KAIST

특 기 입 증 자 료 서 식

【특기자전형】

Korea Advanced Institute of Science & Technology

순번	자료명	자료유형 ①수상내역 ②연구논문/보고서 ③프로젝트/창작물 ④특허/지식재산권 ⑤기타 중 1개 선택	활동기간	개인·공동실적 여부	내용 요약	페이지
1	SW 동행 데모데이 우수상 & 특별상	1	2024.09. ~ 2024.11.	☐ 개인 ☑ 공동(팀원 수: 8명, 팀 내 자신의 역할:인공지능 개발)	해안 그물 쓰레기 수거를 위해 YOLO 파인튜닝 & 자체 개발 옵티마이저 사용	2~19
2	Pascal Biased Attention 프로젝트 보고서	2	2025.05. ~ 2025.08.	☑ 개인 ☐ 공동(팀원 수: 명, 팀 내 자신의 역할:)	attention 학습효율성을 올리기 위해 파스칼 피라미드 사용	2~23
3	NS기반 옵티마이저 프로젝트 보고서	2	2024.07. ~ 2025.04.	☑ 개인 ☐ 공동(팀원 수: 명, 팀 내 자신의 역할:)	나비에 스토크스 방정식을 사용하여 옵티마이저의 성능을 향상시킴	2~42
4	SW영재 창작대회 우수상	1	2023.05. ~ 2023.11.	☐ 개인 ☑ 공동(팀원 수: 4명, 팀 내 자신의 역할: 알고리즘 설계 및 코딩)	아동의 스미싱문제를 해결할수 있는 프로그램을 창작함	2~12
5	차원의 저주 해결 창작물	3	2025.09. ~ 2025.09.	☑ 개인 ☐ 공동(팀원 수: 명, 팀 내 자신의 역할:)	차원의 저주를 해결하기 위한 저만의 생각입니다.	2~3

파스칼 피라미드 편향을 활용한 효율적인 언어 모델 학습

민주호

현대 언어 모델은 트랜스포머 구조를 기반으로 높은 성능을 달성하지만, 학습 및 추론 과정에서 높은 전력 소모와 탄소 배출로 인해 지속 가능성에 한계가 있습니다. 본 연구는 파스칼 피라미드 구조를 어텐션 메커니즘에 편향으로 주입하여 모델 파라미터 수를 줄이고 학습 시간을 단축하며, 전기 사용량과 탄소 배출을 최소화하는 효율적인 언어 모델을 제안합니다. 이를 위해 파스칼 피라미드 기반의 편향을 설계하고, WikiText-103 데이터 세트를 활용하여 실험을 수행하였습니다. 결과적으로 제안된 모델은 기존 트랜스포머 대비 유사한 성능을 유지하면서 학습 시간과 자원 소모를 감소시켰습니다. 이는 지속 가능한 인공지능 개발에 기여할 수 있음을 시사합니다.

현대 인공지능 언어 모델은 google 트랜스포머 구조를 기반으로 높은 성능을 달성하고 있습니다. 그러나 이러한 모델은 대규모 파라미터와 복잡한 연산으로 인해 학습 및 추론 과정에서 상당한 전력 소모와 탄소 배출을 초래합니다. 예를 들어, 대규모 언어 모델의 학습은 수십만 킬로와트시(kWh)의 전력을 소비하며, 이는 환경 지속 가능성에 부정적인 영향을 미칩니다. 이에 따라, 성능 저하 없이 자원 효율성을 높이는 새로운 방안이 필요합니다.

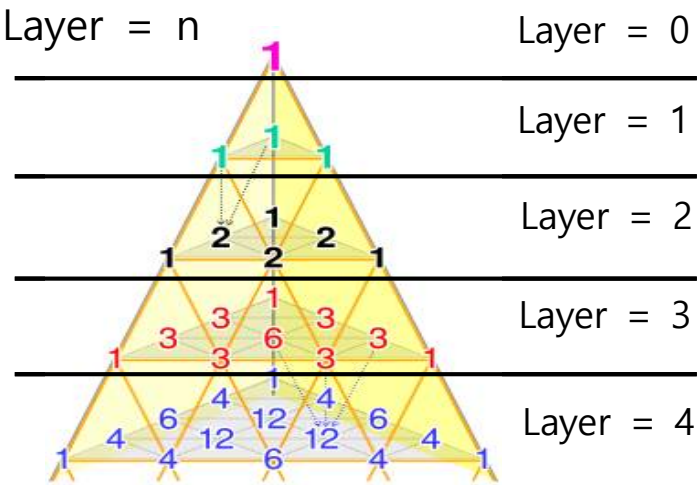
트랜스포머의 효율성을 높이기 위해 다양한 방법들이 제안되었습니다. 예를 들어, 스파스 어텐션을 활용해 계산 복잡도를 줄였으며, 메모리 효율적인 어텐션 메커니즘을 제안한 연구도 있습니다. 그러나 이러한 방법들은 여전히 높은 하드웨어 요구 사항을 필요로 하며, 환경적 지속 가능성 문제를 완전히 해결하지 못합니다.

본 연구는 파스칼 피라미드 구조를 어텐션 메커니즘에 편향으로 주입하여, 모델의 파라미터 수를 줄이고 학습 시간을 단축하는 동시에 성능을 유지하는 언어 모델을 개발하는 것을 목표로 합니다. 이를 통해 전기 사용량과 탄소 배출을 줄여 지속할 수 있는 인공지능 모델을 구현하고자 합니다.

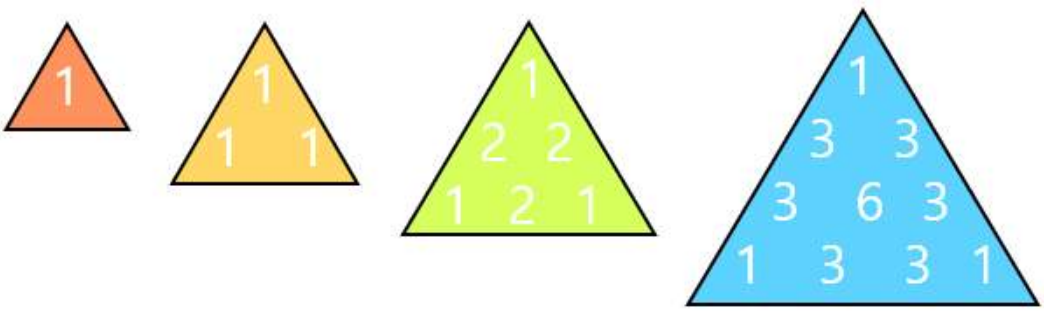
본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

파스칼 피라미드

파스칼 피라미드는 파스칼 삼각형의 변수를 하나 더 추가한 $(x+y+z)^n$의 전개 항의 계수를 나타낸 것입니다. 즉 조합에서의 경우의 수가 되는 것입니다. 이를 층별로 나눈 것을 layer라고 명칭 하겠습니다.



아래는 layer=0~3까지 나타낸 사진입니다.



비록 이 실험에는 layer=3을 사용했지만, 실제 적용 시는 layer 값이 100, 200씩 커질 수도 있습니다. 하지만 이를 계산하는데 필요한 연산량은 3^n 이고 또한 변수를 증가시킬 때마다 연산량은 기하급수적으로 증가하여 오히려 비효율적일 수 있습니다. 이를 파스칼 피라미드의 규칙을 활용하여 해결할 수 있고, n^3으로 연산량을 줄일 수 있습니다.

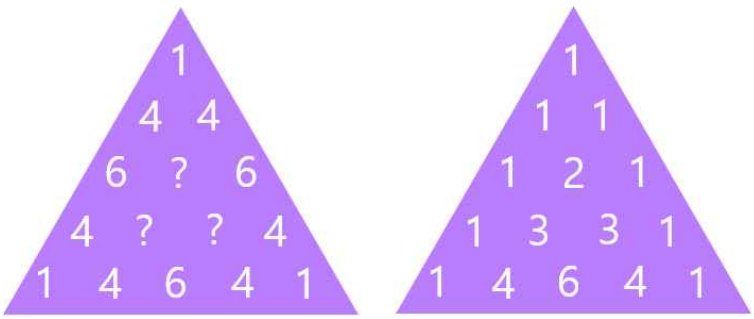
우선 파스칼 삼각형을 생각해보면

```
      1
    1 1
  1 2 1
1 3 3 1
```

이는 결국 $(x+y)^n$전개 항의 개수이고 이를 다시 생각해보면 x와 y가 상호 작용을 주고받아 나오는 경우의 수입니다 즉, $(x+y+z)^n$의 전개 항은 x와 y와 z가 서로 복합적으로 상호작용을 주고 받는 경우의 수입니다. 다시 말하면 파스칼 삼각형 3개를 연결해 놓으면 됩니다.

그럼 우리가 아는 파스칼 피라미드 형태가 나오는데 여기서 문제점은 $(x+y+z)^n$의 항에서 중앙값을 알 수가 없다는 것입니다. 이를 해결하는 규칙을 연구해보니 파스칼 피라미드에서 파스칼 삼각형을 겹친 후 한 번 더 모서리에 있는 숫자를 곱한 형태로 나왔습니다.

예를 들어 layer 4를 보겠습니다.



왼쪽은 파스칼 피라미드이며 오른쪽은 파스칼 삼각형입니다. 그리고 중앙에 ?는 우선 모른다고 하겠습니다. 그럼 두 삼각형을 겹치고 가장자리의 수들과 곱하면 ?의 수를 알 수 있습니다. 즉 ?의 수는 위에서부터 6×2=12, 4×3=12, 4×3=12임을 알 수 있습니다. 즉, 다시 수식으로 풀어서 적으면

layer = n일 때 두 이항계수의 곱 $C(n,r)\times C(r,c)$과 같다는 것을 알 수 있습니다.
이를 증명해보면

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

증명

삼항계수 $(x+y+z)^n$를 전개했을 때, $x^i\ y^j\ z^k$ (단, $i+j+k=n$) 이고 항의 계수는 $\frac{n!}{i!\,j!\,k!}$이며, 이를 $C(n:i,j,k)$로 표기해보겠습니다.
그리고 제가 찾은 규칙을 다른 식으로 표현한다면 $C(n,r)\times C(r,c)$으로 표현이 됩니다.
r: 피라미드 n 층 내부의 줄(row) 번호를 의미합니다. (맨 위 꼭짓점 줄을 r=0으로 시작).
c: r번째 줄 내부에서의 위치(column) 번호를 의미합니다. (가장 왼쪽을 c=0으로 시작).

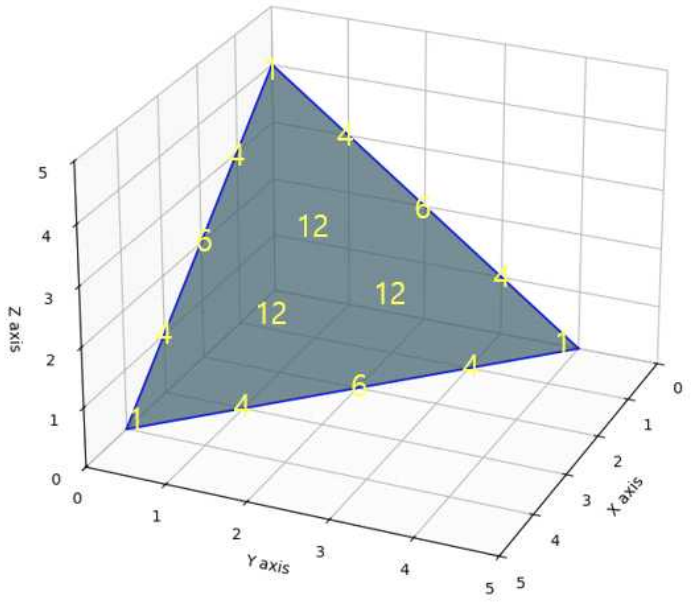
이라고 가정을 한 뒤 (I, j, k)와 (r, c)와의 관계를 봐봤습니다.
i = n - r이라고 하면 j = c, k = r - c의 관계를 가집니다.

이를 다시 항의 계수로 표현하면 $\frac{n!}{(n-r)!\,c!\,(r-c)!}=C(n,r)\times C(r,c)$이고, 이를 만족하는 것을 조사하면 됩니다.
우변을 팩토리얼의 정의에 따라 풀어주면 $\frac{n!}{r!(n-r)!}\times\frac{r!}{c!(r-c)!}$ 이고
r! 을 약분하여 소거해주면 $\frac{n!}{(n-r)!\,c!\,(r-c)!}$으로 좌변과 같다는 것을 알 수 있습니다.

파스칼 피라미드의 해석 및 사용

파스칼 피라미드는 변수가 4개 즉, $(x+y+z)^n$입니다. 이를 다양하게 해석을 할 수 있습니다.
저의 경우는 처음에 벡터로 해석했습니다. 만일 n = 4일 때라고 가정하면 x^2yz, xy^3 등 변수가 4개 곱해진 전개 항의 개수가 나옴을 보고 이를 공간상에 나타나는 위치의 경우의 수라고 생각했습니다. 즉, 다른 표현으로는 n번 양의방향으로 x y z 중 한 칸식 이동했을 때의 이동할 수 있는 경우의 수입니다.

예를 들어 x^2yz 같은 경우 아래 사진을 참고하면 x 성분이 2개 y 성분이 1개 z 성분이 1개이므로 (0, 0, 0)에서 (2, 1, 1)으로 가는 경우의 수는 12입니다.



이러한 성질과 더불어 비유클리드 기하학인 택시 기하학으로 해석해 보았습니다. 이 경우 위의 그래프를 참고하여 layer = n의 값을 많이 증가시키면 이는 곧 택시 기하학에서 $\frac{1}{8}$ 택시 구로 해석이 됩니다. 단, 이럴 때 그래프의 간격을 0으로 수렴시키는데 여기서 차분법의 개념을 생각했습니다.
차분법은 컴퓨터가 미분의 정의를 가지고 미분하지 못하기 때문에 $\lim_{h\to 0.01}\frac{f(x+h)-f(x)}{h}$ 과같이 근사시켜서 계산하는 것입니다.

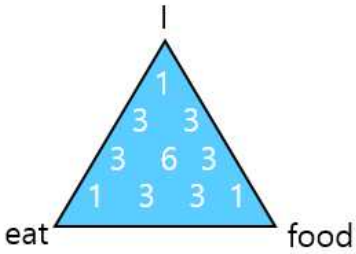
이처럼 n을 100, 200과 같이 큰 수로 두고 좌표평면을 축소한 관점에서 바라보면 연속적인 확률값은 얻을 수 없지만, 근사적인 해는 구할 수 있다고 생각했습니다.
처음에는 이러한 생각을 가지고 양자 수준에서의 역학에 대한 계산 또는 물리학 관점에서 인공지능의 학습을 시킬 수 있을 것이라 생각했지만 저의 지식의 부재로 인해 타당한 근거를 찾기는 어려웠습니다.

따라서 이 방법 외에 다른 활용방안을 고안해 봤습니다. 다시 정의와 벡터의 관점으로 봤을 때 벡터라는 단어가 제 눈에 들어왔고 이는 인공지능 수학 시간 때 언어의 처리를 배우며 벡터를 활용하여 언어들의 유사도 및 여러 계산을 한다는 것에서 영감을 얻었습니다.

만약 언어를 벡터로 표현한 뒤 이들 언어 사이의 관계가 파스칼 피라미드의 확률값 또는 다변수 파스칼 피라미드의 확률값과 연관이 되어있다면 이를 활용하면 인공지능 학습을 효과적으로 빠르게 할 수 있고 기존 트랜스포머의 문제인 연산량 또한 해결할 수 있는 시초 선이 될 것으로 생각이 들어 이를 실험해봤습니다.

아래는 처음 제가 생각한 것입니다.

sentence: {I eat food} 라는 문장이 있으면 이를



정해진 layer의 파스칼 피라미드에 상응시켜주고 이를 순서대로

{I}, {eat}, {food}, {I eat}, {eat food}, {food I}, {I eat food}

처럼 생각한 뒤 파스칼 삼각형의 숫자들을 인공지능 모델이 적합하게 부여하면 인공지능 언어 모델은 단어와 단어를 더 효과적으로 잘 연결지을수 있을 것으로 생각했습니다. 마치 단어와 단어 사이에 응집소 응집원 같은 역할을 할 것이라 생각했습니다.

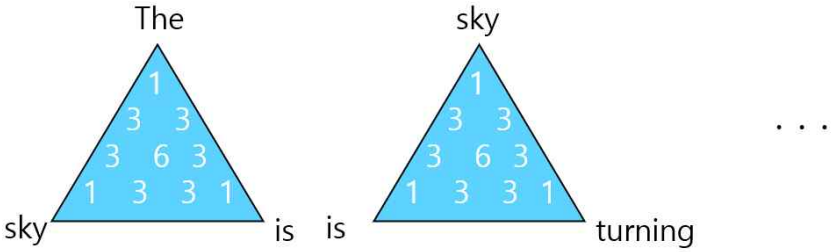
처음 모델은 위의 과정을 통해 학습시켰지만, valid_loss 값이 7.6에서 점점 더 올라가 9.2로 자꾸 올라가는 것을 보았습니다. 따라서 조기 종료한 뒤 어떤 점이 문제였는지 파악해봤습니다.

여기에는 3가지 문제점이 있었습니다.

첫 번째는 {I eat}, {eat food}의 경우에 부여하는 값과 {food I}와 같이 떨어진 단어에도 같은 값을 부여하는 것이 문제였습니다. 이를 해결하기 위해 떨어진 단어의 경우 페널티를 주어서 인공지능 모델이 직접 떨어진 단어에 대해 어느 정도로 중요하게 고려할 것인지 학습할 수 있도록 해주었습니다.

두 번째 문제는 {The sky is turning red.}처럼 단어가 3개로 구성이 안 되어있는 문장의 처리가 문제였습니다. 이 경우 sliding-winows 기법을 활용하여 3단어씩 문장을 쪼개주었습니다.

Window(W=3): {The sky is},{sky is turning},{turning red.}



이후 겹치는 단어들은 산술평균으로 계산했습니다.

이러한 실패의 과정을 거쳐 다시 학습을 진행했습니다. 그 결과 6000이터레이션 학습했더니

Iter	Train_L	Val_L	LR	Alpha	skip_F	Time(s)
500	4.3172	4.3148	1.50e-04	-2.0181	0.875550	1313.66
1000	3.1627	3.1511	3.00e-04	-1.9037	0.861556	1289.95
1500	1.4413	1.4508	3.00e-04	-1.7312	0.837715	1289.09
2000	0.6284	0.6272	3.00e-04	-1.5628	0.810943	1286.74
2500	0.4029	0.3991	3.00e-04	-1.4320	0.787540	1288.26
3000	0.3195	0.3220	3.00e-04	-1.3821	0.777984	1287.67
3500	0.2770	0.2762	3.00e-04	-1.3596	0.773548	1285.83
4000	0.2512	0.2497	2.99e-04	-1.3497	0.771587	1284.84
4500	0.2266	0.2281	2.99e-04	-1.3463	0.770893	1285.44
5000	0.2118	0.2121	2.99e-04	-1.3429	0.770207	1288.70
5500	0.1968	0.1980	2.99e-04	-1.3455	0.770744	1289.00
6000	0.1864	0.1865	2.98e-04	-1.3461	0.770854	1286.48

Starting generation with prompt: 'I think '

--- SAMPLE 1 ---

I think, by the bye, I can tell you that this is not a W ord, that we should.

--- SAMPLE 2 ---

I think it is a misfortune.

--- SAMPLE 3 ---

I think he was himself the man, the most lovely Sophia, and was, at the same time, so.

--- SAMPLE 4 ---

I think they will get the better of us.

--- SAMPLE 5 ---

I think. But I shall set the matter right in a moment, and in a few minutes I shall have.

이런 결과가 나왔습니다. 언어생성 능력은 있지만 PPL 측정 시 약 1.2로 말이 안 되는 수치가 나와서 잘못된 부분이 있는지 검토를 다시 해봤습니다. 그 결과 코드에서 데이터 스플릿을 잘못하여 검증데이터가 분리가 안 되어있었습니다. 또한 직접 수집한 데이터 세트를 사용하여 데이터의 양과 품질이 안 좋았었던 문제점도 있었습니다.
이를 해결하기 위해 WikiText-103을 사용했습니다. 하지만 이러한 시도에도 제 모델은 valid_loss 값이 7.4 미만으로 안 떨어졌습니다.
이는 파스칼 피라미드를 기반으로 한 편형값이 적절한 강도가 아닌 너무 큰 강도로 작용한다고 판단했습니다. 마치 일반적인 풀로 단어와 단어를 붙여주어야 하는데 이를 풀이 아닌 강력접착제로 붙였다고 생각했습니다.

이를 해결하기 위해 사람이 직접 하이퍼파라미터를 설정해도 되지만 그러면 무수한 테스트를 진행해야 하고 이는 오히려 테스트 시간이 더욱 길어지기 때문에 이는 오히려 비효율성이 증가할 것이라 생각했습니다.
따라서 모델이 점진적으로 편향의 강도를 적용할 수 있도록 해주었습니다.

파스칼 피라미드를 적용한 어텐션

기존 어텐션의 경우 $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$ 를 사용합니다. 그러나 이 방식은 모든 단어 쌍에 동일한 초기 조건을 부여하여 문맥적 관계나 지역적인 구조를 초기적으로 학습하여 학습 이터레이션 수가 많아 지고 이로인해 학습 시간이 길어진다는 점이 있습니다.
제가 연구한 어텐션의 경우 3항계수를 단순히 편향으로 넣어준 $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}} + B)V$ 이런 형태가 아닙니다.

편향을 $B_{r,c} = \log(1 + \binom{n}{r} \cdot \binom{r}{c})$ 이와같이 설계하고 적응형 바이어스 스케줄링을

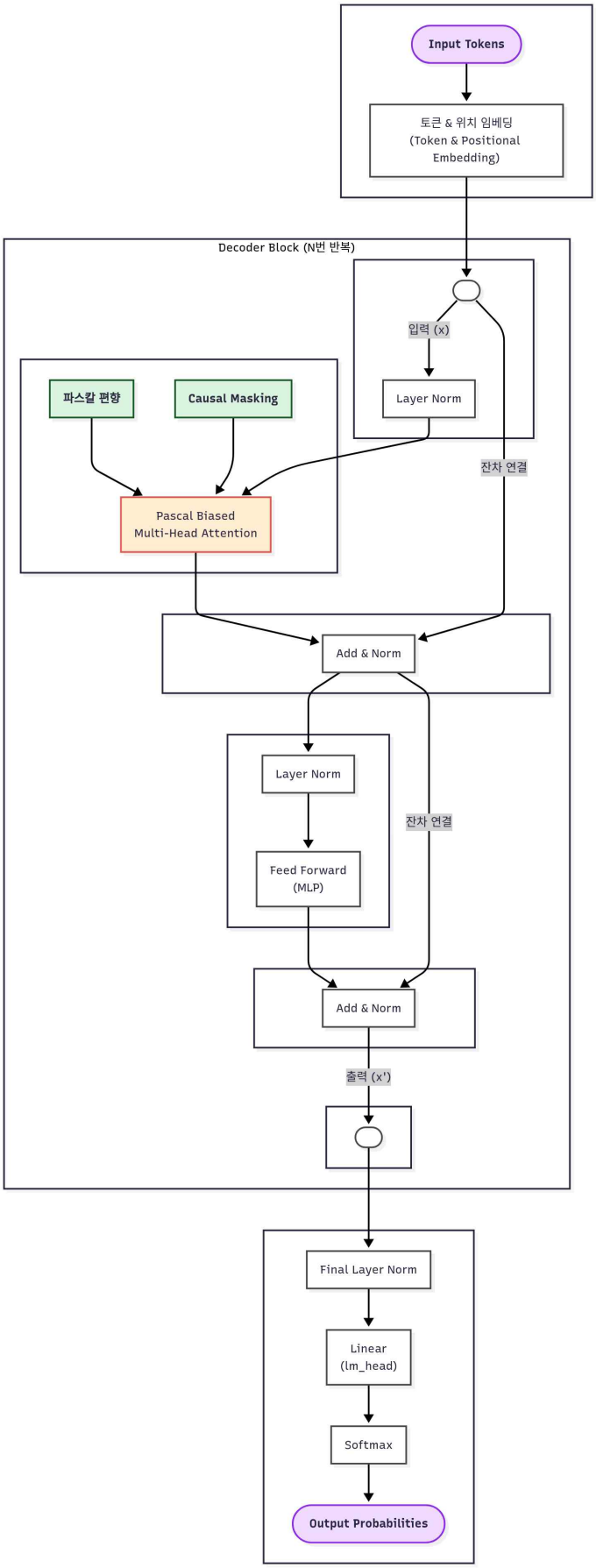
$BiasStrength(t) = \begin{cases} \frac{t}{T} & t < T \\ 1 & t \geqq T \end{cases}$ (T는 warm-up 스텝수)이와 같이 설계했습니다.

또한 skip-gram 패널티를 도입하여 일정 거리 이상 떨어진 토큰에는 편향 감소 계수를 곱하여 편향을 상쇄시켜 주었습니다. 이로 인해 가까운 정보는 강화한 상태로 학습이 진행이 되고 멀리 떨어진 정보는 조금더 약화된 상태로 학습이 가능했습니다.

$B'_{i,j} = B_{i,j} \cdot e^{-\exp(\theta)}$ 의 형태로 스킵 그램 패널티를 도입했습니다.

이를 전부 종합해보면, $Attention_{pascal}(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}} + \alpha(t) \cdot B'(\theta))V$

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.



제가 설계한 인공지능 구조입니다. 디코더 어텐션에서 디코더 파트를 사용했습니다.

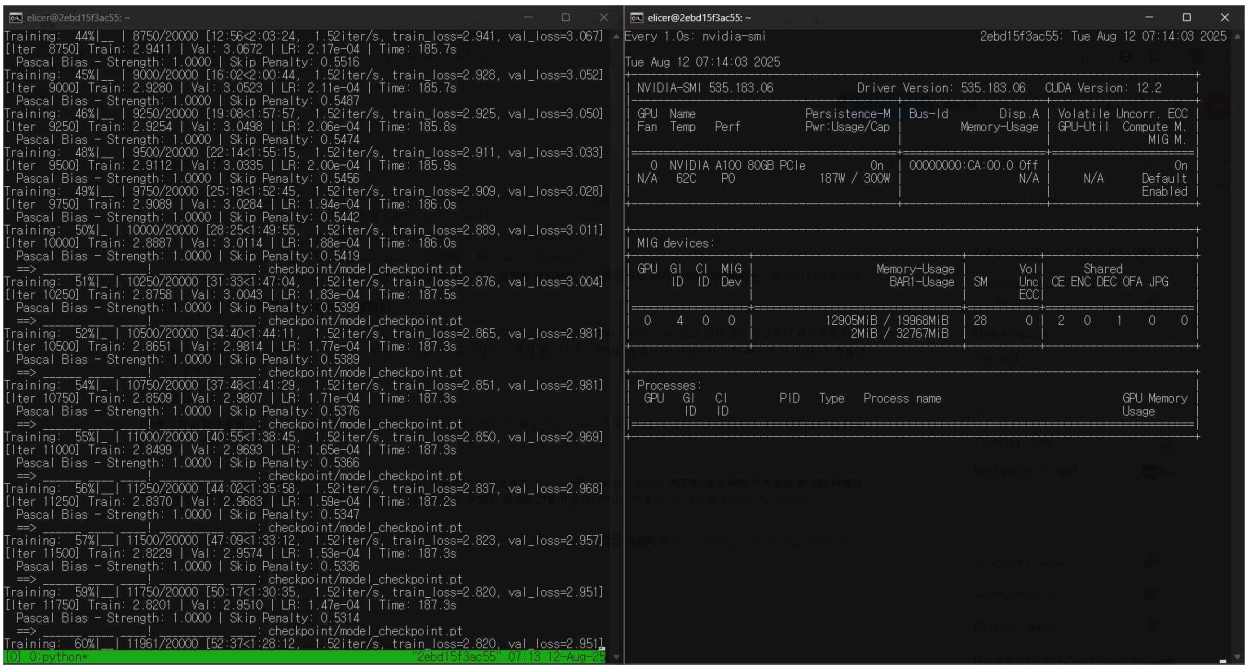
구체적 설계 및 실험

토크나이징의 경우 bpe 토크나이저를 활용해주었습니다.
WikiText-103을 95% 학습용 5% 검증용으로 분리하여 실험했으며
파스칼 피라미드의 layer = 3을 사용했습니다. 또한 파스칼 피라미드이기 때문에 변수가 3개여서 sliding-window는 3으로 설정했습니다.

torch.compile()을 사용하여 컴파일 진행 후 학습을 진행하였고
block_size=256
n_layer: int = 12
n_head: int = 12
n_embd: int = 768
pascal_n: int = 3
bias: bool = False
dropout: float = 0.1
batch_size: int = 64
gradient_accumulation_steps: int = 2
max_iters: int = 20000
learning_rate: float = 3e-4
weight_decay: float = 0.1
beta1: float = 0.9
beta2: float = 0.95
grad_clip: float = 1.0
warmup_iters: int = 2000
bias_warmup_iters: int = 4000
lr_decay_iters: int = 8000
min_lr_ratio: float = 0.1
eval_interval: int = 250
eval_iters: int = 100

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

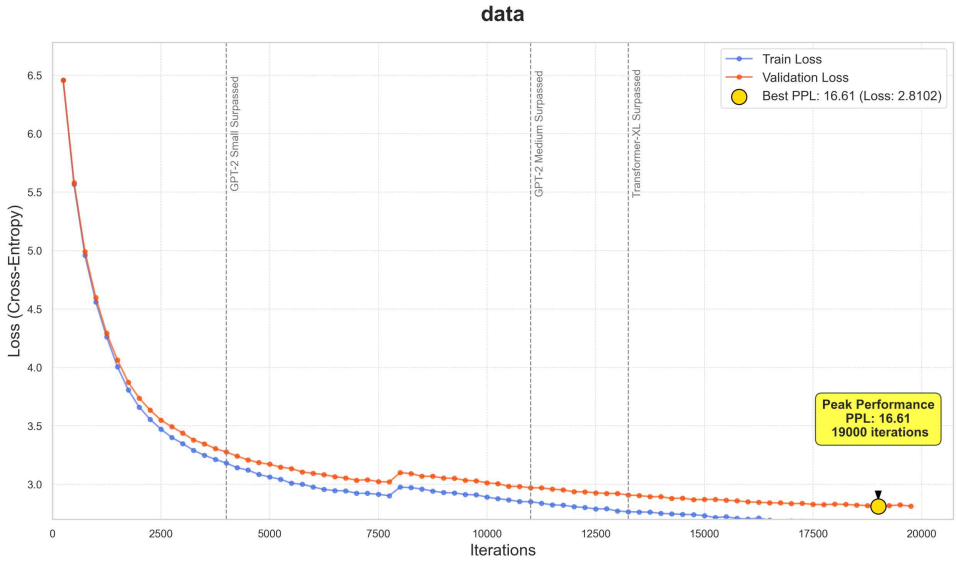
log_interval: int = 10
window_size = 3
이러한 세팅으로 했습니다.



(학습하는 과정 스크린샷)
좌측은 TMUX사용으로 인해 한글 폰트는 깨졌습니다.

결과

A100 GPU에서 GPU 메모리 12905Mb를 사용하여 학습한 결과
약 250분 (4시간 10분) 학습시켰으며(20000이터레이션)



검증손실 값이 19000 ITER에서 2.8102로 예측 PPL 값은 16.61이며 이후 과적합으로 인해 검증 손실 값이 증가하여 학습을 종료했습니다. 또한 위 모델로 WikiText2 데이터 세트와 WikiText103 데이터 세트로 PPL을 측정한 결과 각각 15.1936, 15.4020이라는 값이 관측되었으며 lambada 데이터 세트를 사용한 마지막 단어 맞추는 문제에서는 5,153문제 중 3,752문제를 맞춰 72.81%의 정답률을 보았습니다.

본 연구는 위키텍스트-103 데이터셋에 국한되었으며, 다른 도메인 데이터셋에서의 일반화 가능성은 추가 검증이 필요합니다. 또한, 파스칼 편향의 최적 파라미터 설정에 대한 심층 분석이 요구됩니다. 향후 연구에서는 다양한 데이터셋과 하드웨어 환경에서 실험을 확장하고, 다른 수학적 구조를 활용한 어텐션 메커니즘을 탐구할 계획입니다.

또한 언어 생성에 있어서 수학적 조합이 어떤 연관이 있는지 구체적으로 증명하지는 못했습니다. 하지만 언어학습에 있어서 이점을 보여주어 이를 더욱 심도있게 분석해봐야 할 것 같습니다.

초기 문제점 및 해결방안

문제점1: 바이어스 행렬을 매번 새로 계산
어텐션 레이어가 호출될 때마다 바이어스 행렬 B를 새로 생성하여
모든 레이어, 모든 순전파 단계에서 반복 계산
시퀀스 길이 ↑ + 모델 깊이 ↑ = 기하급수적 계산 부하 증가함
결과: 학습 속도 심각한 저하, 병목 현상이 일어남

문제점2: 처음부터 최대 강도의 고정 바이어스 적용
학습 시작 즉시 강력한 편향 부여
모델의 자유로운 패턴 학습 방해함
과도한 규제 효과가 일어났음
결과: 모델 수렴 실패, 손실 발산(NaN), 극도로 불안정한 학습이 일어남

문제점3: 목표와 정반대 방향으로 구현함
전체 시퀀스의 평균으로 전역 바이어스 생성
모든 어텐션 스코어에 동일한 값 적용
지역적 관계 강화 목표 vs 전역 정보만 활용 구현
결과: n-gram 관계 학습에 전혀 도움 안 됨, 완전히 잘못된 접근이였었음

문제점4: 트랜스포머 작동 원리 무시
어텐션 메커니즘 작동 전에 위치 정보 미리 섞음
원본 순서와 위치 정보 왜곡
트랜스포머의 문맥 학습 능력 파괴
결과: 모델의 핵심 기능인 문맥 파악 능력 심각한 손상

문제 해결 모델
계산 효율성: 바이어스 캐싱 유지
학습 안정성: 적응형 강도 조절 (웜업 스케줄)
초기: 바이어스 강도 ≈ 0
점진적으로 1까지 증가
모델이 먼저 자유롭게 학습 → 점차 가이드 제공
유연성: 학습 가능한 페널티 파라미터

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

소스 코드

```
import os
import time
import pickle
import math
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.amp import GradScaler, autocast
from tqdm import tqdm
from dataclasses import dataclass
from datasets import load_dataset
import re
import random
from tokenizers import Tokenizer, normalizers, pre_tokenizers
from tokenizers.models import BPE
from tokenizers.trainers import BpeTrainer

@dataclass
class DataConfig:
    dataset_name: str ="wikitext"  # 사용할 데이터셋 이름
    dataset_subset: str ="wikitext-103-raw-v1"  # 데이터셋의 서브셋
    data_dir: str ="data"  # 데이터 저장 디렉토리
    clean_text_path: str =os.path.join(data_dir, "input_cleaned.txt")  # 정제된 텍스트 파일 경로
    tokenizer_path: str =os.path.join(data_dir, "bpe_tokenizer.json")  # 토크나이저 파일 경로
    test_size: float =0.05  # 검증 데이터 비율
    random_state: int =42  # 재현성을 위한 랜덤 시드

@dataclass
class ModelConfig:
    block_size: int =256  # 시퀀스 길이
    vocab_size: int =8192  # 어휘 크기
    n_layer: int =12  # 트랜스포머 레이어 수
    n_head: int =12  # 어텐션 헤드 수
    n_embd: int =768  # 임베딩 차원
    pascal_n: int =3  # 파스칼 바이어스 생성에 사용될 n값
    bias: bool =False  # 모델 내 선형 계층의 바이어스 사용 여부
    dropout: float =0.1  # 드롭아웃 비율

@dataclass
class TrainConfig:
    data_dir: str ="data"  # 데이터 디렉토리
    ckpt_dir: str ="checkpoint"  # 체크포인트 저장 디렉토리
    ckpt_path: str =os.path.join(ckpt_dir, "model_checkpoint.pt")  # 체크포인트 파일 경로
    device: str ='cuda'if torch.cuda.is_available() else 'cpu'  # 훈련 장치
    batch_size: int =64  # 배치 크기
    gradient_accumulation_steps: int =2  # 그래디언트 축적 스텝
    max_iters: int =20000  # 최대 훈련 반복 횟수
    learning_rate: float =3e-4  # 학습률
    weight_decay: float =0.1  # 가중치 감쇠
    beta1: float =0.9  # AdamW 옵티마이저의 beta1
    beta2: float =0.95  # AdamW 옵티마이저의 beta2
    grad_clip: float =1.0  # 그래디언트 클리핑 값
    warmup_iters: int =2000  # 학습률 웜업 반복 횟수
    bias_warmup_iters: int =4000  # 파스칼 바이어스 웜업 반복 횟수
    lr_decay_iters: int =8000  # 학습률 감소 반복 횟수
    min_lr_ratio: float =0.1  # 최소 학습률 비율
    eval_interval: int =250  # 평가 간격
```

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

```
    eval_iters: int =100  # 평가 반복 횟수
    log_interval: int =10  # 로그 출력 간격
# 데이터셋을 다운로드하고 정제하여 파일로 저장하는 함수
def prepare_data_main(config: DataConfig):
    # 텍스트에서 불필요한 개행 문자를 제거하는 내부 함수
    def clean_text(text):
        text =re.sub(r'\n{2,}', '\n', text).strip()
        return text
    print(f"'{config.dataset_name}({config.dataset_subset})' download dataset")
    output_dir =os.path.dirname(config.clean_text_path)
    os.makedirs(output_dir, exist_ok=True)

    try:
            dataset  =load_dataset(config.dataset_name,  config.dataset_subset,  split='train',
streaming=True)
    except Exception as e:
        print(f"err load data: {e}")
        return
    with open(config.clean_text_path, 'w', encoding='utf-8') as outfile:
        count =0
        for example in tqdm(dataset, desc="saving data", unit=" docs"):
            text =example.get('text', '')
            if text:
                cleaned_text =clean_text(text)
                if len(cleaned_text) >50:
                    outfile.write(cleaned_text +"\n")
                    count +=1

def run_memory_safe_tokenization(d_config: DataConfig, m_config: ModelConfig):
    if not os.path.exists(d_config.clean_text_path):
        print(f"'{d_config.clean_text_path}' file not exists")
        return

    if os.path.exists(d_config.tokenizer_path):
        tokenizer =Tokenizer.from_file(d_config.tokenizer_path)
    else:
        with open(d_config.clean_text_path, 'r', encoding='utf-8') as f:
            text_iterator =(line for line in f)
            tokenizer =Tokenizer(BPE(unk_token="[UNK]"))
            tokenizer.normalizer =normalizers.NFKC()
            tokenizer.pre_tokenizer =pre_tokenizers.Whitespace()
                        trainer   =BpeTrainer(vocab_size=m_config.vocab_size,   min_frequency=2,
special_tokens=["[PAD]", "[UNK]", "[EOS]"])
            tokenizer.train_from_iterator(text_iterator, trainer=trainer)
            tokenizer.save(d_config.tokenizer_path)
            print(f"tokenizer trained & saved: {d_config.tokenizer_path}")

    data_dir =os.path.dirname(d_config.clean_text_path)
    train_bin_path =os.path.join(data_dir, 'train.bin')
    val_bin_path =os.path.join(data_dir, 'val.bin')
    train_token_count, val_token_count =0, 0
    with open(d_config.clean_text_path, 'r', encoding='utf-8') as f_in, \
         open(train_bin_path, 'wb') as f_train, \
         open(val_bin_path, 'wb') as f_val:
        for line in tqdm(f_in, desc="incoding & data split", unit=" lines"):
            if not line.strip(): continue
            encoded =tokenizer.encode(line)
            ids =encoded.ids
            if random.random() <d_config.test_size:
                f_val.write(np.array(ids, dtype=np.uint32).tobytes())
                val_token_count +=len(ids)
```

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

```
            else:
                f_train.write(np.array(ids, dtype=np.uint32).tobytes())
                train_token_count +=len(ids)
    # 메타데이터 저장 (어휘 크기, 토크나이저 경로 등)
     meta ={'vocab_size': tokenizer.get_vocab_size(), 'tokenizer_path': d_config.tokenizer_path,
'eos_token_id': tokenizer.token_to_id("[EOS]")}
    with open(os.path.join(data_dir, 'meta.pkl'), 'wb') as f:
        pickle.dump(meta, f)

    print(f"val data: {train_token_count:,}")
    print(f"train data: {val_token_count:,}")
# 파스칼 삼각형 계수를 기반으로 어텐션 바이어스를 동적으로 생성하는 모듈
class AdaptivePascalBiasGenerator(nn.Module):
    def __init__(self, n: int, max_seq_len: int, bias_warmup_steps: int):
        super().__init__()
        self.n, self.max_seq_len, self.bias_warmup_steps =n, max_seq_len, bias_warmup_steps
         self.skip_penalty_param =nn.Parameter(torch.tensor(-1.5))  # 건너뛰기 연결에 대한 페널티 파
라미터
        self.pascal_coeffs_map =self._calculate_multinomial_coeffs(n)  # 다항 계수 계산
        self.base_bias_1_gram =math.log(1 +self.pascal_coeffs_map.get((n, 0, 0), 1))
        self.base_bias_2_gram =math.log(1 +self.pascal_coeffs_map.get((n-1, 1, 0), n))
        self._bias_cache ={}  # 계산된 바이어스 캐싱
        self.register_buffer('training_step', torch.tensor(0, dtype=torch.long))
    # 다항 계수(파스칼 삼각형 일반화)를 계산하는 함수
    def _calculate_multinomial_coeffs(self, n: int) -> dict:
        coeffs, f ={}, math.factorial
        for k1 in range(n +1):
            for k2 in range(n -k1 +1):
                k3 =n -k1 -k2
                coeffs[(k1, k2, k3)] =f(n) //(f(k1) *f(k2) *f(k3))
        return coeffs
    # 훈련 스텝에 따라 바이어스의 강도를 조절하는 함수
    def get_adaptive_bias_strength(self) -> torch.Tensor:
        if self.training_step <self.bias_warmup_steps:
            strength =self.training_step.float() /self.bias_warmup_steps
        else:
            strength =torch.tensor(1.0, device=self.training_step.device)
        min_strength =0.01
        return min_strength +strength *(1.0 -min_strength)
    # 순전파 함수: 시퀀스 길이에 맞는 어텐션 바이어스 행렬을 생성
    def forward(self, seq_len: int, device: torch.device) -> torch.Tensor:
        if self.training: self.training_step +=1

        cache_key =f"{seq_len}_{self.n}"
        if cache_key in self._bias_cache:
            base_bias =self._bias_cache[cache_key].to(device, non_blocking=True)
        else:
            base_bias =self._compute_base_bias_matrix(seq_len, device)
            if len(self._bias_cache) <20:
                 self._bias_cache[cache_key] =base_bias.cpu()
        adaptive_strength =self.get_adaptive_bias_strength()
        penalty_factor =torch.exp(-torch.exp(self.skip_penalty_param))
        final_bias =self._apply_skip_penalty(base_bias.clone(), seq_len, penalty_factor, device)
        return final_bias *adaptive_strength
    # 기본 바이어스 행렬을 계산하는 함수
    def _compute_base_bias_matrix(self, seq_len: int, device: torch.device) -> torch.Tensor:
        bias_values ={}
        window_size =3  # 1-gram, 2-gram, 3-gram(skip) 관계를 고려하기 위한 윈도우
        for i in range(max(0, seq_len -window_size +1)):
            positions =[i +j for j in range(min(window_size, seq_len -i))]
            for pos in positions:
```

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

```
                if pos <seq_len: bias_values[(pos, pos)] =self.base_bias_1_gram  # 1-gram (self)
            for j in range(len(positions)):
                for k in range(j +1, len(positions)):
                    if positions[j] <seq_len and positions[k] <seq_len:
                        pair =tuple(sorted([positions[j], positions[k]]))
                        bias_values[pair] =self.base_bias_2_gram  # 2-gram (adjacent)

        B =torch.zeros(seq_len, seq_len, dtype=torch.float32, device=device)
        for(i, j), bias in bias_values.items():
            B[i, j] =bias
            if i !=j: B[j, i] =bias
        return B
    # 건너뛰기 연결(skip-gram)에 대한 페널티를 적용하는 함수
    def _apply_skip_penalty(self, B: torch.Tensor, seq_len: int, penalty_factor: float, device:
torch.device) -> torch.Tensor:
        if seq_len >=3:
            skip_bias =self.base_bias_2_gram *penalty_factor
            idx =torch.arange(0, seq_len -2, device=device)
            B[idx, idx +2], B[idx +2, idx] =skip_bias, skip_bias  # 2칸 떨어진 위치에 페널티 적용
        return B
# 파스칼 바이어스를 사용하는 어텐션 모듈
class PascalBiasedAttention(nn.Module):
    def __init__(self, config: ModelConfig, bias_generator: AdaptivePascalBiasGenerator):
        super().__init__()
        assert config.n_embd %config.n_head ==0
        self.c_attn =nn.Linear(config.n_embd, 3 *config.n_embd, bias=config.bias)
        self.c_proj =nn.Linear(config.n_embd, config.n_embd, bias=config.bias)
        self.n_head, self.n_embd =config.n_head, config.n_embd
        self.bias_generator =bias_generator
        self.dropout =nn.Dropout(config.dropout)
        # 인과적 마스킹(causal mask)을 위한 버퍼 등록
                self.register_buffer("causal_mask",    torch.tril(torch.ones(config.block_size,
config.block_size)).view(1, 1, config.block_size, config.block_size))
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        B, T, C =x.size()
        q, k, v =self.c_attn(x).split(self.n_embd, dim=2)
        k =k.view(B, T, self.n_head, C //self.n_head).transpose(1, 2)
        q =q.view(B, T, self.n_head, C //self.n_head).transpose(1, 2)
        v =v.view(B, T, self.n_head, C //self.n_head).transpose(1, 2)

        att =(q @k.transpose(-2, -1)) *(1.0 /math.sqrt(k.size(-1)))
        # 계산된 어텐션 스코어에 파스칼 바이어스 추가
        att =att +self.bias_generator(T, x.device).unsqueeze(0).unsqueeze(0)
        att =att.masked_fill(self.causal_mask[:, :, :T, :T] ==0, float('-inf'))
        att =F.softmax(att, dim=-1)
        att =self.dropout(att)

        out =(att @v).transpose(1, 2).contiguous().view(B, T, C)
        return self.c_proj(out)
# 트랜스포머 블록
class Block(nn.Module):
    def __init__(self, config: ModelConfig, bias_generator: AdaptivePascalBiasGenerator):
        super().__init__()
        self.ln_1 =nn.LayerNorm(config.n_embd)
        self.attn =PascalBiasedAttention(config, bias_generator)
        self.ln_2 =nn.LayerNorm(config.n_embd)
        self.mlp =nn.Sequential(
            nn.Linear(config.n_embd, 4 *config.n_embd, bias=config.bias),
            nn.GELU(),
            nn.Dropout(config.dropout),
            nn.Linear(4 *config.n_embd, config.n_embd, bias=config.bias),
```

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

```
            nn.Dropout(config.dropout)
)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x =x +self.attn(self.ln_1(x))  # Multi-Head Attention + Residual
        x =x +self.mlp(self.ln_2(x))   # MLP + Residual
        return x
# 파스칼 바이어스 어텐션을 사용하는 전체 언어 모델
class PascalLanguageModel(nn.Module):
    def __init__(self, m_config: ModelConfig, t_config: TrainConfig):
        super().__init__()
        self.config =m_config
                        self.bias_generator     =AdaptivePascalBiasGenerator(n=m_config.pascal_n,
max_seq_len=m_config.block_size, bias_warmup_steps=t_config.bias_warmup_iters)

        self.transformer =nn.ModuleDict(dict(
            wte=nn.Embedding(m_config.vocab_size, m_config.n_embd),  # Token Embedding
            wpe=nn.Embedding(m_config.block_size, m_config.n_embd),  # Positional Embedding
            drop=nn.Dropout(m_config.dropout),
                            h=nn.ModuleList([Block(m_config,    self.bias_generator)    for    _in
range(m_config.n_layer)]),
            ln_f=nn.LayerNorm(m_config.n_embd)
))
        self.lm_head =nn.Linear(m_config.n_embd, m_config.vocab_size, bias=False)
        self.transformer.wte.weight =self.lm_head.weight  # 가중치 공유
        self.register_buffer('pos', torch.arange(m_config.block_size, dtype=torch.long))
        self.apply(self._init_weights)
    # 가중치 초기화 함수
    def _init_weights(self, module):
        if isinstance(module, nn.Linear):
            torch.nn.init.normal_(module.weight, mean=0.0, std=0.02)
            if module.bias is not None: torch.nn.init.zeros_(module.bias)
        elif isinstance(module, nn.Embedding):
            torch.nn.init.normal_(module.weight, mean=0.0, std=0.02)
    # 현재 바이어스 정보를 반환하는 함수
    def get_bias_info(self) -> dict:
        return{'bias_strength': self.bias_generator.get_adaptive_bias_strength().item(),
                                                                                'skip_penalty':
torch.exp(-torch.exp(self.bias_generator.skip_penalty_param)).item(),
                'training_step': self.bias_generator.training_step.item()}
    def forward(self, idx: torch.Tensor, targets: torch.Tensor =None):
        b, t =idx.size()
        pos =self.pos[:t]
        tok_emb =self.transformer.wte(idx)
        pos_emb =self.transformer.wpe(pos)
        x =self.transformer.drop(tok_emb +pos_emb)

        for block in self.transformer.h:
            x =block(x)

        x =self.transformer.ln_f(x)

        if targets is not None:
            logits =self.lm_head(x)
                    loss   =F.cross_entropy(logits.view(-1,   logits.size(-1)),   targets.view(-1),
ignore_index=-1)
        else:
            # 추론 시에는 마지막 스텝의 로짓만 계산
            logits, loss =self.lm_head(x[:, [-1], :]), None
        return logits, loss
# 모델 훈련을 위한 메인 함수
```

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

```
def train_model():
    m_config =ModelConfig()
    t_config =TrainConfig()
    os.makedirs(t_config.ckpt_dir, exist_ok=True)

    device =t_config.device
    device_type ='cuda'if 'cuda'in device else 'cpu'
    torch.backends.cuda.matmul.allow_tf32 =True
    torch.backends.cudnn.allow_tf32 =True
    # 데이터 로드
    data_dir =t_config.data_dir
    train_data =np.fromfile(os.path.join(data_dir, 'train.bin'), dtype=np.uint32)
    val_data =np.fromfile(os.path.join(data_dir, 'val.bin'), dtype=np.uint32)
    with open(os.path.join(data_dir, 'meta.pkl'), 'rb') as f:
        meta =pickle.load(f)
    m_config.vocab_size =meta['vocab_size']
    print(f"vocab_size: {m_config.vocab_size}")
    model =PascalLanguageModel(m_config, t_config).to(device)
    print(f"model-parameter: {sum(p.numel() for p in model.parameters())/1e6:.2f}M")
    try:
        model =torch.compile(model)
        print("torch.compile applied")
    except Exception as e:
        print(f"torch.compile err: {e}")

            optimizer     =torch.optim.AdamW(model.parameters(),     lr=t_config.learning_rate,
weight_decay=t_config.weight_decay, betas=(t_config.beta1, t_config.beta2))
    scaler =GradScaler(enabled=(device_type =='cuda'))
    # 학습률 스케줄러 (코사인 어닐링)
    def get_lr(it):
        if it <t_config.warmup_iters: return t_config.learning_rate *it /t_config.warmup_iters
        if it >t_config.lr_decay_iters: return t_config.learning_rate *t_config.min_lr_ratio
                    decay_ratio    =(it    -t_config.warmup_iters)    /(t_config.lr_decay_iters
-t_config.warmup_iters)
        coeff =0.5 *(1.0 +math.cos(math.pi *decay_ratio))
        min_lr =t_config.learning_rate *t_config.min_lr_ratio
        return min_lr +coeff *(t_config.learning_rate -min_lr)
    iter_num, best_val_loss =0, 1e9
    # 데이터 배치를 가져오는 함수
    def get_batch(split):
        data =train_data if split =='train'else val_data
        ix =torch.randint(len(data) -m_config.block_size, (t_config.batch_size,))
         x =torch.stack([torch.from_numpy(data[i:i+m_config.block_size].astype(np.int64)) for i in
ix])
         y =torch.stack([torch.from_numpy(data[i+1:i+1+m_config.block_size].astype(np.int64)) for
i in ix])
        if device_type =='cuda':
                return  x.pin_memory().to(device,  non_blocking=True),  y.pin_memory().to(device,
non_blocking=True)
        return x.to(device), y.to(device)
    # 손실을 평가하는 함수
    @torch.no_grad()
    def estimate_loss():
        out ={}
        model.eval()
        for split in['train', 'val']:
            losses =torch.zeros(t_config.eval_iters)
                pbar  =tqdm(range(t_config.eval_iters),  desc=f"Evaluating  {split}",  leave=False,
unit="batch")
            for k in pbar:
                X, Y =get_batch(split)
```

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

```
                with autocast(device_type=device_type, dtype=torch.bfloat16):
                    _, loss =model(X, Y)
                losses[k] =loss.item()
            out[split] =losses.mean()
        model.train()
        return out
    X, Y =get_batch('train')
    start_time =time.time()
    print(f"{t_config.max_iters}training")
    main_pbar =tqdm(range(iter_num, t_config.max_iters), desc="Training", unit="iter")
    # 메인 훈련 루프
    for iter_num in main_pbar:
        lr =get_lr(iter_num)
        for param_group in optimizer.param_groups: param_group['lr'] =lr
        # 주기적으로 평가 및 체크포인트 저장
        if iter_num >0 and iter_num %t_config.eval_interval ==0:
            losses =estimate_loss()
            elapsed_time =time.time() -start_time
            start_time =time.time()

            uncompiled_model =model._orig_mod if hasattr(model, '_orig_mod') else model
            bias_info =uncompiled_model.get_bias_info()

                                        main_pbar.set_postfix(train_loss=f"{losses['train']:.3f}",
val_loss=f"{losses['val']:.3f}")
            print(f"\n[Iter {iter_num:5d}] Train: {losses['train']:.4f}| Val: {losses['val']:.4f}|
LR: {lr:.2e}| Time: {elapsed_time:.1f}s")
              print(f"  Pascal Bias - Strength: {bias_info['bias_strength']:.4f}| Skip Penalty:
{bias_info['skip_penalty']:.4f}")

            if losses['val'] <best_val_loss:
                best_val_loss =losses['val']
                            checkpoint  ={'model':  uncompiled_model.state_dict(),  'optimizer':
optimizer.state_dict(), 'iter_num': iter_num, 'best_val_loss': best_val_loss}
                print(f"save ckpt: {t_config.ckpt_path}")
                torch.save(checkpoint, t_config.ckpt_path)
        optimizer.zero_grad(set_to_none=True)

        # 그래디언트 축적
        for micro_step in range(t_config.gradient_accumulation_steps):
            with autocast(device_type=device_type, dtype=torch.bfloat16):
                _, loss =model(X, Y)
                loss =loss /t_config.gradient_accumulation_steps

            if torch.isnan(loss):
                print(f"Iter {iter_num}loss NaN")
                return

            X, Y =get_batch('train')
            scaler.scale(loss).backward()

        scaler.unscale_(optimizer)
        torch.nn.utils.clip_grad_norm_(model.parameters(), t_config.grad_clip)
        scaler.step(optimizer)
        scaler.update()
    print("end")
# 스크립트 실행 지점
if __name__=='__main__':
    d_config =DataConfig()
    m_config =ModelConfig()
    # 데이터 준비
```

본 페이지부터는 본인의 특기 활동을 가장 잘 표현할 수 있도록 자유롭게 작성할 수 있습니다.

```
    if not os.path.exists(d_config.clean_text_path):
        prepare_data_main(d_config)
    else:
        print(f"'{d_config.clean_text_path}'")
    # 토크나이저 및 바이너리 데이터 준비
    data_dir =d_config.data_dir
      required_files  =[os.path.join(data_dir,  'train.bin'),  os.path.join(data_dir,  'val.bin'),
os.path.join(data_dir, 'meta.pkl')]
    if not all(os.path.exists(p) for p in required_files):
        run_memory_safe_tokenization(d_config, m_config)
    else:
        pass
    # 모델 훈련 시작
    train_model()
```