Project 5 Overview

KECE456 Code and System Optimization (Fall 2024)

T.A. Jisung Pack

School of Electrical and Computer Engineering

Korea University, Seoul

Contents

- **Environmental Setup**
  - Recommendations
  - Prerequisite 1: Install Anaconda
  - Prerequisite 2: Download GPT-2 & PyTorch Source
  - Prerequisite 3: Install gcc-9
  - Build PyTorch with Source
- **Benchmark & Modification**
  - Executing A Language Model
  - Optimizing A Matrix Multiplication Library
- **Profiling Guide**
  - PyTorch Profiler (Execution Time Breakdown)
  - perf-stat/perf-mem (Hardware Event Count)
- **Appendix (Additional)**
  - About AI Model & GPT-2 Model

Environmental Setup

Recommendations

- **If you are using VMware, apply the settings below:**
  - Virtual Machine Settings -> Processors
    - Number of processor cores >= 4
    - Enable "Virtualize CPU performance counters"
    - Enable "Virtualize IOMMU (IO memory management unit)"
  - Virtual Machine Settings -> Processors
    - Memory for this virtual machine >= 16384MB
      - If your memory size is small, building PyTorch may fail…
      - If your RAM size is small, expand swap memory

Prerequisite 1: gcc-9 Install

- **To build PyTorch with source, gcc with version > 9.3 is required.**
  - If your gcc version is already higher than 9.3, pass the commands below.
  - gcc --version
- **command list (for gcc upgrade)**
  - sudo add-apt-repository ppa::ubuntu-toolchain-r/test
  - sudo apt update
  - sudo apt install gcc-9
  - sudo apt install g++-9
  - sudo update-alternatives--install /usr/bin/gcc gcc /usr/bin/gcc-9 60 --slave /usr/bin/g++ g++ /usr/bin/g++-9

Prerequisite 2: Anaconda install

- **Anaconda provides virtual environment specialized on Python.**
- **command list**
  - wget https://repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86_64.sh
  - sh Anaconda3-2022.05-Linux-x86_64.sh
    - If you already have anaconda installed, you can skip this procedure.
  - conda create -n proj5 python=3.9
  - conda activate proj5

Prerequisite 3: Download GPT-2 & PyTorch Source

- **Download GPT-2**
  - HuggingFace provides lots of AI models & dataset for free.
  - command list
    - cd (YOUR_WORKSPACE)
    - sudo apt-get install git-lfs
    - git clone https://huggingface.co/openai-community/gpt2
- **Download PyTorch Source**
  - In this project, we are modifying some part of PyTorch.
  - To do so, we need to access C++ code of PyTorch.
  - NOTICE!! Do not install pytorch with conda or pip commands…
  - command list
    - cd (YOUR_WORKSPACE)
    - git clone https://github.com/pytorch/pytorch.git

Build PyTorch with Source

- **Reference: https://github.com/pytorch/pytorch**
- **command list**
  - cd (YOUR_WORKSPACE)/pytorch
  - conda activate proj5
  - conda install ninja
  - conda install rust
  - pip install -r requirements.txt
  - pip install mkl-static mkl-include
  - export CMAKE_PREFIX_PATH="${CONDA_PREFIX:-'$(dirname $(which conda))/../'}:${CMAKE_PREFIX_PATH}"
  - python setup.py develop
  - conda install -c conda-forge libstdcxx-ng=12

Benchmark & Modification

Executing A Language Model (1)

- **run.py**
  - Execute text generation with 1 input sentence(single batch).
    - Sequence length(=number of tokens) of input sentence is 835.
    - Maximum length of generated text is 836.
    - Therefore, only 1 token is generated.
      - a.k.a. summarization stage(↔generation stage)
  - Line 1~21: code for preparing inference
  - Line 22~ : code for running inference & show the result

Executing A Language Model (2)

- **PyTorch Profiler**
  - in "run.py", line 24~29, line 53 → release and execute.
    - To use PyTorch profiler, block line 30 & 52.

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::addmm	43.66%	10.663s	44.29%	10.816s	5.496ms	1968
aten::linear	0.00%	377.931us	20.14%	4.917s	119.934ms	41
aten::matmul	0.01%	2.103ms	20.13%	4.916s	119.908ms	41
aten::mm	20.12%	4.913s	20.12%	4.914s	119.842ms	41
aten::cat	6.50%	1.587s	6.63%	1.620s	1.555ms	1042
aten::sort	5.26%	1.285s	6.47%	1.581s	38.557ms	41
aten::softmax	0.00%	1.084ms	5.10%	1.246s	15.201ms	82
aten::_softmax	5.10%	1.245s	5.10%	1.245s	15.187ms	82
aten::multinomial	0.02%	5.554ms	3.51%	856.735ms	20.896ms	41
aten::copy_	3.49%	851.299ms	3.49%	851.299ms	188.466us	4517

Optimizing A Matrix Multiplication Library (1)

- **Matrix Multiplication(baseline)**
  - If you run GeMM on CPU with PyTorch, it finally ends up at cpublas::gemm defined in pytorch/aten/src/ATen/native/CPUBlas.cpp.
  - cpublas::gemm calls GeMM functions provided by Intel MKL library.
  - To use our baseline GeMM code, some modifications are required.
    - Replace pytorch/aten/src/ATen/native/LinearAlgebra.cpp to LinearAlgebra.cpp that I provided.
      - I defined baseline GeMM function in LinearAlgebra.cpp. (void matmul_proj5( ))
      - For-loop in matmul_proj5( ) can be executed in parallel with OpenMP threads.
    - Build PyTorch again:
      - cd (YOUR_WORKSPACE)/pytorch
      - python setup.py develop
  - NOTICE!! Every printf( ) should be blocked for performance analysis.

Optimizing A Matrix Multiplication Library (2)

- **Matrix Multiplication Optimization (TODO)**
  - Apply code optimization on matmul_proj5( ) function.
  - Build PyTorch again.

Profiling Guide

Profiling Guide (1)

- **Checking your CPU information**
  - command: lscpu, cpuinfo, cpuid, …
  - Check how many cores you have. (computing ability)
  - Check which type of SIMD instructions the core use.
    - AVX, SSE, …
    - Single-thread-execution = single-element-at-a-time? NO!

```
(proj5) jisung@hera:~/work/compiler_2024/project5$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              8
On-line CPU(s) list: 0-7
Thread(s) per core:  2
Core(s) per socket:  4
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               58
Model name:          Intel(R) Xeon(R) CPU E3-1270 V2 @ 3.50GHz
Stepping:            9
CPU MHz:             1596.526
CPU max MHz:         3900.0000
CPU min MHz:         1600.0000
BogoMIPS:            6984.23
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            8192K
NUMA node0 CPU(s):   0-7
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm cons
tant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid sse4
_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm cpuid_fault epb pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase s
mep erms xsaveopt dtherm ida arat pln pts md_clear flush_l1d
```

Profiling Guide (2)

- **perf stat**
  - Linux provides an interface for gathering performance counter statistics.
  - Reference: https://man7.org/linux/man-pages/man1/perf-stat.1.html
    - "man perf-stat"
  - "perf list" : list of events you can track while running a program
  - "sudo perf stat -e (events) (program)"
    - ex) sudo perf stat -e cycles,instructions,uops_executed.stall_cycles,branch-misses,cache-references,cache-misses /home/jisung/anaconda3/envs/proj5/bin/python run.py

```
Performance counter stats for '/home/jisung/anaconda3/envs/proj5/bin/python run.py':

   53,452,380,715      cycles
  116,184,466,898      instructions              #    2.17  insn per cycle
   10,713,373,046      uops_executed.stall_cycles
       85,255,095      branch-misses
      283,696,427      cache-references
       79,617,735      cache-misses              #   28.064 % of all cache refs

      5.911468796 seconds time elapsed
```

Profiling Guide (3)

- **perf mem**
  - Linux provides an interface for gathering memory operation data statistics.
  - Reference: https://man7.org/linux/man-pages/man1/perf-mem.1.html
    - "man perf-mem"
  - "sudo perf mem record (program)"
    - Gather data for memory operation
  - "sudo perf mem -t load report --sort=mem"
    - Show the data gathered

```
Samples: 87K of event 'cpu/mem-loads,ldlat=30/P', Event count (approx.): 27218671
Overhead       Samples  Memory access
  73.46%         56763  LFB or LFB hit
  14.90%          6574  Local RAM or RAM hit
   6.79%         16061  L1 or L1 hit
   2.02%          3143  L3 or L3 hit
   1.45%          2216  L2 or L2 hit
   1.28%          2671  L3 miss
   0.07%           172  Uncached or N/A hit
   0.03%            91  I/O or N/A hit
```

```
Samples: 96K of event 'cpu/mem-stores/P', Event count (approx.): 96468
Overhead       Samples  Memory access
  82.53%         79613  L1 hit
  17.47%         16855  L1 miss
```

Profiling Guide (4)

- **Profiling strategy**
  1. Select the events you want to track.
     1. Refer to "perf list".
  2. For each thread count(1, 2, 4, 8),
     1. Run perf-stat.
     2. Run perf-mem.
  3. Gather the data and analyze the reason for the speed-up.

Profiling Guide (5)

- **NOTICE! (Trouble-shooting)**
  - When running "perf-stat"/"perf-mem" with different BLAS num_threads
    - Change Linux user to superuser(su)
      - sudo su
    - Change the number of OpenMP threads
      - export OMP_NUM_THREADS=n
  - python import error while running "perf-stat/perf-mem"
    - Don’t forget to use python of anaconda environment!
      - conda activate proj5 & which python
      - ex) /home/jisung/anaconda3/envs/proj5/bin/python
    - perf stat -e … <u>/home/jisung/anaconda3/envs/proj5/bin/python</u> run.py (o)
    - perf stat -e … python run.py (x)
    - perf mem record <u>/home/jisung/anaconda3/envs/proj5/bin/python</u> run.py (o)
    - perf mem record python run.py (x)

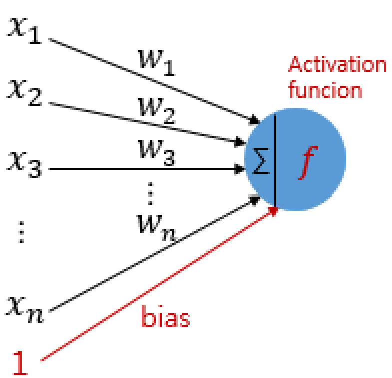
Profiling Guide (6)

- **NOTICE! In problem 3,**
  - You need to estimate the event counts for executing code region "line 22~".
  - To do so,
    - ① Run run.py and estimate the event counts.
    - ② Run prepare.py and estimate the event counts.
    - ③ Substract the result of ① to the result of ②.

Appendix

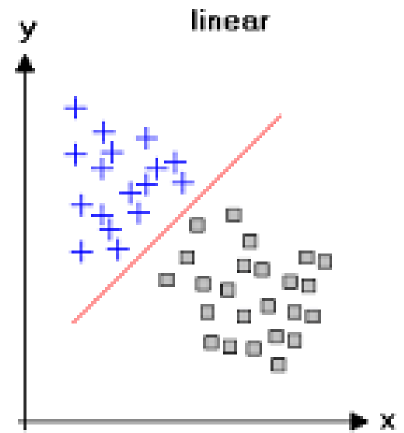
MLP (1)

- **Perceptron**
  - Classify, or find the answer based on n-dimensional input.
  - Single-layer perceptron is sufficient only for linear cases.
  - Training: weight & bias are determined in a way that minimizes cost.
    - Cost function: J(θ) = Σ{f(X,θ) - Y}^{2} (θ is a set of weight & bias)
    - Train by gradient descent learning
  - y = w·x + b: vector-vector / vector-matrix computation



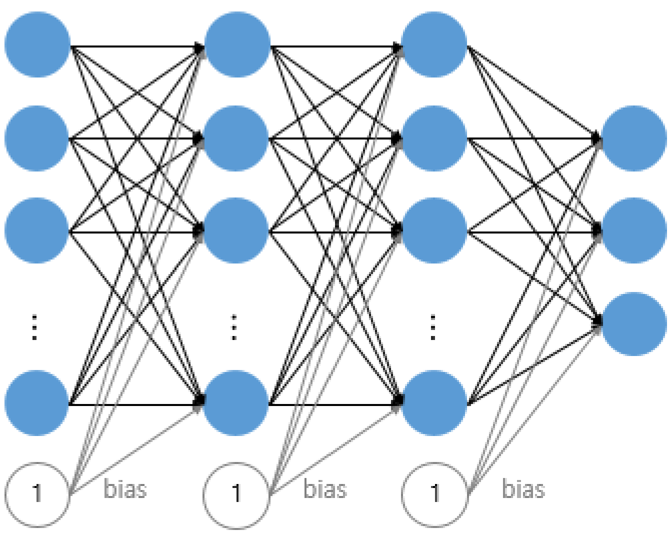
$$\sum_i^n w_i x_i + b \geq 0 \rightarrow y = 1$$

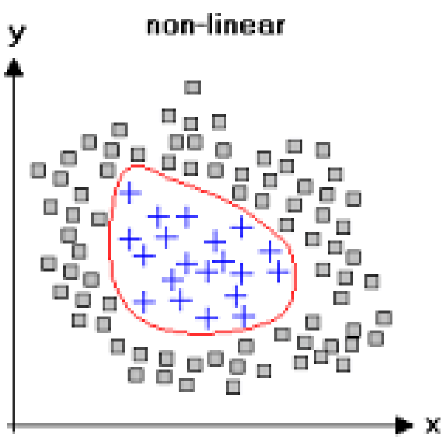
$$\sum_i^n w_i x_i + b < 0 \rightarrow y = 0$$



MLP (2)

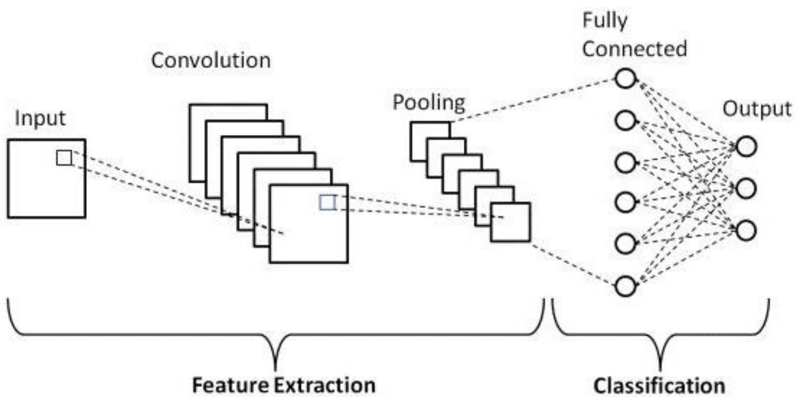
- **MLP(Multi-Layer Perceptron)**
  - a.k.a. FC (fully-connected) layer
  - To solve a non-linear problem, single-layer perceptron is not sufficient.
  - MLP can find answer for non-linear cases.
  - Each layer corresponds to GEMV/GEMM.

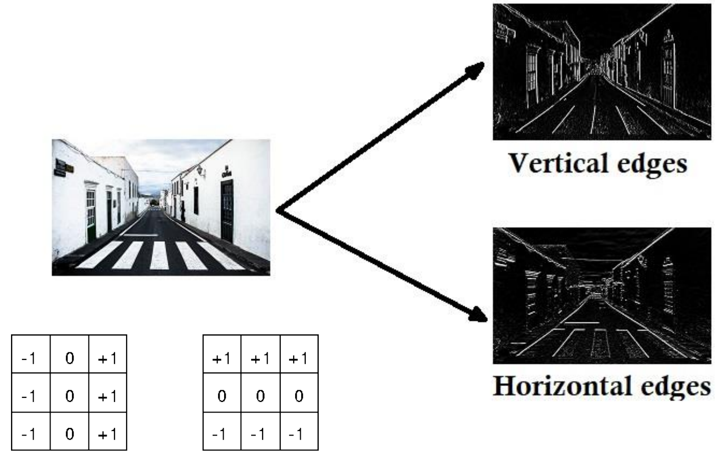




CNN

- **CNN(Convolution Neural Network)**
  - What happens if we use MLP on image classification…?
    - Image: 2-dimensional / video: 3-dimensional (x3: RGB) → requires massive computation
  - CNN was first introduced by Yann LeCun, in 2014.
    - CNN improved classification performance & reduced calculation overhead.
  - CNN consists of 2 phases: feature extraction & classification.
    - Feature extraction: convolution(=pattern recognition)+pooling(=representative value selection)
    - Classification: MLP



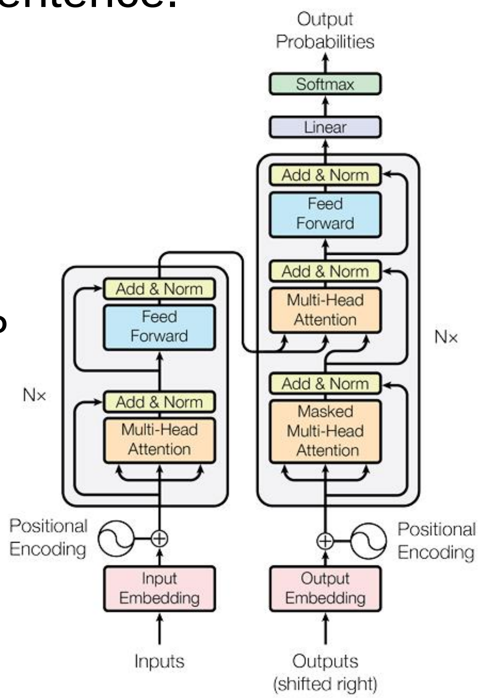


Seq2seq Model

- **Seq2seq model: $\mathbf{X_{1:n} \rightarrow Y_{1:m}}$**
  - $p(Y_{1:m}|X_{1:n}) = p(Y_1|X_{1:n}) \cdot p(Y_2|X_{1:n}, Y_1) \cdot \ldots \cdot p(Y_m|X_{1:n}, Y_{1:m-1})$ (by the Bayes' rule)
    - Each elements in output sequence Y was chosen to maximize $p(Y_{1:m}|X_{1:n})$.
  - Composed of "encoder" and "decoder".
  - RNN-based, LSTM-based, transformer, etc.
- **Encoder**
  - $f_{\theta_{enc}} : X_{1:n} \rightarrow \mathbf{X}_{1:n}$
  - Encoder maps the input sequence to a contextualized encoding sequence.
- **Decoder**
  - Decoder calculates the most probable next token.
  - Decoder generates token one-by-one autoregressively until it meets <EOS>.
    - <EOS>: End-Of-Sentence (token)

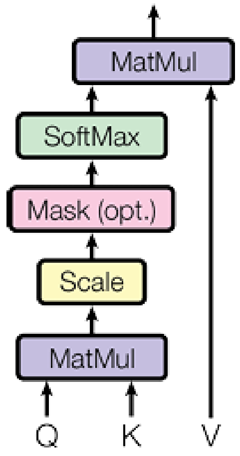
Transformer & Attention algorithm (1)

- **About "transformer"**
  - Vaswani et al., Attention is All You Need
  - Transformer uses "attention" layer.
- **Self-Attention**
  - Each element attends to every other element in the same sentence.
  - Consider as calculating relations among each elements.
    - ex) The <u>pizza</u> came out of the <u>oven</u> and <u>it</u> tasted good!
  - Encoder
    - contextual representation of input vector using attention & MLP
  - Decoder
    - <u>masked</u> attention: attention only for previous tokens

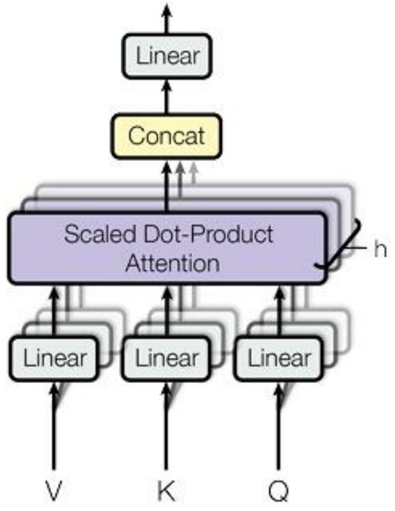


Transformer & Attention algorithm (2)

- **Attention layer in detail**
  - Scaled dot-product attention
    - Attention(Q, K, V) = $\text{softmax}(Q \cdot K^T/\sqrt{d_k}) \cdot V$
    - Query(Q): representation of input tokens ($Q=W_Q \cdot X$)
    - Key(K): representation of previous tokens + input tokens ($K=W_K \cdot X$)
    - Value(V): representation of previous tokens + input tokens ($V=W_V \cdot X$)
    - K & V can be used → use KV cache to reduce amount of calculation

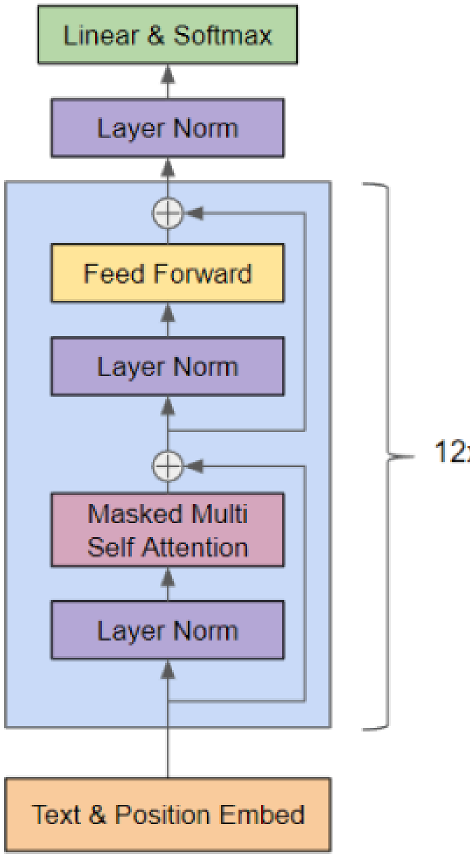


  - Multi-head attention
    - Easy to capture multiple relationships simultaneously.
    - Easy to apply parallel computation.
    - In GPT-2, MHA layer consists of 12 heads.



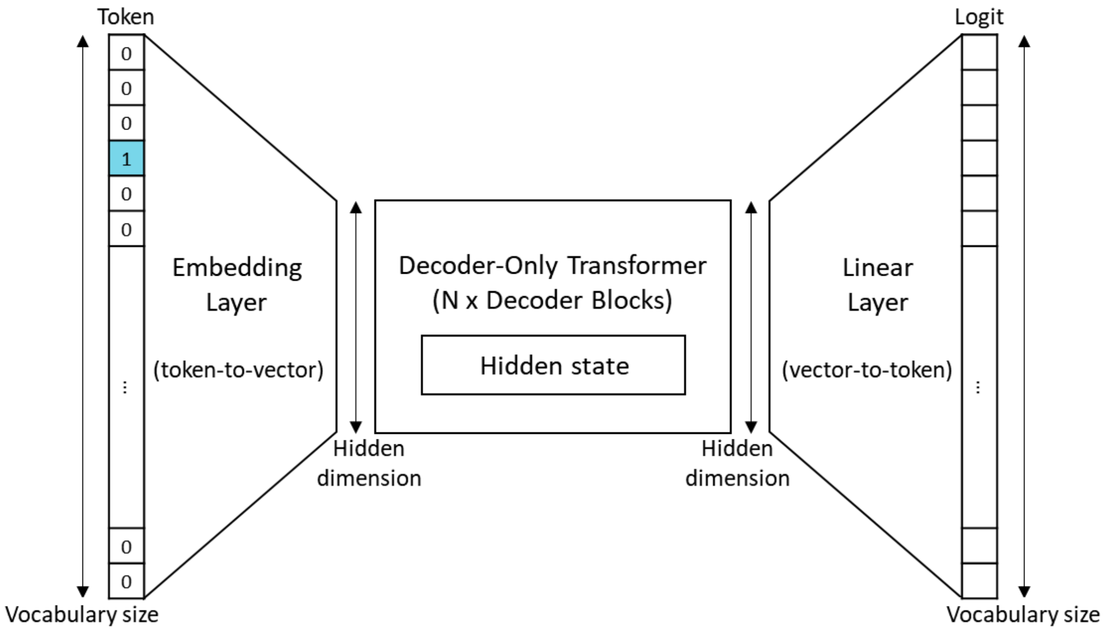
GPT-2 Model Architecture (1)

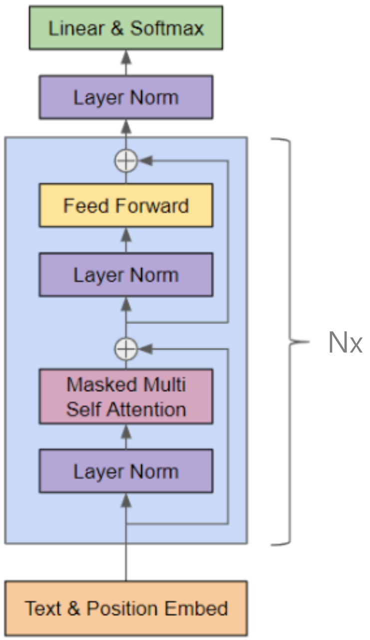
- **GPT-2: Based on decoder-only transformer**
  - 1) Tokenizer
    - Text Embedding: mapping tokens to integer values
    - Position Embedding: adding position information to each token
  - 2) Decoder blocks
    - Masked multi-head self attention
      - to prevent attention with future tokens
    - MLP
    - Layer normalization: X~N(μ, σ)=(X- μ)/σ (normal distribution)
      - adjusting values measured on different scales to a common scale
  - 3) MLP
    - Calculate probabilities of each token being on next token
    - Select the most probable token and use it as new input token



GPT-2 Model Architecture (2)

- **Tokenizer**: text ↔ token IDs
- **LLM**: token ID (+hidden state) → Logit
  - $logit(A) = log\left(\frac{P(A)}{1-P(A)}\right)$ → P(A) = sigmoid(logit(A))

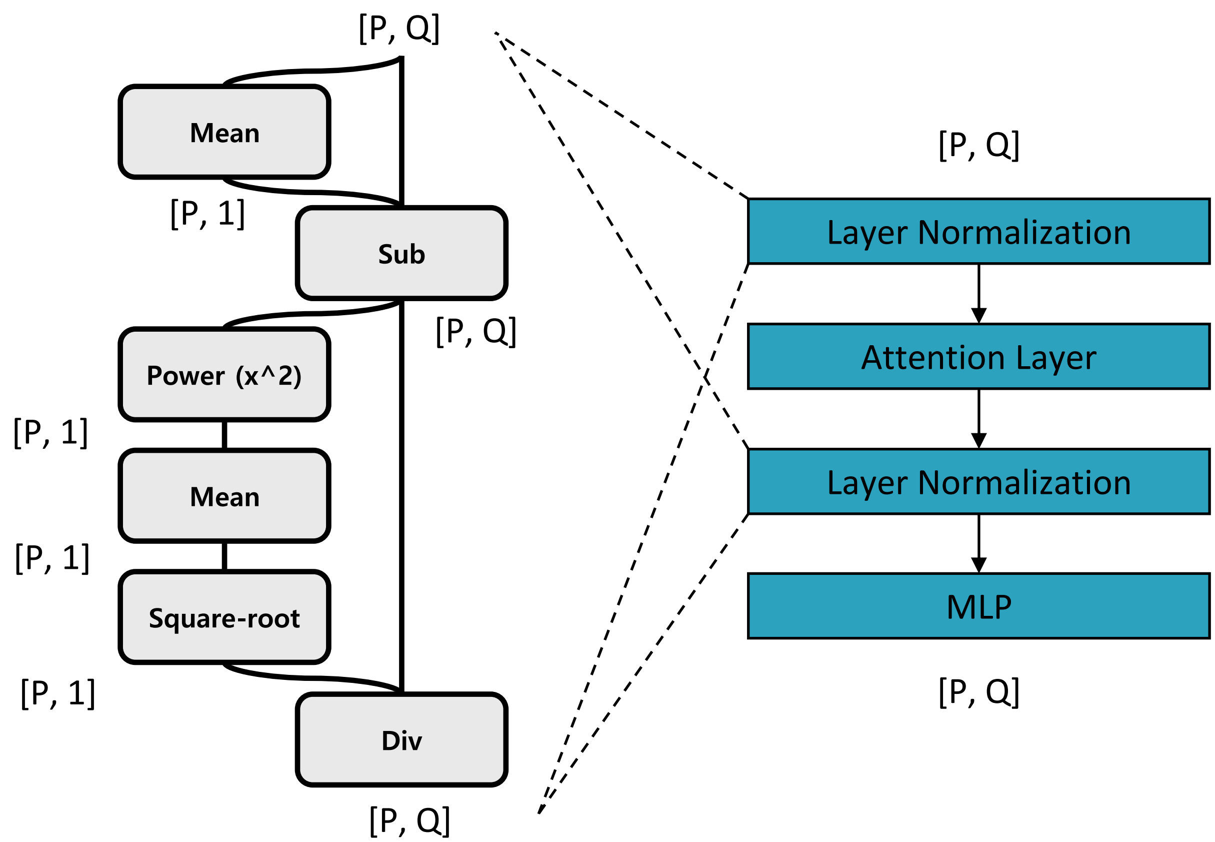




Decoder Block Breakdown (1)

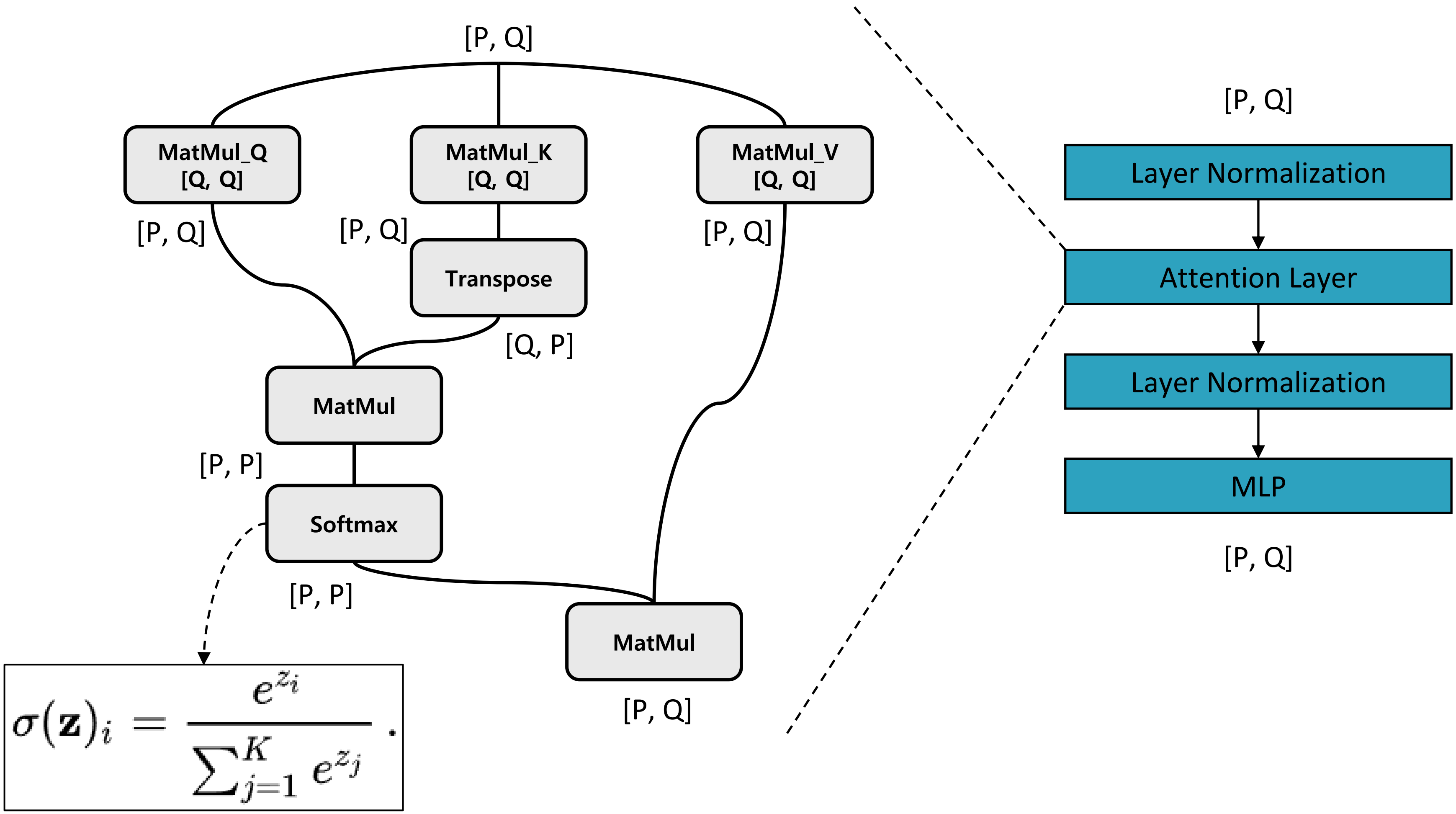
- **[P=sequence_length, Q=hidden_dim]**
  - sequence length: number of tokens came through the model

$$y = \frac{x - \mu}{\sqrt{\sigma^2}}$$



Decoder Block Breakdown (2)

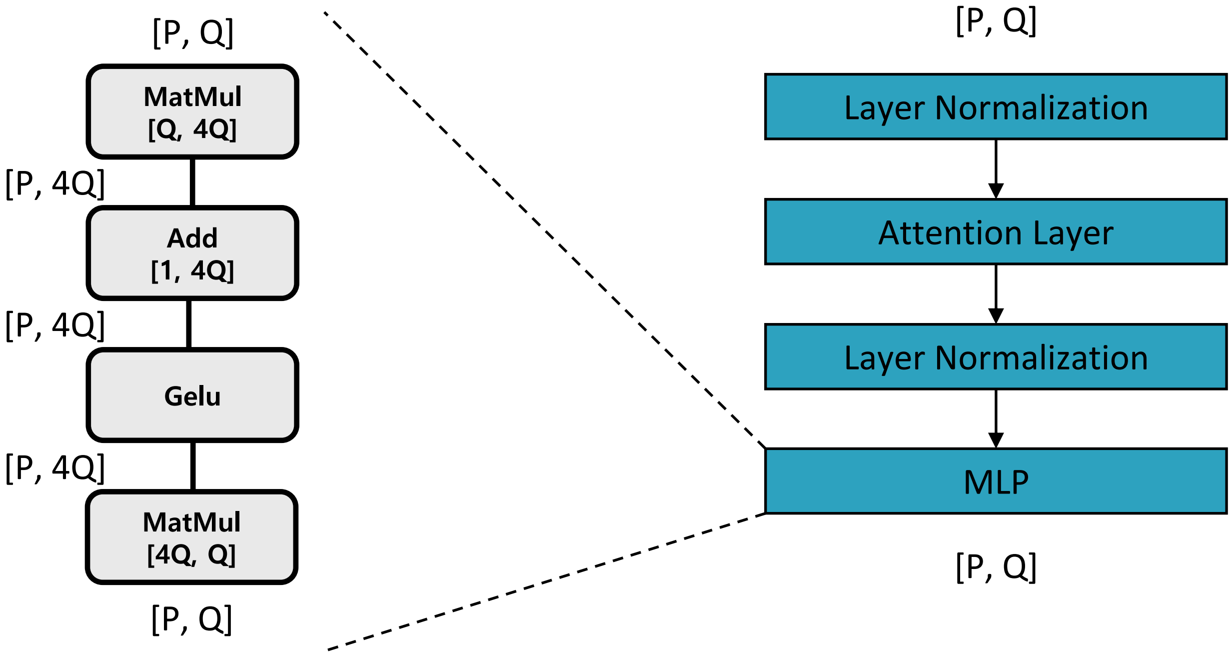
- **[P=sequence_length, Q=hidden_dim]**
  - sequence length: number of tokens came through the model



$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} .$$

Decoder Block Breakdown (3)

- **[P=sequence_length, Q=hidden_dim]**
  - sequence length: number of tokens came through the model
  - Gelu: activation function



Q/A