Attention Is All You Need Review

- Authors : Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. (Google Brain/Research)
- Conference : NIPS 2017
- Link : https://arxiv.org/abs/1706.03762
- Tags : `#Transformer` `#Attention` `#NLP`

1. Introduction

이 논문에서는 순차적인 계산이 필요한 RNN이나 CNN 없이 오직 Attention 메커니즘만으로 복잡한 문맥 정보를 파악하는 Transformer 구조를 제안하였다.

2. Transformer Model Architecture

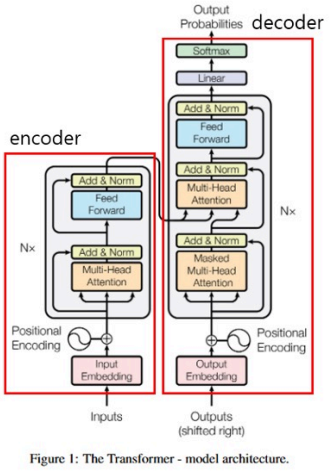


Figure 1: The Transformer - model architecture.

그림1 Transformer 아키텍쳐

<Attention is all you need> 논문에서 Transformer 모델을 설명할 때, 6개의 층으로 이루어진 Encoder와 6개의 층으로 이루어진 Decoder를 사용하였다.

[Encoder]

Encoder 1개의 Layer는 각각 Multi-Head Attention과 MLP, 두 개의 sub-layer로 구성되었다. 이러한 Layer가 Encoder에 총 6개 staking되어 있는 구조이다. 이때 Layer 간 잔차연결(residual connection)과 레이어 정규화(Layer normalization)가 이루어진다. Residual connection은 deep neural network에서 기울기 소실 문제와 성능 저하 문제를 해결하기 위한 방법이다. Layer normalization은 입력의 분포를 일정하게 유지함으로써 안정적인 학습을 가능하게 하는 효과가 있다.

그림2에서는 아래 사진과 같은 구조가 Encoder에 해당한다.

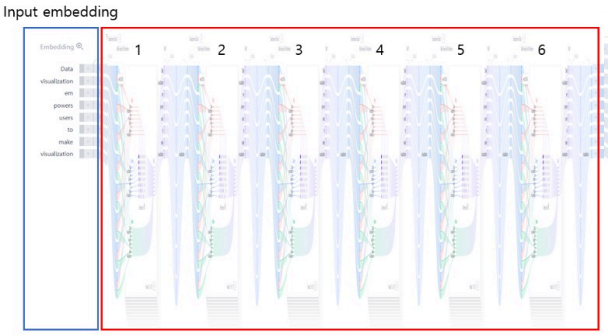


그림2 Encoder 전체 구조

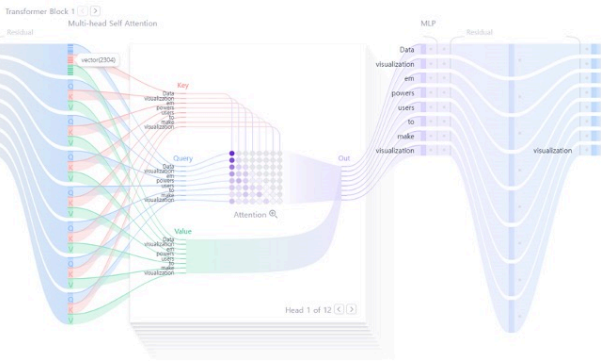


그림3 **Encoder 1개 층 구조 (Multi-Head Attention → MLP)**

[Decoder]

Decoder 1개의 Layer는 각각 Encoder의 Sub-layer인 Multi-Head Attention와 MLP에 Masked multi-head self-attention를 추가하여, 총 세 개의 sub-layer로 이루어져있다.

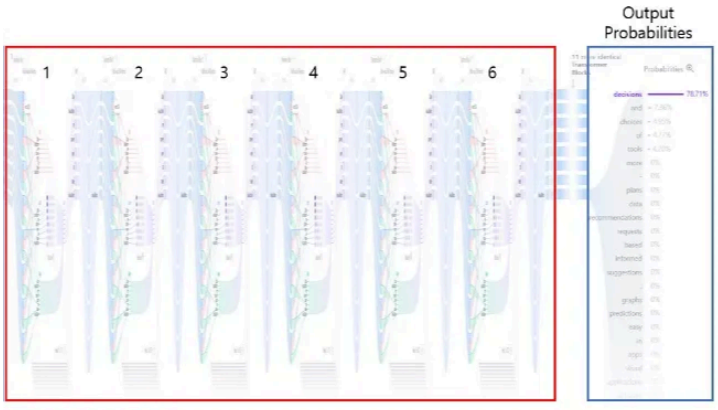


그림4 Decoder 전체 구조

[Encoder-Decoder Attention]

Encoder와 Decoder는 Decoder의 두번째 sub-layer인 multi-head self-attention Layer의 input으로 사용되면서 서로 연결되어 있다. 이 Layer에서는 Decoder의 이전 Layer의 출력

값을 Query로 사용하고, Encoder의 출력값을 Key, Value로 사용하는 attention 연산을 실행한다.

Decoder의 이전 Layer는 output embedding인데, Encoder와 마찬가지로 token embedding과 positional embedding을 진행한다. Train에서는 Decoder의 output이란 정답(y)를 말하고, Test에서는 현재 예측 시점에서 모델이 예측하여 이미 생성된 단어의 출력값들이 들어간다.

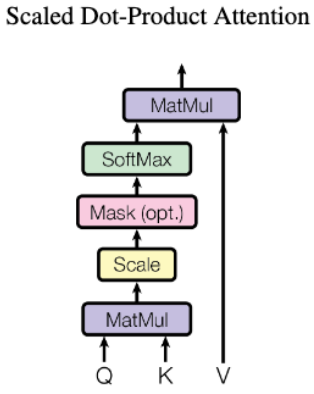


그림5 Attention 연산

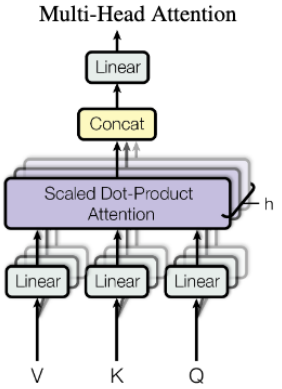


그림6 Multi-Head Attention

Attention은 같은 문장 내에서 단어들 간의 관계를 나타내는 연산으로, Query와 Key-Value 쌍의 집합을 Output으로 매핑하는 수행하는 함수를 말한다. 이때 출력은 Value의 가중합으로 계산되며, 여러 값들 중 중요하다고 판단되는 값에 더 높은 가중치를 부여해서 합산된다. 가중치는 Key와 Query 간의 호환성함수(compatibility function)를 통해 계산된다. 호환성함수는 Query가 특정 Key와 얼마나 관련성이 높은지를 의미한다. 이러한 attention 연산이 여러 개 모여서 만들어진 것이 multi-head attention이다.

3. Token embedding & Positional Embedding

Transformer의 input data의 대표적인 형태는 텍스트 데이터이다. 텍스트 데이터를 학습시키기 위해서는 단어(Token)를 벡터화 하게되는데, 사이트에 나오는 GPT-2 모델은 768차원의 벡터에 단어를 표현해서, 각 단어들은 768차원의 공간에 점으로 찍히게 된다.

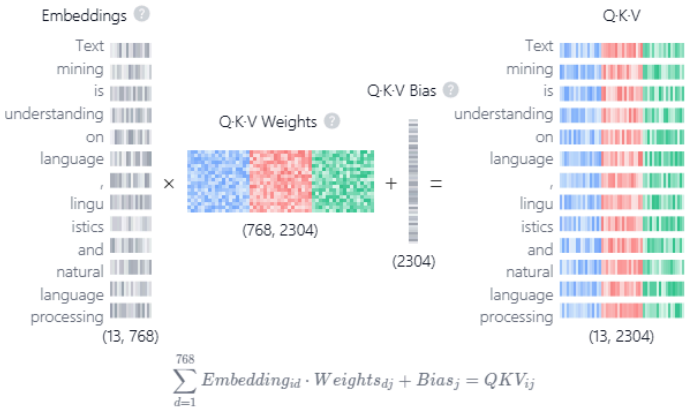
텍스트 데이터는 앞뒤의 맥락에 따라 다르게 해석될 수 있기 때문에, 단어의 순서 정보를 보존하는 것이 필요하다. 기존에 텍스트 데이터를 많이 보는 방식인 RNN, LSTM에서는 문장을 앞에서부터 순차적으로 읽기 때문에 단어의 순서를 알지만, Transformer는 병렬 연산으로 한번에 token을 읽기 때문에 순서에 대한 정보를 알수 없다. 따라서 각 token이 어느 위치에 있는 단어에 해당하는지 표식을 남겨주는 것이 positional embedding이다. positional embedding은 sine, cosine 함수를 사용해서 다양한 주파수로 위치를 인코딩하며, 그 크기는 token embedding과 동일하게 768차원을 갖는다.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

그림7 positional embedding

4. Query / Key / Value 벡터의 연결 방법



- Embedding : Input인 Embedding 단계에서는 데이터가 13개 단어가 768차원으로 존재한다. ⇒ (13, 768)
- Weights : Q,K,V 세 개의 Weight를 옆으로 이어 붙여서 한번에 연산하는 것이 효율적이라 옆으로 붙여넣은 형태 ⇒ (768, 2304)

  Wq (768, 768) / Wk (768, 768) / Wv (768, 768)

  768 * 3 = 2304

  그리고 Weight를 곱하면 각 차원마다 1개의 Bias가 생기는데, 2304차원을 곱하기 때문에 2304개의 bias가 생성된다.
- 그림8과 같은 연산을 모든 input에 대해 반복하면서 Embedding X Weight 를 연산하면 (13, 2304) 의 행렬이 나온다.

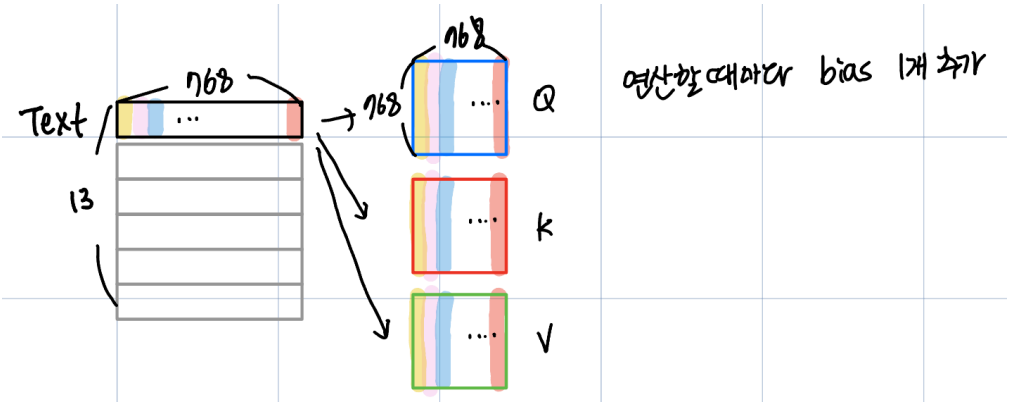
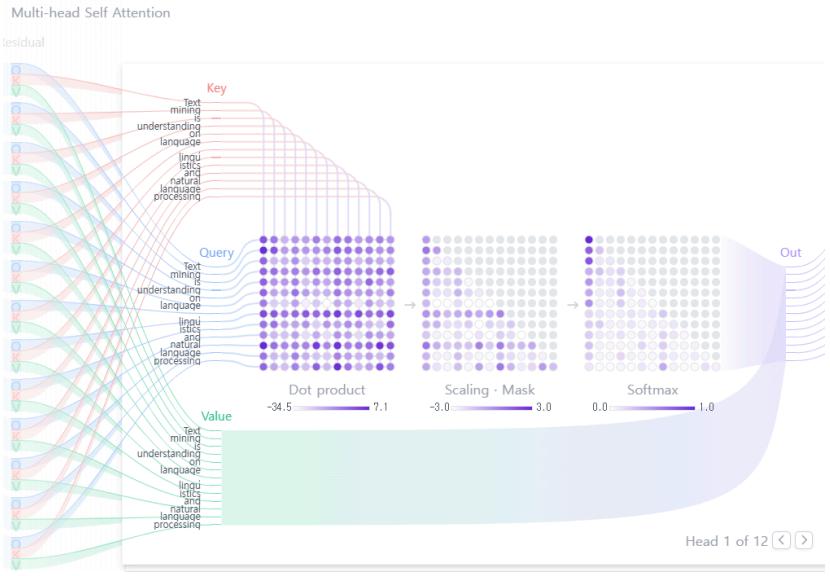


그림8 input 중 'Text'의 Q, K, V 연산 과정

- 아까 연산 효율을 위해 이어 붙였기 때문에, 이 행렬을 다시 (13, 768) * 3 의 형태로 세개로 쪼개서 각각을 Q, K, V 사용한다.

5. Scaled dot product 연산



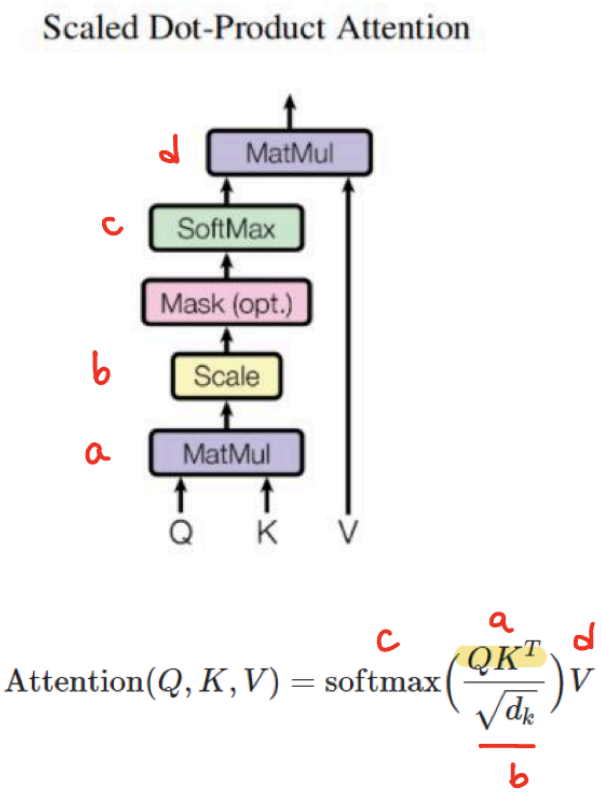


그림9 Scaled Dot-product Attention 과정

Scaled dot product 연산은 그림9 와 같은 순서로 진행된다.

- Q(13, 768)
- K(13, 768)
- V(13, 768)
- d_k = Q와 K의 차원 = 768
- d_v = V의 차원 = 768

a. QK^T (MatMul, dot-product 연산): Query 행렬과 Key 행렬 의 Transpose 를 곱하는 과정을 수행한다. 이 단계에서는 각 쿼리가 모든 키와 얼마나 관련이 있는지를 나타내는 유사도 점수를 계산하며, 유사도 점수가 높을수록 쿼리와 키의 연관성이 높다는 의미이다.

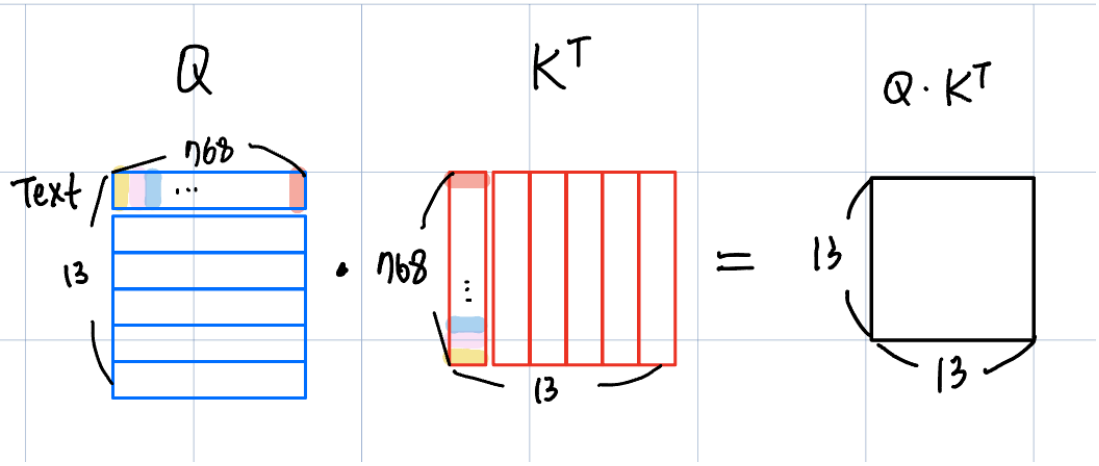


그림10 Q와 K Transpose의 dot-product 연산 과정

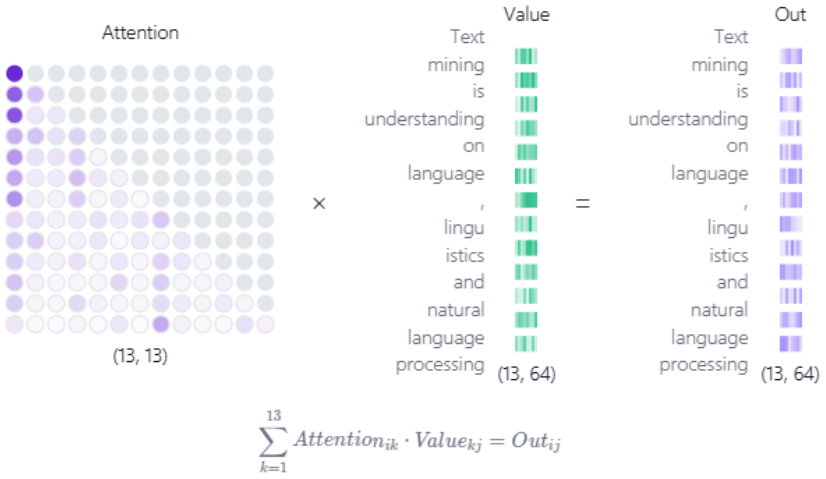
b. $QK^T/\sqrt{d_k}$ (Scale): 계산된 유사도 점수를 Q와 K의 차원의 제곱근인 $\sqrt{d_k}$로 나눈다. Q와 K의 차원이 커질수록 Softmax 함수가 0 또는 1처럼 극단적인 값으로 치우치게 되어, 기울기가 매우 작아지는 현상이 있어서 기울기소실 문제가 발생할 수 있다. 따라서 $\sqrt{d_k}$로 나누어 소프트맥스 함수의 input이 적절한 범위 내에 있도록 하는 효과가 있다.

c. $softmax(QK^T/\sqrt{d_k})$ (Softmax 함수 적용) : b에서 Scaling한 결과에 소프트맥스 함수를 적용하여 각 값에 대한 가중치를 얻는다.

d. $softmax(QK^T/\sqrt{d_k})\cdot V$: 소프트맥스를 통해 얻은 가중치를 V 행렬에 곱하여 최종 출력값을 계산한다. ⇒ 이 과정이 5번 문제에 나오는 Multi-head attention 그림에 표시되어 있다.

e. (추가) Mask 옵션 : GPT-2 모델에서 Masked attention은 Decoder에서 지금 시점보다 미래에 나오는 단어를 미리 보지 못하도록 막는 Look-ahead Mask를 사용한다. 따라서 Attention score matrix에서 대각선 위쪽 부분이 모두 $-\infty$로 채워져서 해당 부분의 확률값이 0이 되면서 가중치가 0이 되면서 정보를 전달하지 않게 된다.

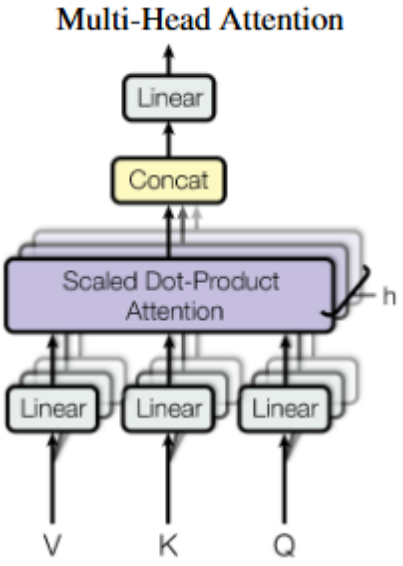
6. Multi-head attention 연산



$$\sum_{k=1}^{13} Attention_{ik} \cdot Value_{kj} = Out_{ij}$$

4번 문제의 d 과정에서 $softmax(QK^T/\sqrt{d_k})\cdot V$ 연산을 수행하고, 소프트맥스를 통해 얻은 가중치를 V 행렬에 곱하여 최종 출력값을 계산했다. 이러한 Attention을 하나로 실행하는 것이 아니라 여러 개의 작은 Attention으로 병렬 실행하게 되면, 다양한 종류의 관점으로 학습을 할 수 있게 된다. GPT-2 모델에서는 총 12개의 Attention head를 사용하는데, 모델의 embedding 차원은 768차원이었기 때문에 12개의 attention head를 사용하면 하나의 attention head당 768 / 12 = 64 차원의 정보를 처리하게 된다. 따라서 어떤 head는 문법, 어떤 head는 의미, 어떤 head는 거리 등에 집중하게 되면서 입체적인 관점으로 문장을 이해할 수 있게 된다.

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



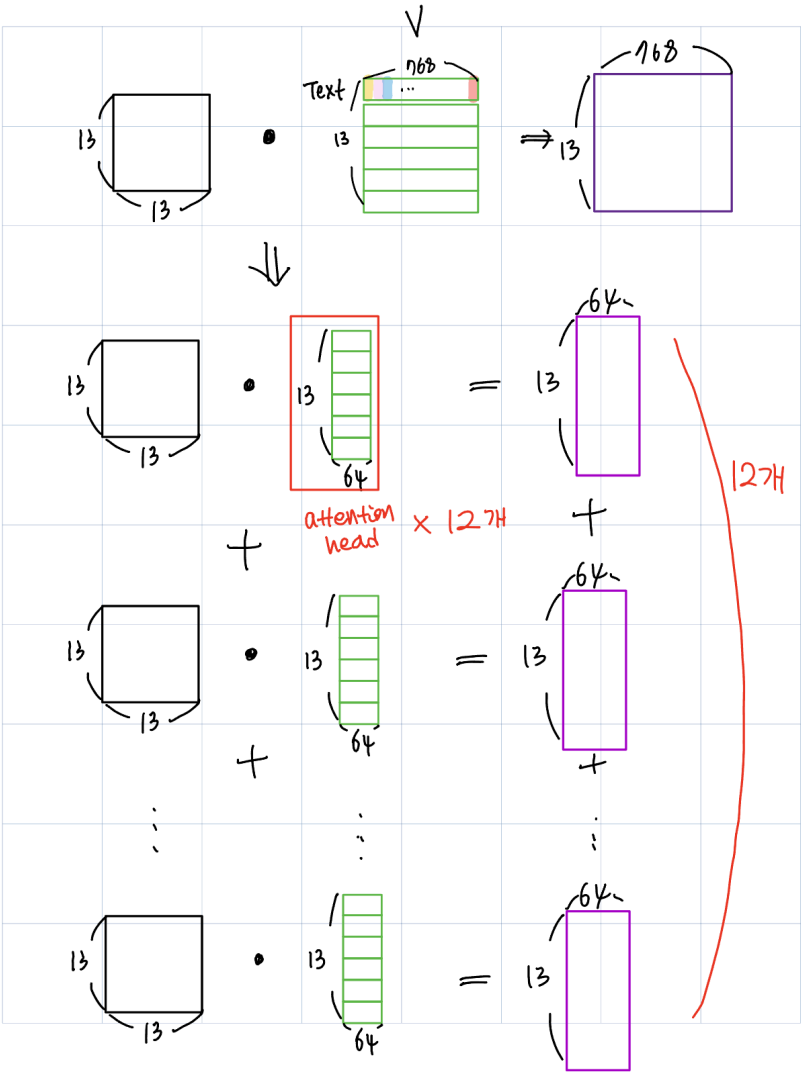


그림11 Multi-head Attention의 병렬 연산이 이루어지는 모습을 도식화

7. Residual 연산

MLP(Feed Forward Networks)에서는 선형변환 → GeLU → 선형변환 의 순으로 연산이 진행된다. 첫번째 선형변환에서는 Attention Layer를 input x로 받아서 $xW_1 + b_1$ 으로 선형변환을 진행함으로써 더 높은 차원으로 매핑한다. GPT-2에서는 입력값을 3072 / 768 = 4배 더 큰 차원으로 매핑하게 된다. 따라서 차원이 더 커지기 때문에 더 풍부한 표현력을 얻을 수 있게 된다.

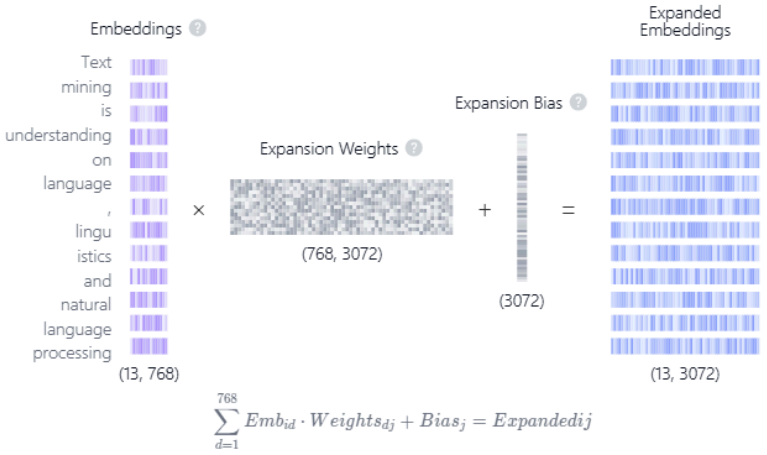


그림12 768차원의 input을 4배 더 큰 3072차원으로 팽창하는 선형변환

그 다음에는 비선형성을 추가하는 GeLU 함수를 적용하고 그 결과를 다시 $GeLU(xW_1 + b_1)W_2 + b_2$ 로 선형변환하여 원래의 차원으로 돌아오도록 축소한다.

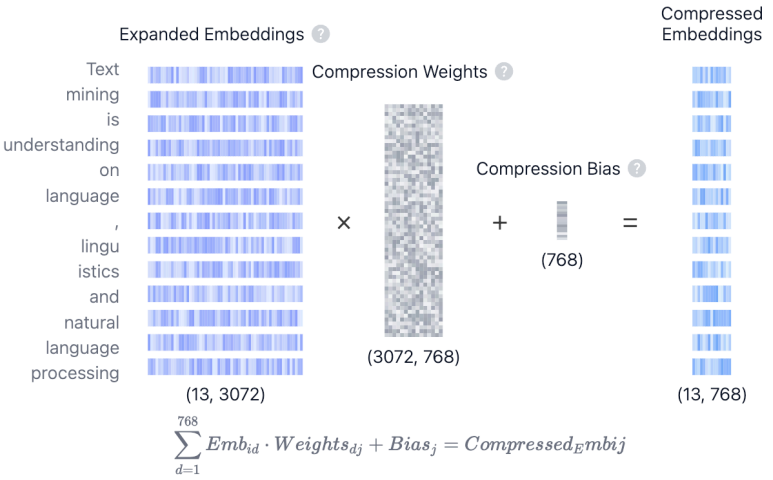
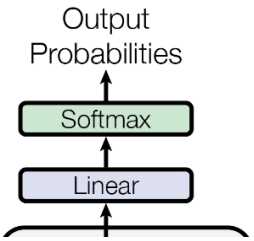
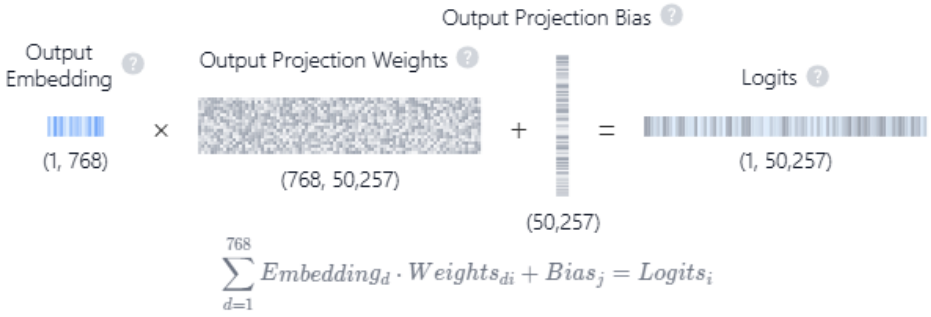


그림13 GeLU함수가 적용된 3072차원의 input을 다시 768 차원으로 축소

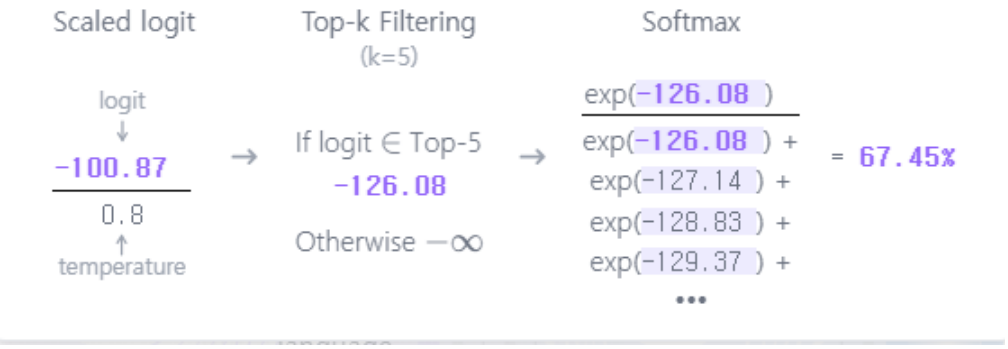
8. Logit 연산

Logit 연산은 decoder의 Linear에 해당하는 연산을 수행하는 과정이다. Logit 연산이란 Output embedding에 Output Projection Weight를 곱하고 bias를 더하는 선형변환을 수행한다. Logit 연산에서는 input값이 어떤 어휘(토큰)에 어떤 점수를 갖는지를 계산하게 된다. 따라서 모든 어휘(토큰)에 대한 점수를 알아야하기 때문에 Weight의 수는 모델의 어휘 수만큼 갖게 된다. GPT-2 논문에서는 Output Projection Layer의 어휘 크기가 50257개이다. 따라서 Weight가 50257차원으로 확장하는 역할을 한다.





9. 마지막 출력 작업



- Scaled Logit : 이 단계에서는 앞에 7번에서 진행한 Logit을 Temperature로 나눈다. 그림에서는 Temperature를 0.8로 지정하였다.
  - Temperature가 1보다 크면 Logit의 절댓값이 더 작아진다. 따라서 Logit 값들 사이의 차이가 줄어들어서 전반적으로 Logit 값이 비슷해지는 결과가 있다. 따라서 점수가 낮은 단어들도 선택될 가능성이 높아지는 효과가 있다. 이 경우에는 더 random하고 창의적인 Text가 생성될 수 있지만, 의미없는 내용이 나올 가능성도 커진다.
  - 반대로 Temperature가 1보다 작으면 Logit의 절댓값이 더 커지기 때문에 Logit 값들 사이의 차이가 더 커진다. 따라서 점수가 높은 단어가 선택될 가능성이 더 높아지고 점수가 낮은 단어들은 거의 선택되지 않게 된다. 즉, Temperature를 1보다 작은 값으로 지정하면 일반적이고 일관적인 내용이 나오게 될 것이다.
- Top-k random sampling : 모델이 항상 가장 확률이 높은 단어를 선택하게 될 경우 반복적이고 지루한 텍스트를 생성할 수 있다. 이를 해소하기 위해 확률에 따라 무작위로 샘플링을 하고 단어를 고르게 될 경우 관련이 없거나 의미가 없는 단어가 튀어나올 수 있다. 따라서 모

델의 엉뚱한 단어를 고르지 않으면서 표현력을 높이기 위해 Top-k random sampling 방식을 사용한다. 그림에서는 k=5로 설정되어 있기 때문에 Logit 값 상위 5개 까지는 그대로 점수를 남기고 나머지 50257 - 5 = 50252개는 $-\infty$로 채운다.

- Softmax : 위에서 Top-k random sampling을 한 결과를 softmax에 넣으면 $exp(-\infty) = 0$ 이라서 Top-k의 logit 값 5개만 softmax 함수에 넣어서 연산한 확률 값과 같다.

10. 결론

- Multi-Headed Self-Attention으로 재귀성을 제거함으로써, 시퀀스 처리 과정에서 순차적인 계산의 제약에서 벗어날 수 있다.
- 병렬연산으로 RNN와 CNN 기반의 기존 아키텍쳐보다 훨씬 빠르게 훈련할 수 있다.
- 우수한 번역성능
  - WMT 2014 영어-독일어 번역 태스크: Transformer의 최적 모델(Transformer (big))은 기존의 가장 우수한 모델(앙상블 모델 포함)보다 2.0 BLEU 이상 뛰어난 28.4 BLEU 점수를 달성하며 SOTA 성능 확립
  - WMT 2014 영어-프랑스어 번역 태스크: 기존 단일 모델 중 최고 성능인 41.8 BLEU 점수를 달성
- 비용 효율성 : 기존 최고 성능 모델들에 비해 훨씬 적은 Training Cost로 뛰어난 성능을 달성