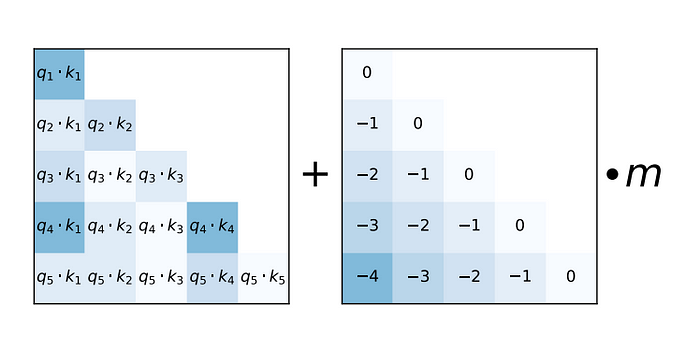
<https://blog.gopenai.com/how-to-speed-up-llms-and-use-100k-context-window-all-tricks-in-one-place-ffd40577b4c>

* Context Length가 중요한 이유
  + Large Context Length를 가지고 있으면 모델을 추가학습 할 필요 없이 “dropping a large pile of custom data into an LLM”이 가능하고 개인화된 답변을 얻을 수 있다.
* 큰 context length를 처리할 때 기존 트랜스포머 구조의 한계
  + 트랜스포머의 모든 학습 가능한 weight 행렬들은 input token의 길이와 무관하다. 따라서 2K context length로 학습을 하면 input token의 길이와 상관없이 추론에 사용 가능하다.
  + 하지만 더 긴 context length에서 학습되지 않았으므로 100K 입력에서는 의미없는 결과를 생성할 것이다. 이 말은 train data의 distribution이 inference할 때와 멀어진 다는 뜻이다.
  + 한가지 해결책은 2K 토큰으로 pre-train 하고 난 뒤 더 긴 contexts로 fine-tuning 하는 방법이다.(MosaicML이 한 방법) 해당 방법은 기존의 트랜스포머 구조에서는 적용이 불가능하다. 추가적인 기법 적용이 필요(Trick #1)
* 트랜스포머 구조의 계산 복잡도
  + **Q —**queries, **K** — keys and **V** — values, notations from the paper relating to the information retrieval, where you insert a “query” to the system and search the closest “key”  
    **n** —the input number of tokens  
    **d** — text embedding dimension  
    **h** — the number of attention heads  
    **k**— linear projection size for Q and K  
    **v** — linear projection size for V
  + Large context length의 계산 복잡도 또한 트랜스포머 구조에서 문제가 된다.
  + 계산을 간소화하기 위해 k\*h=O(d)로 가정
  + Q, K, V로의 Linear Projection은 O(nd2)의 복잡도
  + Q와 K의 곱과 V와의 곱은 O(n2d)의 복잡도를 가짐
  + 그러므로 어텐션 레이어는 **O(n²d + nd²)** 의 복잡도를 가진다. 입력의 길이에 대해 이차 함수이고 또한 차원 크기에 대해서도 이차함수이다.
  + The term O(nd²) is important when d > n (for example, in LLaMa, n=2K and d=4K).
  + The term O(n²d) is important when n > d (for example, training MosaicML with n=65K and d=4K).
  + 예를 들어 65B의 LLaMa 모델, 2K 입력 길이, 4K 임베딩 크기를 학습시키기 위해서 대략 $3M가 필요하다. 입력 길이를 100K로 늘려 학습하는 경우 비용이 50배 증가한다. (입력 길이가 늘어났으므로 iteration은 줄어들지만 각 iteration마다 걸리는 시간이 증가한다.)
    - *(1) Complexity for****n****~****M \* (n²d + nd²)****(2)**Complexity for****p\*n****~****M/p \* ((p\*n)²d + (p\*n)d²)***
    - *(2)/(1) ~****(d + p\*n)/(d + n)***
* 학습과 추론때의 차이점 존재
  + 학습할 때는 병렬로 돌릴 수 있지만, 추론 때의 텍스트 생성은 순차적으로 생성해야한다. 따라서 학습과 추론 둘 다에 적용되는 최적화 기법도 있고 각각 적용되는 최적화 기법도 존재한다.
* 트랜스포머 속도 개선 및 context length를 늘리기 위한 기법
  + 구조 개선

1. 더 나은 Positional encoding ALiBi: 기존의 Positional Sinusoidal Encoding은 Extrapolation 능력(학습데이터를 기반으로 없는 데이터의 결과를 추론)이 취약함. 그 말은 학습에서 사용된 입력 길이보다 더 긴 입력을 인퍼런스와 파인튜닝에서 사용할 수 없다는 뜻이 된다.

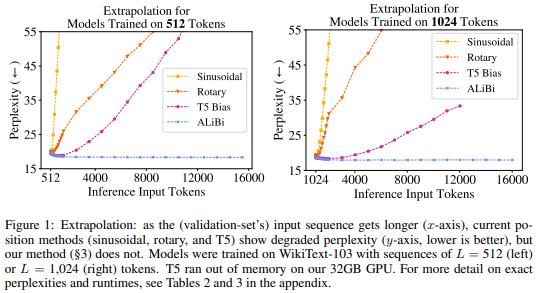
그러므로 이를 대체할 position embedding을 사용한다. – Attention with Linear Biases(ALiBI)는 Sinusoidal과 다르게 네트워크의 시작점에서 적용되지 않고 attention head에 적용되고, QK attention score에 그들의 거리와 비례하는 편향을 추가한다.

https://miro.medium.com/v2/resize:fit:700/1*l90IOO5Se6Db60K6-EASZg.png



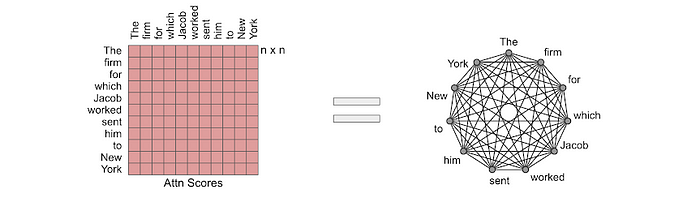
모델이 n개의 head를 가지고 있으면 2-8/n으로 시작한다. 예를 들어 8개의 head가 있는 경우 1/2로 시작해 1/22, 1/23 …., 1/28 이런 식으로 적용된다.

이렇게 구한 slope는 여러 도메인, 여러 모델사이즈에서 다양하게 적용할 수 있으며, 저자들은 새로운 모델에 따라 slope를 구할 때 튜닝이 필요하지 않다고 믿는다.

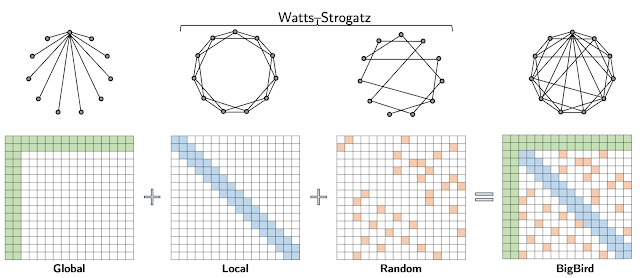


1. **Sparse Attention**: 100K의 토큰이 입력으로 주어질 때 모든 토큰이 서로 관련이 있다고 볼 수 없다. 이러한 계산 복잡도를 줄이기 위해서 어텐션 스코어를 계산할 때 일부 토큰만 고려하는 방법이 있다. 해당 방법은 학습과 추론 둘 다의 속도를 개선해준다.

이러한 방법을 통해 계산 복잡도를 n의 제곱이 아닌 n에 선형적으로 비례할 수 있게 sparsity를 추가해 주는 것.



[Full attention의 경우]



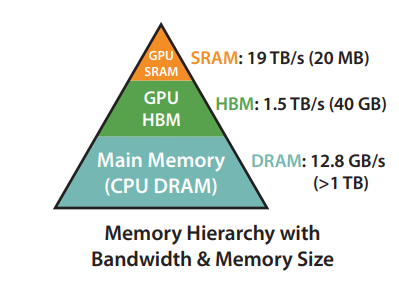
[Sparse Attention Method]

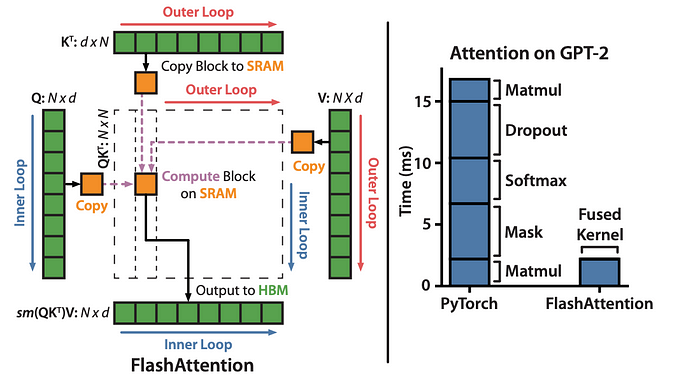
Sliding Window Attention(also called as Local)의 경우 윈도우의 크기 w(각 토큰마다 양옆의 w/2개의 토큰들을 어텐션 스코어 계산에 사용한다. 그러면 계산 복잡도는 O(n\*w)가 되고 입력 크기에 대해 선형적 복잡도를 가지게 된다. (효율을 위해 w는 n보다 작아야 함)

BigBird Attention(method that combines global, local, random mechanisms)

계산하는 유사성 점수의 개수와 다른 노드 간의 정보 흐름간의 관계가 존재한다.

1. **FlashAttention**: attention layer에서 계산하기 위한 중간 결과의 크기는 (n x n)으로 매우 크다. 이를 계산하려면 중간 결과 저장과 계산을 하기 위해 HBM(large but slower)과 SRAM(small but faster)를 왔다 갔다 해야 한다. 속도의 차이와 크기에 따라서 bottleneck이 발생한다. FlashAttention은 학습과 추론의 속도를 향상시켜준다.



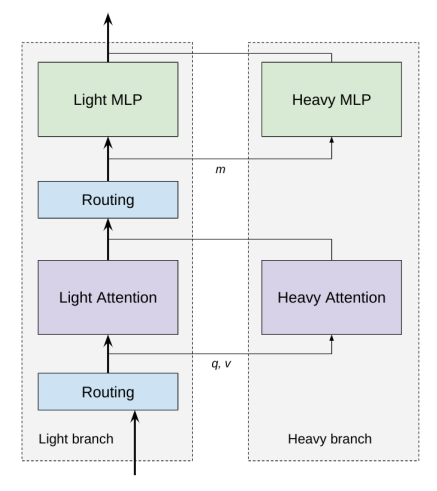


FlashAttention 알고리즘은 Q, K, V 행렬들을 블록으로 나눠서 SRAM으로 로딩한 뒤, 해당 블록에 대한 어텐션 결과를 계산해 최종 결과 값을 SRAM에 저장한다. 이러한 프로시져를 **tiling** 이라고 부른다.

Pytorch 2.0에는 flash-attention이 내장되어 있다.

1. **Multi-Query attention(MQA)**: 추론 속도를 향상시켜준다. 기존의 Multi-Head Attention은 각 K와 V 행렬마다 다른 선형 레이어를 가진다. MQA는 K와 V를 선형 투영할 때 동일한 가중치를 모든 attention heads와 공유해 사용하는 방법이다. 따라서 (n, k)와 (n, v) 크기의 가중치 행렬 두개만 둔다. (잘 모르겠다…)
2. **Conditonal Computation**: 입력 길이가 임베딩 크기보다 적으면 feedforward와 선영투영 시 bottleneck이 있다. 이를 해결하기 위한 일반적인 방법은 모델 모든 파라미터를 중요한 토큰에만 전부 적용하고, 모든 토큰은 일부 파라미터만 적용하는 방법이다. 해당 방법은 64K 와 같이 긴 입력에도 정확도가 속도가 좋았다.

다음 그림과 같이 Light와 Heavy 두 브랜치를 나눠서 적용하는 방법이다. Light 레이어는 모든 토큰에 적용되고, Heavy 레이어는 중요한 일부 토큰에만 적용된다.



1. Large RAM GPUS: 필수는 아니지만 large context를 로드하기 위해서 많은 RAM이 필요하다. (80GB A100 GPUs)

요약

Context Length가 정해져 있는 이유는 해당 길이로 학습되었기 때문이다. 더욱 긴 Context Length를 입력으로 받으려면 Fine-Tuning이 필요하다. 이를 위해서는 기존 트랜스포머의 Positional Sinusidal Embedding의 한계가 존재하기 때문에 다른 Positinal embedding, ALiBi를 사용해야 한다.

나머지는 학습과 추론 속도를 높이기 위한 방법론들이다.