)



Shape = {4, 3, 2, 4}

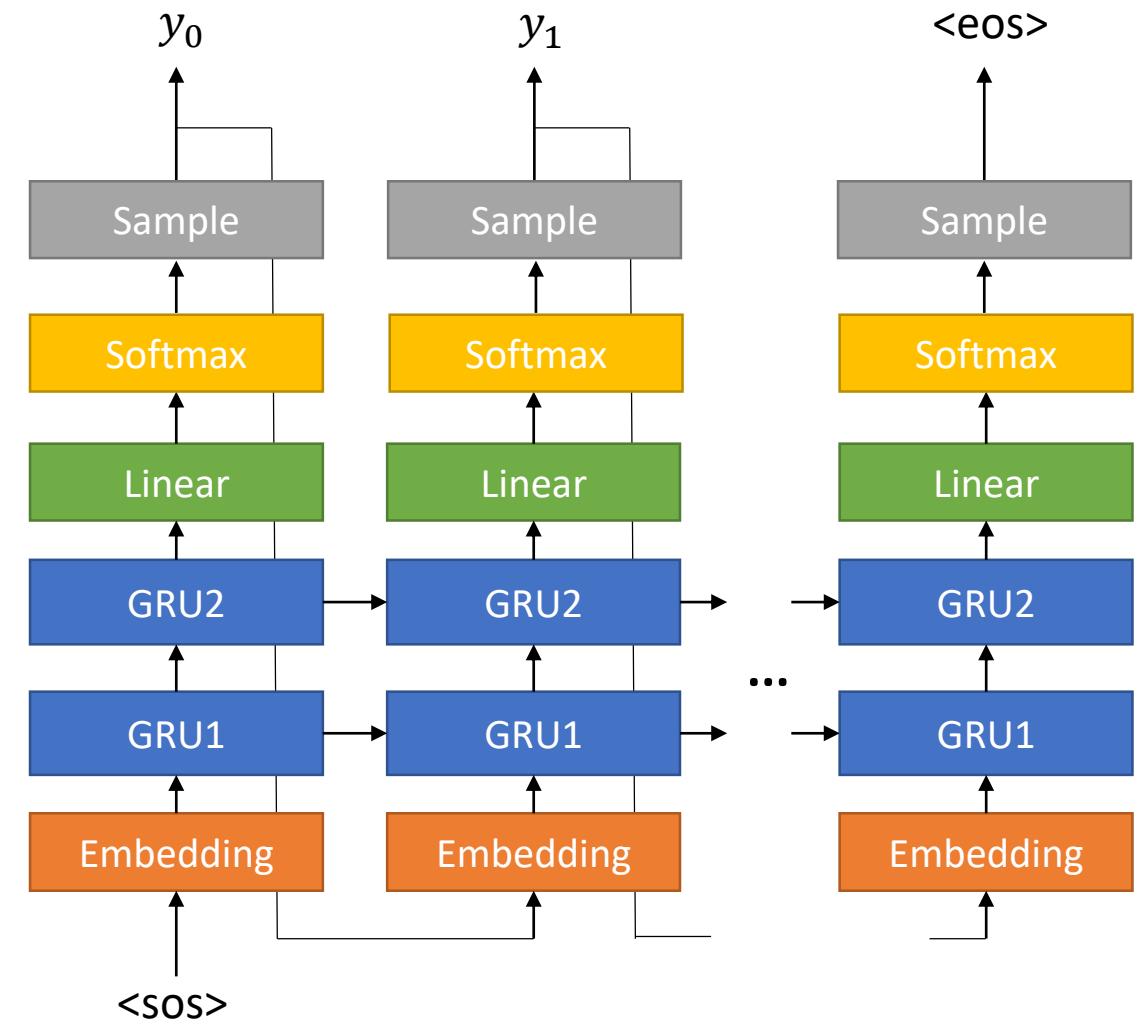
# Background - Model Architecture

- Sequence model with 2 GRU layers
- A name is a sequence of  $l + 1$  characters
  - $y_0 \ y_1 \dots \ y_l \text{ <eos>}$
- Iteratively run model inference for  $L$  times
  - $L = 10$  for this project
  - Start from `<sos>` character
  - Can stop early if the model outputs `<eos>` character



# Background - Model Architecture

- Embedding
  - Map a character to embedding vector
  - Input  
Index of the input character
  - Output  
Embedding vector of the input character



# Background - Model Architecture

- GRU1

- Run following operations

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr})$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz})$$

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \cdot (W_{hn}h_{t-1} + b_{hn}))$$

$$h_t = (1 - z_t) \cdot n_t + z_t \cdot h_{t-1}$$

- Input

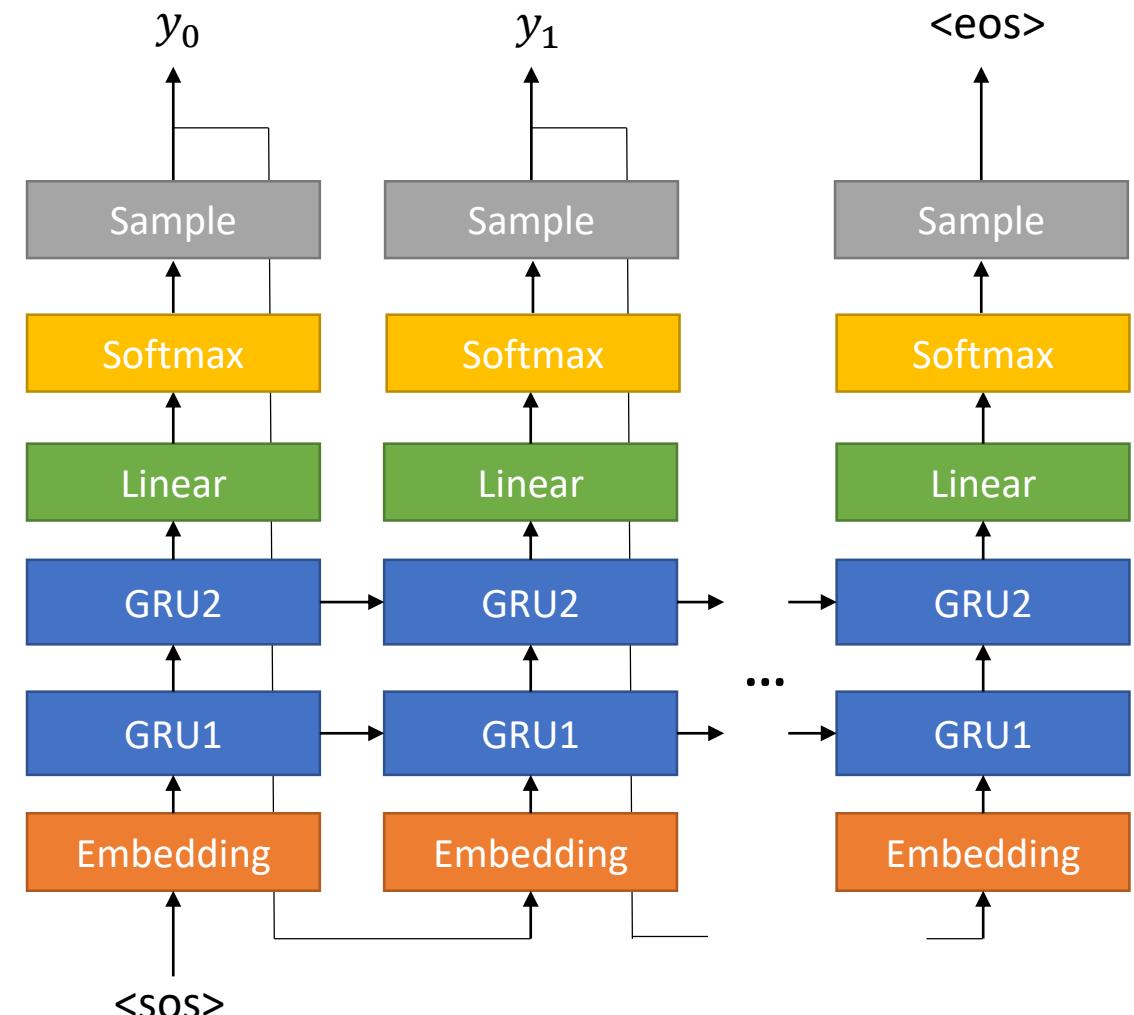
$x_t$ : output of the Embedding layer

$h_{t-1}$ : hidden vector from previous iteration

- Output

$h_t$ : hidden vector of current iteration

- $W, b$  are model parameters
- $\sigma(), \tanh()$  are element-wise functions
- $+, -, \cdot$  are element-wise operations



# Background - Model Architecture

- GRU2

- Run following operations

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr})$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz})$$

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \cdot (W_{hn}h_{t-1} + b_{hn}))$$

$$h_t = (1 - z_t) \cdot n_t + z_t \cdot h_{t-1}$$

- Input

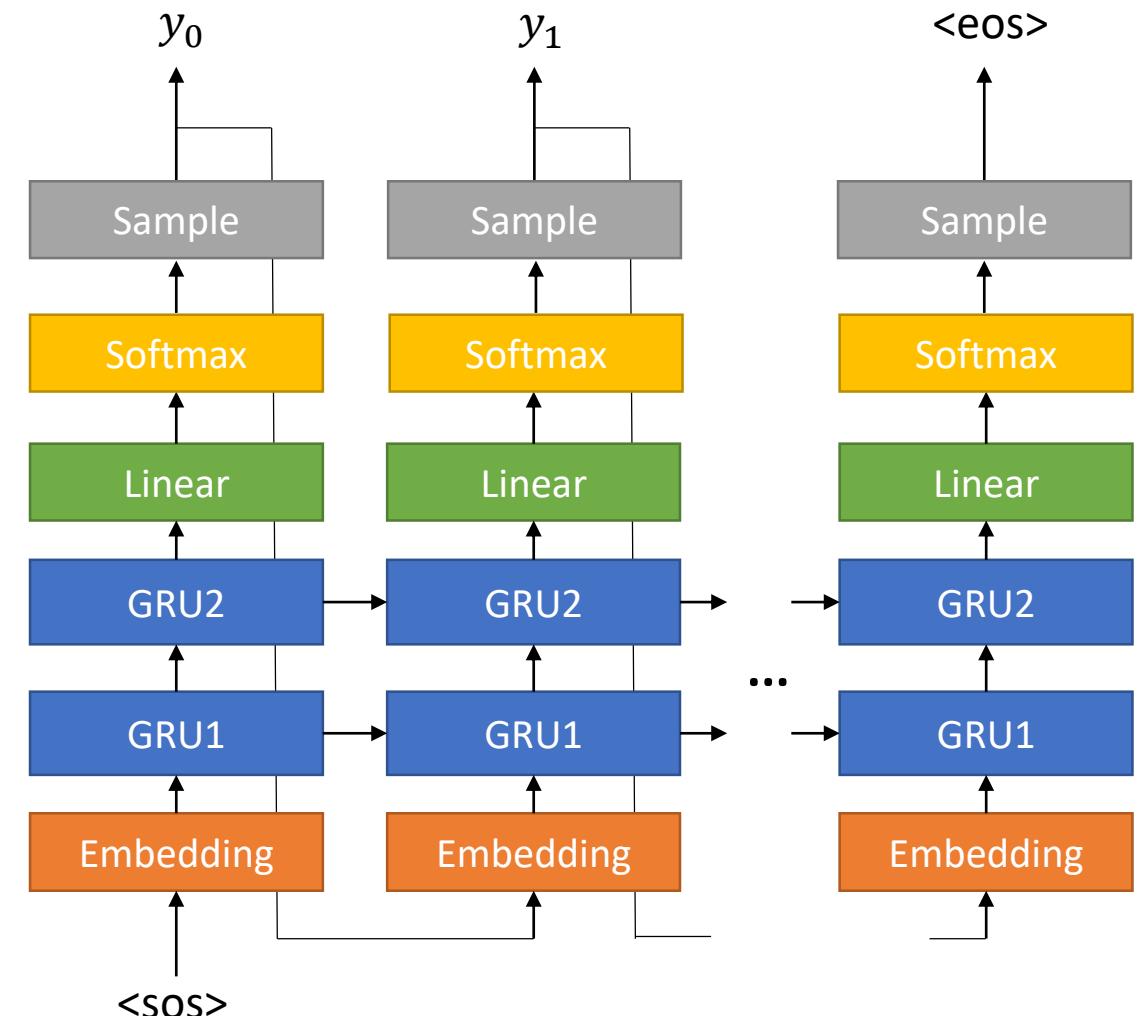
$x_t$ : output of the previous GRU layer

$h_{t-1}$ : hidden vector from previous iteration

- Output

$h_t$ : hidden vector of current iteration

- $W, b$  are model parameters
- $\sigma(), \tanh()$  are element-wise functions
- $+, -, \cdot$  are element-wise operations



# Background - Model Architecture

- Linear

$$y = Wx + b$$

- Input

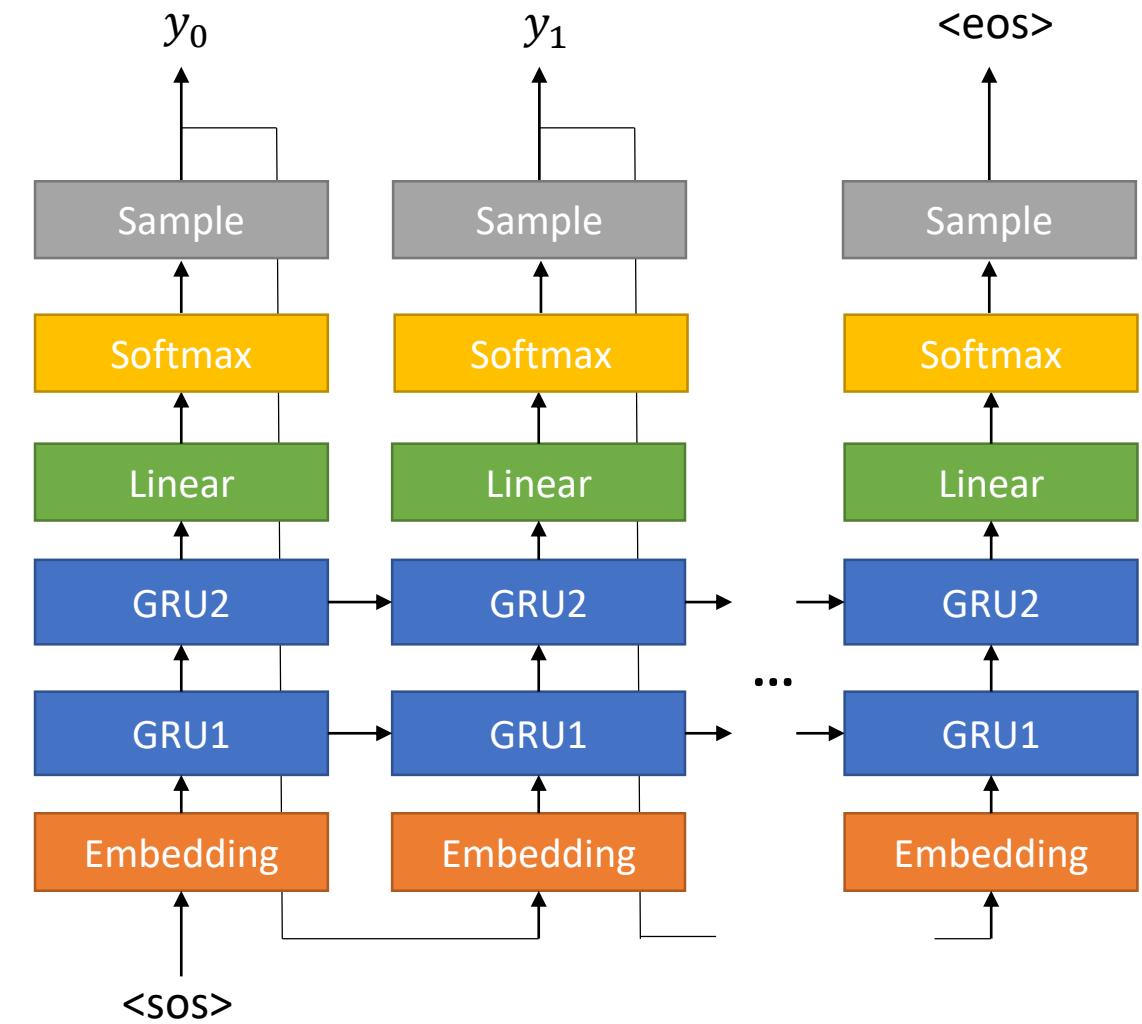
$x$ : output of the GRU2 layer

- Output

$y$ : output vector

- $W, b$  are model parameters

- $+$  is element-wise addition



# Background - Model Architecture

- Softmax

$$y_i = \frac{\exp(x_i)}{\sum_i \exp(x_i)}$$

- Input

$x$ : output of the Linear layer

- Output

$y$ : output vector

- Output can be seen as a probability distribution

- Sample

- Randomly select an element from given probability distribution

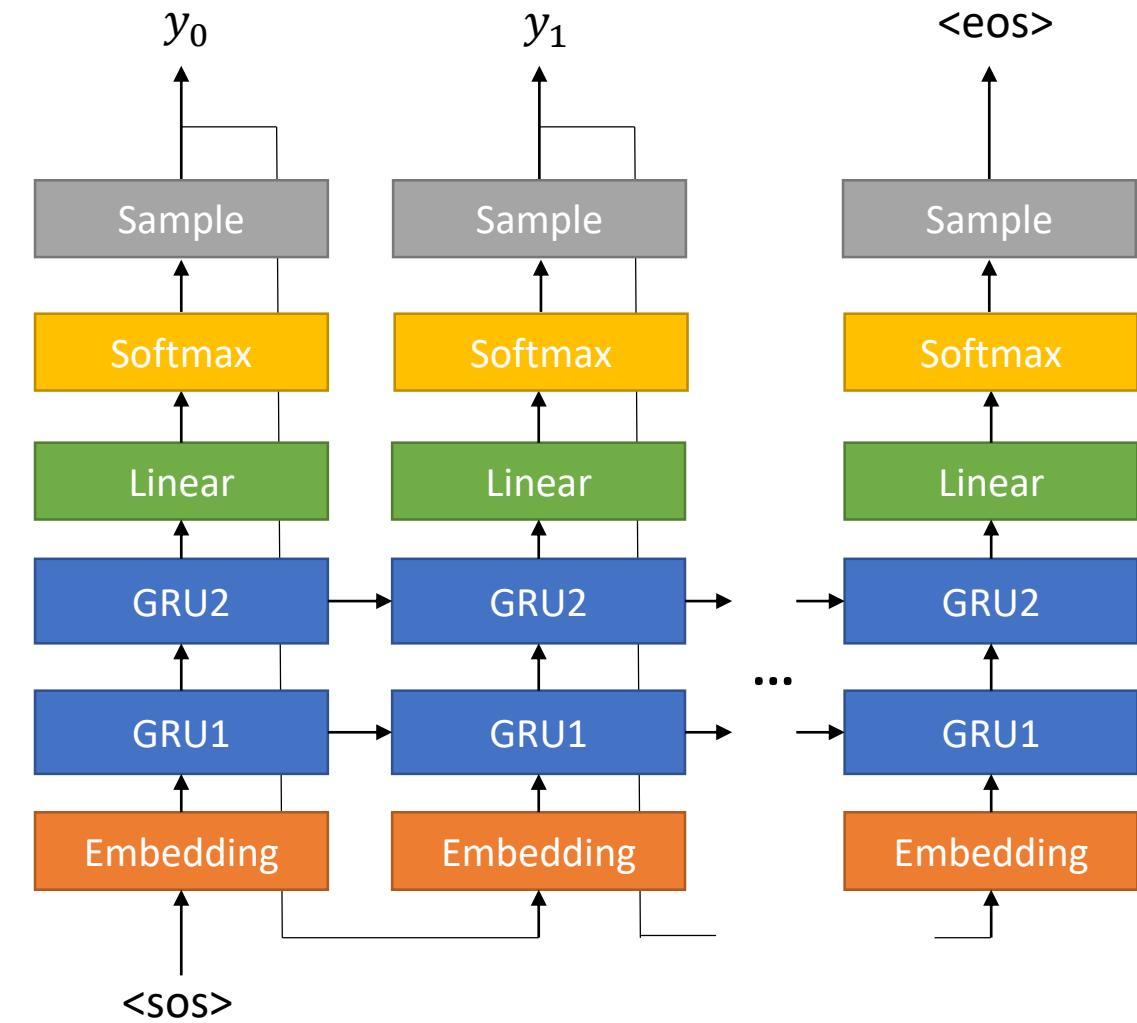
- Input

$p$ : output of the Softmax layer

$r$ : a random float

- Output

$y$ : selected element (character)



# Background

- 각 연산의 이론적 배경을 이해할 필요는 없음
  - 딥 러닝에 대해 전혀 몰라도 프로젝트를 진행하는 데 문제 없음
  - 연산들의 특징 및 연산간 dependence 를 이해하는 것이 중요
  - 뼈대 코드를 보고 각 연산이 어떤 일을 하는지 이해할 것
- 각 연산들에 대한 더욱 자세한 설명은 PyTorch 문서를 참고
  - Embedding: <https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html?highlight=embedding#torch.nn.Embedding>
  - GRU: <https://pytorch.org/docs/stable/generated/torch.nn.GRU.html?highlight=gru#torch.nn.GRU>
  - Linear: <https://pytorch.org/docs/stable/generated/torch.nn.Linear.html?highlight=linear#torch.nn.Linear>
  - Softmax: <https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html?highlight=softmax#torch.nn.Softmax>

# Skeleton Code

- 로그인 노드의 /shpc22/skeleton/final-project 디렉토리
- 수정 불가능한 파일
  - model.bin: 학습된 모델 파라미터가 저장된 파일
  - main.cpp, util.cpp, namegen.h, util.h
- 수정 가능한 파일
  - namegen.cu: 병렬화 대상
  - Makefile, run.sh: 필요에 따라 적절하게 수정해서 사용
    - make 명령으로 main 이 빌드 되어야 함
    - ./run.sh model.bin output.txt N seed 명령으로 실행 되어야 함

# Constraints

- 프로그램 로직 혹은 모델 구조를 변경하는 수정은 불허
- 가능한 수정의 예시
  - 메모리 레이아웃 변경, 루프 순서 변경, 패딩 데이터/연산 추가, fusion 등
  - <eos> character에서 멈추지 않고  $L$  회의 iteration을 모두 실행하는 것
  - zero element 를 파악해 일부 연산을 생략
- 불가능한 수정의 예시
  - Initialize 함수에서 모델 추론 관련 연산 수행
  - 동일한 출력을 만들어내는 다른 모델/알고리즘 사용
- 애매한 것은 조교에게 문의할 것

# Constraints

- x86 intrinsics, Pthread, OpenMP, MPI, CUDA 외 라이브러리 사용 불가능
  - 사용 불가능한 라이브러리 예시: cuBLAS, cuDNN, cBLAS, MAGMA, BLIS, PyTorch, Tensorflow, ...
  - GPU 사용 시 CUDA를 권장하나, OpenCL을 꼭 사용하고 싶다면 조교에게 문의
- 애매한 것은 조교에게 문의할 것

# Grading

- 성능 (80%)
  - 최종 성능에 따른 상대 평가
  - log scale
  - 1등에 비해 2배 느릴때마다 10% 감점
    - e.g., 1등 프로그램이 100 names/sec 이라면, 25 names/sec 프로그램은 성능 점수의 80% 부여
- 보고서 (20%)
  - report.pdf, 양식 및 분량 자유
  - 자신이 한 일들 위주로 설명
  - 병렬화 방법, 메모리 레이아웃, 최적화 기법 등 조교에게 알리고자 하는 내용을 설명
  - 자신이 측정한 프로그램의 성능을 포함할 것. 조교가 측정한 성능과 차이가 나는 경우 2차 확인을 위함

# Grading

- 채점 프로세스

1. 제출된 모든 파일을 tmp 디렉토리에 복사
  2. 뼈대 코드의 main.cpp, util.cpp, namegen.h, util.h, model.bin 을 tmp 디렉토리에 복사
  3. tmp 디렉토리에서 make 명령으로 컴파일
  4. tmp 디렉토리에서 ./run.sh model.bin output.txt N seed 명령으로 프로그램 실행
  5. 실행 결과가 뼈대 코드의 결과와 정확히 일치하는지 확인
    - 실수 오차를 감안해, 대략 99% 이상의 이름이 일치하면 통과
- N = [16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, ...]
  - seed: 비공개

- 다양한 N 중 가장 높은 성능을 내는 것이 본인의 최종 성능
  - 프로그램이 1분 이내에 종료되지 않는 경우 제외
  - 답이 틀린 경우 제외

# Submission

- Deadline: 12월 19일 오후 11:59:59
  - Grace day 사용 불가
- submit 스크립트를 이용해 제출
  - shpc22-submit submit final-project namegen.cu
  - shpc22-submit submit final-project Makefile
  - shpc22-submit submit final-project run.sh
  - shpc22-submit submit final-project report.pdf

# Comments

- 각 연산의 특징, 연산간 dependence 등을 이해하는 것이 중요
- 뼈대 코드를 잘 이해한 뒤 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 많다. Kernel fusion 을 통해 kernel launch overhead 를 줄이자
  - [X] Select 연산이 제일 복잡해 보이니까 최적화하는게 좋지 않을까?
  - [O] 실행 시간을 측정해 보니 Select 연산이 가장 오래 걸린다.

# Comments

- ETL 게시판을 통해 성능 공유 및 토론 권장
- 해볼만한 것들
  - 여러개의 입력을 묶음(batch)으로 처리
  - 그동안 해왔던 matmul 최적화
  - 각 이름마다 길이가 다른것을 어떻게 잘 처리할지 생각
  - CPU-GPU 통신 시간을 감안해도, 몇몇 연산은 CPU에서 하는 것이 이득일수도 있음
  - 조교의 추측일 뿐, 정답은 없음
- 가급적 빨리 시작할 것
  - Deadline 1~2일 전에는 작업이 몰려 제대로 된 실험이 불가능

# Q&A

# 2022.12.14 TA Session

# Grading

- 채점 프로세스
  1. 제출된 모든 파일을 tmp 디렉토리에 복사
  2. 뼈대 코드의 main.cpp, util.cpp, namegen.h, util.h, model.bin 을 tmp 디렉토리에 복사
  3. tmp 디렉토리에서 make 명령으로 컴파일
  4. tmp 디렉토리에서 ./run.sh model.bin output.txt N seed 명령으로 프로그램 실행
  5. 실행 결과가 뼈대 코드의 결과와 정확히 일치하는지 확인
    - 실수 오차를 감안해, 대략 99% 이상의 이름이 일치하면 통과
    - $N = [16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, \dots]$
    - seed: 비공개
- 다양한 N 중 가장 높은 성능을 내는 것이 본인의 최종 성능
  - 프로그램이 1분 이내에 종료되지 않는 경우 제외
  - 답이 틀린 경우 제외

# Submission

- Deadline: 12월 19일 오후 11:59:59
  - Grace day 사용 불가
- submit 스크립트를 이용해 제출
  - shpc22-submit submit final-project namegen.cu
  - shpc22-submit submit final-project Makefile
  - shpc22-submit submit final-project run.sh
  - shpc22-submit submit final-project report.pdf

# Comments

- 각 연산의 특징, 연산간 dependence 등을 이해하는 것이 중요
- 뼈대 코드를 잘 이해한 뒤 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 많다. Kernel fusion 을 통해 kernel launch overhead 를 줄이자
  - [X] Select 연산이 제일 복잡해 보이니까 최적화하는게 좋지 않을까?
  - [O] 실행 시간을 측정해 보니 Select 연산이 가장 오래 걸린다.

# Comments

- ETL 게시판을 통해 성능 공유 및 토론 권장
- 해볼만한 것들
  - 여러개의 입력을 묶음(batch)으로 처리
  - 그동안 해왔던 matmul 최적화
  - 각 이름마다 길이가 다른것을 어떻게 잘 처리할지 생각
  - CPU-GPU 통신 시간을 감안해도, 몇몇 연산은 CPU에서 하는 것이 이득일수도 있음
  - 조교의 추측일 뿐, 정답은 없음
- 가급적 빨리 시작할 것
  - Deadline 1~2일 전에는 작업이 몰려 제대로 된 실험이 불가능

## Q&A

- [Q1] namegen\_initialize에서 random\_floats를 생성해도 되나요?

- [A1]

namegen\_initialize에서 rng\_seed를 사용하는 최적화는 허용되지 않습니다.

namegen\_initialize의 파라미터로 rng\_seed가 들어오는 것은 의도한 바가 아닙니다. 뼈대 코드를 작성한 조교의 실수입니다.

## Q&A

- [Q2] namegen\_initialize에서 weight 을 전처리한 뒤 GPU메모리에 올려도 되나요?
- [A2]  
모델의 입력 (random\_floats) 를 모르는 상황에서 할 수 있는 전처리는 initialize 에서 하셔도 됩니다.  
다만 initialize 함수에서 rng\_seed 를 이용하는 최적화는 불가능합니다.

## Q&A

- [Q3] N 이 node의 배수라 가정해도 되나요?

- [A3]

네, N은 16부터 시작해 2의 지수승으로 주어집니다. (16, 32, 64 ...)

이외의 N에 대해서는 프로그램이 정상적으로 작동하지 않아도 괜찮습니다.

# Q&A

- [Q4] 실수 계산 오차때문에 답이 다르게 나옵니다.
- [A4]

채점 시 실수 계산 오차는 감안할 예정입니다.  
최적화한 프로그램으로 생성한 이름들이 일정 비율(대략 99%) 이상 정확하면 답으로 인정할  
계획입니다.  
다만, 이를 악용해 고의적으로 연산/통신/동기화 등을 누락하는 것은 허용하지 않습니다.  
1%정도의 이름을 아예 생성하지 않는 등의 최적화는 적발 시 0점 처리되므로 주의해 주세요.